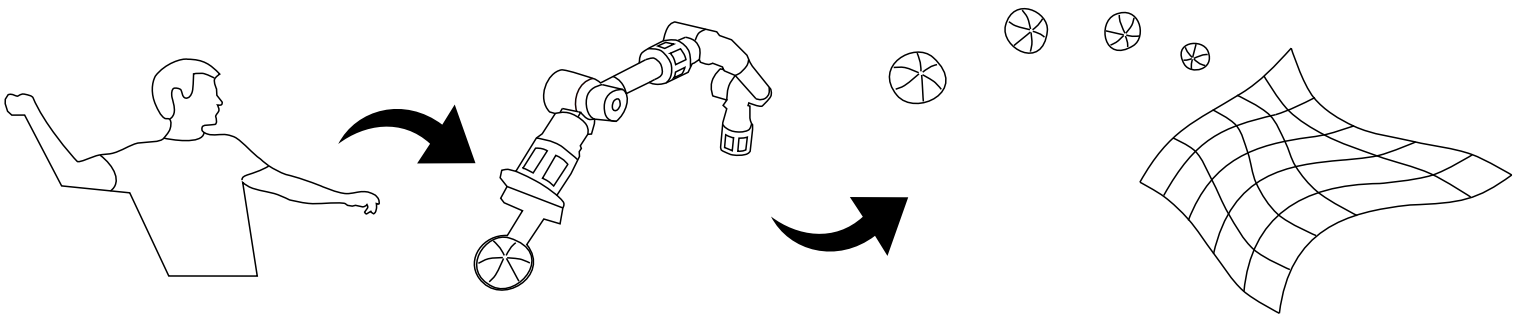
Learning and Generalizing Behaviors for Robots from Human Demonstration

von Alexander Fabisch



Dissertation

zur Erlangung des Grades eines

Doktors der Ingenieurwissenschaften (Dr.-Ing.)

Vorgelegt im Fachbereich 3
(Mathematik und Informatik)
der Universität Bremen
im Oktober 2020

Datum des Promotionskolloquiums: 03.12.2012

Gutachter: Prof. Dr. Dr. h. c. Frank Kirchner
Universität Bremen

Prof. Constantin A. Rothkopf, PhD
Technische Universität Darmstadt

Contents

Acknowledgments

I would like to apologize for statements like “artificial intelligence won’t ever work” or “I do not believe that reinforcement learning has a future”. A dissertation is a journey that sometimes drives sane people to the edge of insanity. I would like to thank Mario Michael Krell, Constantin Bergatt, Marc Otto, Marc Tabie, Kai von Szadkowski, Martin Schröer, and Frank Kirchner for encouraging me to continue this journey and to prove these statements wrong.

I would like to thank Marc Tabie, Marc Otto, Mario Michael Krell, Matias Alejandro Valdenegro Toro, Lisa Gutzeit, Hendrik Wöhrle, Bilal Wehbe, José de Gea Fernández, and Thomas M. Roehr for their valuable feedback during research, work and writing this thesis as well as Verena Fabisch for finding numerous mistakes in the text at hand.

I would like to thank particularly Yohannes Kassahun and Jan Hendrik Metzen for being mentors at the beginning of my research, Constantin Bergatt and Manuel Meder for providing simulation environments in which I could let robots learn behaviors, Mario Michael Krell for collaboration on deriving the optimization algorithm for the positive upper boundary support vector estimation (PUBSVE), Lisa Gutzeit for providing a library for trajectory segmentation that works really well and collaborating on recording a large amount of motion data, Marc Otto for collaboration on robotic throwing and on researching which other behaviors could be learned, Christoph Petzoldt for contributing a structured and application-oriented perspective, Hendrik Wiese for implementing a procedure for automatic labeling of motion capture markers and ROS support, Jonas Hansen for bringing contextual policy search to COMPI with excellent results, Sebastian Hellmann for recording a dataset of human grasps, and Bernd Poppinga for recording throwing movements.

There are many more current and former colleagues with whom I collaborated, who inspired me, and who I do not explicitly mention here. In all the research projects that I have participated in as part of the Robotics Research Group of the University of Bremen or the Robotics Innovation Center of the DFKI I learned a lot from you. I would like to thank you for that and I tried my best to also share my knowledge and experience with you.

This thesis would not have been possible without a huge open source community in the field of machine learning. I would like to mention particularly the project scikit-learn, which provides a large amount of standard machine learning algorithms in Python, but the whole scientific ecosystem in Python has much more to offer. With several projects that resulted from this thesis I try to give something back to the community.

Zusammenfassung

Verhaltenslernen ist eine vielversprechende Alternative zu Planung und Regelung für Verhaltensgenerierung in der Robotik. Das Feld wird zunehmend populär in Anwendungen, in denen Modellierung von Umgebung und Roboter umständlich, schwierig oder vielleicht sogar unmöglich ist.

Das Lernen von Verhalten für echte Roboter, die über Aufgabenparameter mit so wenig wie möglich Umgebungsinteraktionen generalisieren, ist eine Herausforderung, mit der sich diese Dissertation auseinandersetzt. Welche Probleme wir derzeit durch Verhaltenslernen lösen können und welche Algorithmen wir in der Robotik brauchen, ist zurzeit nicht offensichtlich, da es viele verwandte Felder gibt: Imitationslernen, Bestärkendes Lernen, selbstüberwachtes Lernen und gradientenfreie Optimierung.

Nach einer ausführlichen Literaturübersicht entscheiden wir uns dazu, Methoden aus den Bereichen Imitationslernen und Strategiesuche zu verwenden, um die Herausforderung anzugehen. Wir nutzen Imitationslernen mit Bewegungsprimitiven und verwenden menschliche Demonstrationen, die durch ein Bewegungsaufnahmesystem erfasst werden, um initiale Verhalten zu erzeugen, die wir dann durch kontextuelle Strategiesuche generalisieren.

Imitation von menschlichen Bewegungen führt zum Korrespondenzproblem: die kinematischen und dynamischen Fähigkeiten von Menschen und Robotern sind oft fundamental verschieden und deshalb müssen wir dies kompensieren. Diese Dissertation schlägt eine Prozedur für eine automatische Übertragung auf Roboter durch Optimierung und Strategiesuche vor und evaluiert diese mit verschiedenen robotischen Systemen.

Algorithmen für kontextuelle Strategiesuche sind häufig nicht dateneffizient genug, um direkt auf echten Robotern zu lernen. Diese Dissertation versucht das Problem mit aktiver Kontextauswahl, aktiver Auswahl von Trainingsdaten, Stellvertretermodellen und Dimensionsreduktion zu lösen. Der Fortschritt wird mithilfe von einigen simulierten und realen Lernaufgaben für Roboter illustriert. Starke Verbindungen zwischen Strategiesuche und gradientenfreier Optimierung werden in diesem Teil der Arbeit offengelegt und genutzt. Diese Dissertation demonstriert, dass das Lernen von Manipulationsverhalten in wenigen hundert Episoden direkt auf einem realen Roboter möglich ist.

Des Weiteren werden diese neuen Methoden für Imitationslernen und kontextuelle Strategiesuche in einem kohärenten Rahmenwerk integriert, das zum fast automatischen Lernen neuer Verhalten aus aufgezeichneten menschlichen Bewegungen genutzt werden kann. Entsprechende Implementierungen, die während dieser Dissertation entwickelt wurden, sind öffentlich verfügbar.

Abstract

Behavior learning is a promising alternative to planning and control for behavior generation in robotics. The field is becoming more and more popular in applications where modeling the environment and the robot is cumbersome, difficult, or maybe even impossible.

Learning behaviors for real robots that generalize over task parameters with as few interactions with the environment as possible is a challenge that this dissertation tackles. Which problems we can currently solve with behavior learning algorithms and which algorithms we need in the domain of robotics is not apparent at the moment as there are many related fields: imitation learning, reinforcement learning, self-supervised learning, and black-box optimization.

After an extensive literature review, we decide to use methods from imitation learning and policy search to address the challenge. Specifically, we use human demonstrations recorded by motion capture systems and imitation learning with movement primitives to obtain initial behaviors that we later on generalize through contextual policy search.

Imitation from motion capture data leads to the correspondence problem: the kinematic and dynamic capabilities of humans and robots are often fundamentally different and, hence, we have to compensate for that. This thesis proposes a procedure for automatic embodiment mapping through optimization and policy search and evaluates it with several robotic systems.

Contextual policy search algorithms are often not sample efficient enough to learn directly on real robots. This thesis tries to solve the issue with active context selection, active training set selection, surrogate models, and manifold learning. The progress is illustrated with several simulated and real robot learning tasks. Strong connections between policy search and black-box optimization are revealed and exploited in this part of the thesis. This thesis demonstrates that learning manipulation behaviors is possible within a few hundred episodes directly on a real robot.

Furthermore, these new approaches to imitation learning and contextual policy search are integrated in a coherent framework that can be used to learn new behaviors from human motion capture data almost automatically. Corresponding implementations that were developed during this thesis are available in an open source software.

Prior Publication

Although this dissertation is a monograph, large parts of it have been published before and some parts are submitted to conferences or journals and might be published soon. Marginal notes at the beginning of a chapter or section will refer to publications that the text is based on. At the end of Chapters 1–6 the corresponding publications will be listed.

At the same place I will describe my contributions and the contributions of my co-authors. I published some works without co-authors but most are published with at least one co-author who contributed considerably, and in a few I am just a co-author. There are several publications in which we integrated multiple components in a joint effort. Hence, the first two or three authors contributed equally and are main authors of the paper. In some cases it is necessary to also present the work of my colleagues in this thesis to show the full system and evaluation or as a prerequisite for following sections. I only summarize their work or I highlight these sections or figures as their work.

Throughout this thesis I will use the first person plural meaning *you and me together*, including the reader. I will only break this style when I explicitly distinguish my contribution from my colleagues work. In some cases, however, *we* can also include colleagues with whom I published the paper. In this case it is clearly stated at the end of the chapter who contributed in which form to this *we*.

Part I.

Foundations and Background

Chapter 1.

Introduction to Robot Behavior Learning

> The ultimate goal of AI and Robotics is to realize autonomous agents that organize their own internal structure in order to behave adequately with respect to their goals and the world. **That is, they learn.**
>
> *(Asada et al. [Asa+96])*

Parts of this chapter were published originally in [Fab+20] and have been revised.

As robots are deployed in increasingly complex environments and have to fulfill a range of different tasks, hard-coding the full set of behaviors before deployment becomes more difficult. An alternative approach is to allow robots to learn behaviors.

Machine learning and particularly deep learning [Sch14; LBH15] made groundbreaking success possible in many domains, such as computer vision [KSH12], speech recognition [Hin+12], playing video games [Mni+15], and playing Go [Sil+16]. It is unquestionable that learning from data, that is, learning from experience and observations, is the key to adaptive and intelligent agents—virtual or physical.

Yet, people are often susceptible to the fallacy that the state of the art in robotic control today heavily relies on machine learning. This is often not the case. An example for this is given by Irpan [Irp18]. At the time of writing this thesis, the humanoid robot Atlas from Boston Dynamics is the most impressive work in robot control. It is able to walk and run on irregular terrain, jump precisely with one or two legs, and even do a back flip [Bos18]. Irpan [Irp18] reports that people often assume that Atlas uses reinforcement learning. Publications from Boston Dynamics are sparse, but they do not include machine learning algorithms for control [Rai+08; Nel+12]. Instead, Kuindersma et al. [Kui+16] present state estimation and optimization methods for locomotion behavior of the robot Atlas. Robotic applications have demanding requirements on processing power, real-time computation, sample efficiency, and safety. These often make the application of state-of-the-art machine learning to generate robotic behavior infeasible.

The goal of this thesis is to make behavior learning a common tool for roboticists, as common as planning and control. We need intuitive and reliable tools to achieve this and learning should happen directly on the real system without complicated simulations and with few interactions between the robot and its environment.

1.1. Behavior

This thesis discusses behavior learning for real robots—physical agents that move in the real world. In this section, we clarify what behavior is and in which forms it can occur. In the following chapters we will analyze the state of the art of behavior learning for robots, derive a set of interesting open problems and questions, and tackle these.

1.1.1. Definition

We adapt the definition of the term *behavior* from Levitis et al. [LLF09], who derive this definition from a survey among behavioral biologists: "behaviour is the internally coordinated responses (actions or inactions) of whole living organisms (individuals or groups) to internal and/or external stimuli [...]" [LLF09]. Note that we excluded the end, as it only applies to biological systems. For our purposes, we extend this definition to artificial systems that are called robots.

Levitis et al. [LLF09] point out: "Information processing may be a necessary substrate for behaviour, but we do not consider it a behaviour by itself." This is important because it excludes perception, state estimation, and building world models from the definition of behavior, although they may be part of a behavior.

There are other terms related to behavior and behavior learning that we will use. Shadmehr and Wise [SW05, page 46] write about reaching behavior:

> Once the CNS [central nervous system] selects the targets (or goals) of reach [...] it must eventually compute a motor plan and generate the coordinated forces needed to achieve the goal, even if this computation evolves during the movement. The ability to achieve such goals typically requires a motor skill.

Hence, we can distinguish the general concept of a motor skill and an explicit and specific motor plan. The term *skill* is widely used and we define it as a learned ability of an organism or artificial system. It is not the behavior but a behavioral template that can be adapted to a behavior for certain situations that are similar to those in which it was learned. A set of skills constitutes a *skill library* or *motor repertoire*. A *motor plan* is a sequence of actions (control commands on a lower level) to be taken in order to achieve a given goal. Hence, it is a skill adapted to a specific situation.[1]

Another term that is often used in the context of robot skill learning is *movement primitive*. Movement primitives are "fundamental units of motor behavior" [GMB93], more precisely, "indivisible elements of motor behavior that generate a regulated and stable mechanical response" [GMB93]. Thus, a movement primitive can represent a learned skill.

[1] These definitions of skill and motor plan originated from discussions with Elsa Andrea Kirchner, Lisa Gutzeit, José de Gea Fernández, Alexander Dettmann, Sebastian Stock, Dennis Mronga, Nils Niemann, Sebastian Bartsch, Marc Otto, and Christoph Petzoldt whom I would like to thank for their contribution.

1.1.2. Classification

Now that we have defined behavior and related terms, we will introduce categories to distinguish behaviors and behavior learning problems. Note that some behaviors are not clearly categorizable and some categories do not apply to all behaviors.

Perception and action: Behaviors often involve perception and action (see Figure 1.1). Some behaviors, however, can also be executed open-loop, that is, they do not incorporate any sensory feedback after they have been started. Conversely, pure perception does not match our definition of behavior. Often a coupling between perception and action is required. Sometimes both components are learned, sometimes only the action is learned, and sometimes there is a stronger focus on learning the perception part of the behavior.

Deliberative vs. reactive behaviors: Arkin [Ark98] distinguishes between deliberative and reactive robot control, which also applies to learned robotic behavior. Deliberative control often relies on a symbolic world model. Perception is not directly coupled to action but it is used to populate and update the world model. Actions are derived from the world model. Deliberative control is often responding slowly with a variable latency and can be regarded as high-level intelligence. We define deliberative behaviors as behaviors that only have an indirect coupling between sensors and actuators through a world model. Reactive control does not rely on a world model because it couples perception and action directly. It responds in real-time, relies on simple computation, and is a form of low-level intelligence. Reactive control architectures often combine multiple reactive behaviors. An interesting property of these architectures is that often unforeseen high-level behavior emerges from the interplay between robot and environment. Reflexive behavior is reactive with tight sensor-actuator coupling. Both deliberative and reactive behaviors are closed-loop behaviors.

Discrete vs. rhythmic behavior: Most behaviors cause movements. Schaal et al. [Sch+04] distinguish between two forms of movements: discrete and rhythmic movements. Discrete movements are point-to-point movements with a defined start and end point. Rhythmic movements are periodic without a start or end point or could be regarded as a sequence of similar discrete movements. This distinction has often been used for robotic behaviors. Some behaviors might be rhythmic at one hierarchy level and discrete at another. Schaal et al. [Sch+04] show that discrete movements often involve higher cortical planning areas in humans and propose separate neurophysiological and theoretical treatment.

Dynamic vs. static behavior: Momentum is important in dynamic behaviors because it will either be transferred to the environment or it is required because the robot or the environment is not stable enough to maintain its state without momentum. Static behaviors can be interrupted at any time and then continued without affecting the outcome of the behavior. In practice, some behaviors also lie in between, because momentum is

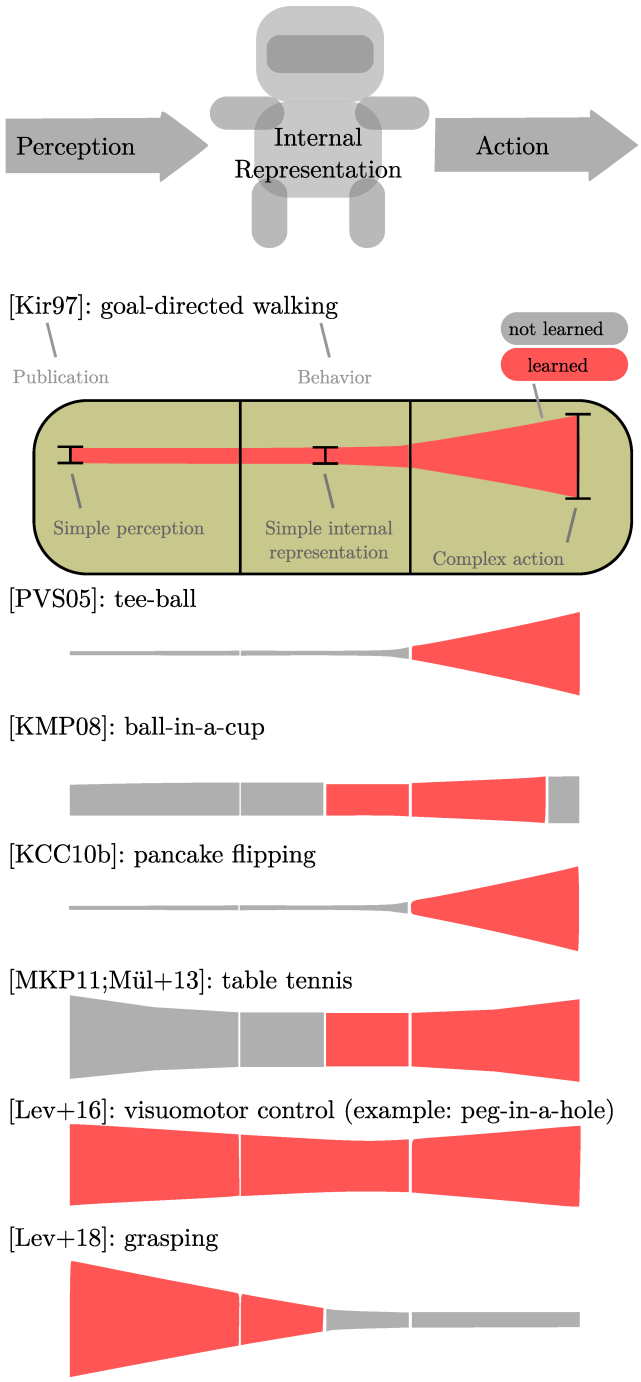


Figure 1.1.: Perception and action. The red background indicates which parts of the behavior are learned. Sometimes both, perception and action, are learned and sometimes only some aspects are learned. The height of each bar indicates complexity of the corresponding part.

not important but interrupting the behavior might alter the result insignificantly. Some problems would be solved by a human with dynamic behaviors, but when the behavior is executed slow enough, it loses its dynamic properties. This often happens when robots solve dynamic problems, hence, we call them quasi-static. This categorization is inspired by research in walking robots: a static walk can be stopped at any time and the robot will stay indefinitely at the same position [BF97]. A categorization into dynamic and static motion is also made in rock climbing [Wik18], and Mason and Lynch [ML93] provide complementary definitions for manipulation: static manipulation is defined as an operation "that can be analyzed using only kinematics and static forces", quasi-static manipulation can be analyzed "using only kinematics, static forces, and quasi-static forces (such as frictional forces at sliding contacts)", and dynamic manipulation can be analyzed "using kinematics, static and quasi-static forces, and forces of acceleration".

Active vs. passive: Some behaviors are executed with the intention to actively change the state of the robot or the world. Others are only passive, that is, they have the goal of maintaining a state and change of the environment is a side effect. An example is homeostasis, a state of steady internal conditions, which is a fundamental concept of the robotic behavior architecture of Rauch et al. [Rau+12].

Locomotion vs. manipulation: Many implemented behaviors of existing robotic systems can be categorized as locomotion or manipulation. Locomotion includes all behaviors that move the robot and, thus, change the state of the robot in the world, while change of the environment is a side effect. Manipulation is mechanical work that modifies the arrangement of objects in the world. Therefore, manipulation behaviors change the state of the environment and changing the state of the robot is a side effect.

Hierarchy of behaviors: Behaviors can have different timescales and levels of abstraction. For example, keeping a household clean is more abstract and time-consuming than picking up a particular cup. Furthermore, behaviors can consist of other behaviors. For example, a resource management behavior can achieve the goal of maintaining a storage filled by keeping track of the stored amount (stocktaking) and collecting resources (foraging) when necessary. As goals become more concrete and faster to achieve, their priority generally increases. For example, keeping balance or avoiding an obstacle are often obligatory, which leads to compromises in the achievement of higher level goals. Sub-behaviors may be executed in parallel or in a sequence and generally, the type of their combination (output weighting, suppression, sequence) is learnable.

Hierarchical behavior organization dates back at least to the field of behavior based robotics [Ark98], manifested, for example, in the subsumption architecture of Brooks [Bro86]. Organizing behaviors hierarchically has been demonstrated to be of practical relevance to organize hand-coded behaviors for the complex domain of robot soccer. The behavior specification languages XABSL [LRJ06] and CABSL [Röf18] are common among robot soccer teams. A hierarchical behavior structure is also useful to divide the learning procedure, as demonstrated by Kirchner [Kir97].

System requirements: Behaviors have different requirements on the hardware design of the robot. Many locomotion behaviors require legs, although navigation and exploration behaviors often only require wheels to move. Manipulation behaviors require grippers, hands, and / or arms. Some behaviors rely on particular sensors such as cameras, force-torque sensors, or distance sensors.

Noise and uncertainty: Generating behaviors is considerably more difficult if there is noise in state transitions or state perception. Sometimes the state is not fully observable and, hence, there is uncertainty in perception. Sometimes the state transition is not fully determined by the actions that the robot can execute because the environment itself is stochastic.

1.2. Behavior Learning

1.2.1. Behavior Generation for Real, Autonomous Robots

What is a good strategy to generate behaviors for real, autonomous robots? There are several options. We can hard-code solutions to common problems (similar to hard-wired reflexes in biological agents), which requires a good implicit model of the problem domain. We can also build a model during runtime and use offline or online solvers such as planners to come up with a solution. But what if the model is not *good enough*? Maybe the model is not accurate, maybe we cannot build a model because we are missing some required information, or maybe it is too difficult to model some aspects of the interaction between robot and environment. Learning behaviors from experience is a way to avoid this problem; however, learning behaviors for robots that move in the real world is difficult.

1.2.2. What Makes the Domain Difficult?

There are numerous reasons why machine learning is more focused on perception or is done in simulation. Robotic behaviors cannot be executed indefinitely often as robots suffer from wear and tear, require maintenance (for example, battery changes or hardware repairs [KS04]), and hardware is expensive [KBP13]. Furthermore, human supervision is often required, particularly when there is physical contact between robot and environment so that imperfect behavior might break either the robot or the environment [CP07; ET18]. Hence, training data are often sparse and learning must be effective with small datasets [KS04]. Further challenges are partial observability, uncertainty, and noise [KS04; KBP13], as well as the curse of dimensionality, since humanoid robots can have 40 or more state space dimensions [MD01]. Even modeling and simulation are difficult; in particular dynamics of many robots and their environments are complex. Behaviors are also often hard to reproduce [KBP13], since robots can even change their properties over time because of wear or changing temperatures [KBP13]. In contrast to simulation, the only option to speed up learning in the real world is to add more robots, which require more maintenance work [KS04].

Learning behaviors for robots in the real world is difficult for all those reasons. Some of them can be mitigated in laboratory conditions. Nonetheless, this domain is among the hardest for today's machine learning algorithms.

1.2.3. Complexity of Systems Is Increasing

The domain of robotics is not only difficult but it is becoming even more so with the increasing complexity of robotic systems and posed problems. A complex six-legged walking robot had 12 degrees of freedom (DOF) [MB90] at the beginning of the 90s. In 2016, a quadrupedal robot with two arms for manipulation had to handle 61 DOF [Bar+16]. Controlling such a complex robot is challenging. While the majority of the works in the field of manipulation only have to handle six or seven DOF, complex robots today control 17 [Kor+11b] or 24 DOF [Bar+16] to generate a walking behavior or 24 DOF for in-hand manipulation [Raj+17a; Ope+20]. For comparison, a well-studied biological system is the human body. It has an estimated total number of 244 DOF and a conservatively estimated number of 630 skeletal muscles [ZP12]. It is, hence, a much more complex system to control than any robot used in works that are cited in this thesis.

Not only the actuation capabilities improved but also the complexity of sensors increased considerably in almost three decades of behavior learning research on real robots. In early applications only simple sensors have been used (for example, four light sensors [Kir97]). Alternatively, the perception problem has been decoupled from the action problem to solve it with computer vision and state estimation [Mül+13; Par+15]. In more recent works, raw camera images have been used directly by the learned behavior [LR13; Lev+16; Lev+18] and RGB-D cameras have been used [LLS15]. RGB-D cameras are probably the most complex sensors that are used in learned behaviors today. Robotics research in general is already more advanced though and we will see other complex sensors in addition to conventional cameras. Current robotic systems can have advanced tactile sensor arrays based on fiber-optic sensing principles [Bar+16].

1.2.4. When Should Behaviors Be Learned?

One of the main questions that we would like to answer here is which behaviors robots should learn given the availability of alternative approaches and difficulties applying machine learning to real robotic systems. This is often intuitively clear to machine learning and robotics researchers, but the intuition is often not underpinned by evidence. The field is diverse so that it is easy to miss something.

We see several strengths of learned behaviors that have been mentioned quite often:

- Handling uncertainty and noise.
- Dealing with inaccurate or non-existing models.
- Learning can be better than hand-crafted solutions.
- They are easier to implement.
- They are often simple, sufficient (or even optimal) heuristics.

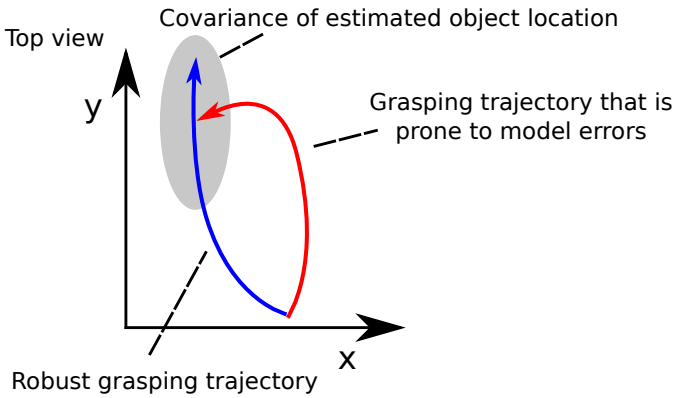


Figure 1.2.: Sketch of a robust grasping trajectory from top view. The ellipse indicates the uncertainty of the object's estimated position. A grasp that moves along the axis of highest variance of the estimate (blue trajectory) will succeed with a higher probability than a grasp that moves along the axis of lowest variance (red trajectory).

We will back up these findings with sources in the following paragraphs. Machine learning is also considered to be the direction to real artificial intelligence or, as Asada et al. [Asa+96] put it: "The ultimate goal of AI and Robotics is to realize autonomous agents that organize their own internal structure in order to behave adequately with respect to their goals and the world. That is, they learn."

1.2.4.1. Handling Uncertainty and Noise

Uncertainty and noise are predominant problems in robotics. Sensors and actuators suffer from noise, which makes noise part of the world from the perspective of a robot. Information about the world is usually incomplete and knowledge is not certain. Therefore, uncertainty played a central role in robotics research since its beginning [Mas12] and probabilistic methods are popular in the robotics community (see, for example, Thrun et al. [TBF05]). The following works demonstrate that learning can handle uncertainty.

Stulp et al. [Stu+11; STS12] show that state estimation uncertainty in a pick and place problem can be compensated with an adapted motion. We illustrate how a compensatory motion can address the problem of state estimation uncertainty in Figure 1.2.

An example of incomplete information is presented by Levine et al. [Lev+18], where just a single RGB camera is used to learn grasping end to end.[2] The distance and the three-dimensional structure of objects cannot be inferred from only one camera. Objects are in the same distance to the robot, however, when they are at the same position in the image. Hence, the system implicitly learns the objects' distances.

Furthermore, Deisenroth et al. [DFR15] use a low-cost robotic manipulator and show that their method can compensate for actuator noise, Carrera et al. [Car+12] find that

[2]End to end means that a direct mapping from sensor data to actuator commands is learned.

learning offers the adaptability and robustness that is required to solve their problem of turning a valve, Kober et al. [KMP08] learn a coupling of perception and action to handle perturbations of trajectories, and Gullapalli et al. [GFB94] learn peg-in-a-hole insertion with sensor noise in position encoders and in a wrist force sensor and demonstrate that reinforcement learning can be used to generate robust insertion behavior.

Oßwald et al. [OHB10] report that execution of motion commands is noisy on their humanoid robot because of backlash in joints and foot slippage. This results in motion blur and makes pose estimation more difficult. Nevertheless, they are able to learn a high-level navigation behavior that reduces pose estimation uncertainty that arises from the noise.

Johns et al. [JLD16] consider the problem of grasp pose prediction and write:

> issuing commands to align a robot gripper with that precise pose is highly challenging in practice, due to the uncertainty in gripper pose which can arise from noisy measurements from joint encoders, deformation of kinematic links, and inaccurate calibration between the camera and the robot. [JLD16]

They develop a learning method that explicitly addresses these uncertainties.

1.2.4.2. Dealing with Inaccurate or Non-Existing Models

When there is no model of the robot or the world or existing models are too inaccurate, machine learning is often able to compensate for that. This has been shown in the context of dynamic behaviors. It is hard to model dynamics correctly but it is often not required. For example, Mülling et al. [Mül+13] use state estimation to predict ball trajectories in table tennis but neglected the spin of the ball. Similarly, Parisi et al. [Par+15] use a simplified model of the forward dynamics of a robotic arm with springs and the learned behavior was able to work with the simplified model. Furthermore, Kormushev et al. [Kor+11b] solve the problem of energy minimization in a walking behavior that is used with a robot that has springs in its legs. They report that it is nearly impossible to solve the problem analytically "due to the difficulty in modeling accurately the properties of the springs, the dynamics of the whole robot and various nonlinearities, such as stiction."

In general, soft bodies and soft-body dynamics are difficult to model, which motivates Colomé and Torras [CT18] to use machine learning for the task of folding a polo shirt. Moreover, Englert and Toussaint [ET18] write that a

> main issue is that the external degrees of freedom can only be manipulated through contacts, which are difficult to plan since a precise and detailed physical interaction model is often not available. This issue motivates the use of learning methods for manipulation skills [...]. [ET18]

1.2.4.3. Learning Can Be Better than Hand-Crafted Solutions

Kohl and Stone [KS04], Kwok and Fox [KF04], Kober et al. [KMP08], and Parisi et al. [Par+15] compared machine learning and hand-crafted approaches. These works show

that learning is able to yield better behaviors than model-based or hand-tuned solutions, although these results have to be read carefully because they are subject to publication bias. There are only a few publications in which machine learning for robotic behaviors and another method are compared with the result that machine learning performs worse. For example, Bargsten et al. [BGK16] compare machine learning with dynamic model identification to learn a model of inverse dynamics with the result that the machine learning method is worse because it does not generalize well. Although it has to be noted that the dynamic model identification is also a data-driven method with incorporated physical prior knowledge.

1.2.4.4. Learning Behaviors is Easier than Other Approaches

It is often easier to specify the problem than to specify the solution. A reward for reinforcement learning, for example, can encode the problem specification. Thus, learning approaches are often easier to implement because they are not problem-specific. Examples of problems where it is easy to define the reward are: walking as fast or straight as possible, jumping as far as possible, throwing as close to a target as possible, or grasping because we could apply random perturbations after the grasp and measure if the gripper still holds the object. While *walk as fast as possible* alone might not be a sufficient reward function, additional components of the reward function are usually intuitive and part of the problem specification; we can penalize behaviors that let the robot fall down or exert high forces on parts of the robot. Kormushev et al. [KCC10b] made an observation that supports this point. They found that the solution to the pancake flipping problem that has been discovered by learning contains an unexpected compliant catching behavior at the end of the movement, which prevents the pancake from bouncing off the pan. They conclude "such undesigned discoveries made by the RL [reinforcement learning] algorithm highlight its important role for achieving adaptable and flexible robots". Imitation learning is another method that is particularly easy to use from an end users perspective. It enables users to teach robots new behaviors without requiring expert knowledge or programming skills [ACC14]. We do not want to deny that tuning hyperparameters of a machine learning algorithm is a complex task and requires expert knowledge, but Parisi et al. [Par+15] found that tuning hyperparameters can be less time intensive than building a mathematical model for a given task. Amor et al. [Amo+14] justify the use of machine learning in the context of human-robot interaction: "programming robots for such interaction scenarios is notoriously hard, as it is difficult to foresee many possible actions and responses of the human counterpart". Matsubara et al. [Mat+05] learn a walking behavior and point out the drawback of classical, model-based approaches. These require precise modeling of the dynamics of the robot and the environment. Fidelman and Stone [FS04] write that their paper

> is concerned with enabling a robot to learn high-level goal-oriented behaviors. Coding these behaviors by hand can be time-consuming, and it often leads to brittle solutions that need to be revised whenever the environment changes or the low-level skills that comprise the behavior are refined. [FS04]

Levine et al. [Lev+18] assume that "incorporating complex sensory inputs such as vision directly into a feedback controller is exceedingly challenging" and show with their approach that learning complex emergent behavior can be done without much prior knowledge.

Considering the long-term perspectives of robotics and artificial intelligence, the following works are relevant. Cully et al. [Cul+15] tackle online adaptation to hardware defects, similar to how injuries are compensated by animals' behavior. They found:

> while animals can quickly adapt to a wide variety of injuries, current robots cannot 'think outside the box' to find a compensatory behavior when damaged: they are limited to their pre-specified self-sensing abilities, can diagnose only anticipated failure modes, and require a pre-programmed contingency plan for every type of potential damage, an impracticality for complex robots. [Cul+15]

Kirchner [Kir97] considers the problem of an autonomous robot that adapts its behavior online and assumes that robots acting in the real world will encounter similarly unforeseeable situations, which makes learning a necessity.

1.2.4.5. Simple, Sufficient Heuristics

Before we elaborate on the last point, we will draw an analogy to behaviors of biological systems. Many behavior learning algorithms do not guarantee optimality. Hence, we consider learned behaviors to be heuristics, which are often computationally efficient. Nonetheless, they are not necessarily second-best strategies. In real world situations, where an agent is embodied in a physical system with noisy sensors and actuators that result in uncertainty, heuristics often yield good behaviors. An example for heuristic behavior is the gaze heuristic that is used to catch a ball that is high up in the air: "Fix your gaze on the ball, start running, and adjust your running speed so that the angle of gaze remains constant." [GB09] The agent will be at the position where the ball comes down. Other variables can be ignored, for example, distance, velocity, and spin of the ball, air resistance, and speed and direction of the wind. Gigerenzer [Gig08] explains why heuristics are useful in the case of human behavior, but these arguments also apply to robotics. An optimal solution to a real-world problem is often computationally intractable, for example, NP-hard[3] or so ill-defined that we do not know exactly what we should optimize for. In addition, real-world problems demand for robustness of behaviors. More information and computation is not always better according to Gigerenzer [Gig08]. Reasoning often results in worse behavior because of model errors. Robustness sometimes even requires to ignore or forget information. The following papers from the robotics community support these statements. Berg et al. [Ber+10] consider the problem of cutting, which would be hard to model completely but has simple solutions. Benbrahim and Franklin [BF97] suggest: "The fact that walking is most of the time

[3] NP means *nondeterministic polynomial time*; NP-hard for our purpose means that the optimum solution usually cannot be determined practically.

done unconsciously suggests that maybe it does not require constant heavy computing in normal walking conditions." Kuindersma et al. [KGB11] learn balancing behaviors with arm motions and point out: "This general problem also has several attributes that make it interesting from a machine learning perspective: expensive evaluations, non-linearity, stochasticity, and high-dimensionality. In our experiments, a low-dimensional policy space was identified".

1.2.4.6. A Perspective from 1995

More than two decades ago, Thrun and Mitchell [TM95] already tried to answer when behaviors should be learned. They distinguish between model-based approaches (with a model of the robot and the world) and learning. In a way we can consider every approach that does not use machine learning to be model-based because it either uses an explicit model (for example, planning, reasoning, or optimal control) or an implicit model (for example, behavior definitions with finite state machines or hard-coded motions). Learned behaviors also build models but learned models directly encode real experience. Thrun and Mitchell [TM95] identify four bottlenecks of model-based methods. (1) There is a *knowledge bottleneck*: knowledge has to be provided by a human. While this is not totally accurate anymore because robots are, for example, able to build detailed maps of their environment on their own, this is still an issue because a programmer has to define how to interpret the data: what is rigid and what is soft, which objects are movable and which are fixed? (2) There is an *engineering bottleneck*: it requires a lot of time to implement and generate these explicit models. For example, realistic modeling and physics simulation of soft bodies, divisible bodies, deformable objects, fluids, or granular media are still difficult. (3) There is a *tractability bottleneck*: many realistic problems are computationally complex or even intractable which results in slow responses. For example, Kuindersma et al. [Kui+16] report times of 1.5 or 10 minutes to plan simple jumping motions. (4) There is a *precision bottleneck*: the robot must be able to execute plans accurately enough. This is still an issue and is becoming more relevant with flexible and compliant robots.

While all of the mentioned points are still valid, some of them also apply to state-of-the-art machine learning. The knowledge bottleneck is an issue if pre-structured policies or models are used, for example, dynamical movement primitives [Ijs+13]. The tractability bottleneck has a counterpart in machine learning: a lot of experience might be required. As we have seen, simple heuristics are often sufficient, which means that neither pre-structuring or restricting the policies or models necessarily results in bad performance, nor will learning require much data. The precision bottleneck is related to the simulation-reality gap [JHH95] that is a problem if behaviors are learned in simulation and transferred to real systems (Kwok and Fox [KF04] report this problem).

1.2.5. An Analogy: Shifting from Deliberative to Reactive Behaviors

Human behavior has been analyzed from many different perspectives and one that is relevant for this thesis is the following:

> Conscious thinking takes time and mental resources. Well-learned skills bypass the need for conscious oversight and control: conscious control is only required for initial learning and for dealing with unexpected situations. Continual practice automates the action cycle, minimizing the amount of conscious thinking and problem-solving required to act. Most expert, skilled behavior works this way, whether it is playing tennis or a musical instrument, or doing mathematics and science. Experts minimize the need for conscious reasoning. [Nor13, pages 100–101]

Skilled human behavior is trained and repeated often. We do not waste many computational resources and are able to execute it fast and precisely. In other words

> motor learning matters because it allows you to act while directing your attention and intellect toward other matters. Imagine that you needed to attend to all of the routine aspects of your reaching or pointing movements. Motor learning provides you with freedom from such a life. [SW05, page 2]

Exactly the same statement could be made for robotic behaviors. Learning individual skills also simplifies reasoning and planning because planning can take place purely on a high level and solve the problem of combining individual skills.

An argument in favor of learning robotic behaviors is this analogy to well-learned human behavior. As we have seen, learned behaviors are mostly reactive behaviors or heuristics. This is the precise opposite of the useful combination of mapping, state estimation, and planning which we categorize as deliberative behavior. While state estimation and planning works without previous interaction with the environment, learned behaviors can be faster and can have a higher performance if enough data are available or trials are allowed. While deliberative behavior can be a safe first solution, it can be replaced by learned and reactive behaviors. This is actually similar to what humans do.

In summary, there is an analogy between humans and robots: learned behavior can perform better while requiring less computational resources in comparison to high-level reasoning in certain problem domains.

1.2.6. When Should Behaviors Not Be Learned?

Imagine you are a robot and you are in a critical situation that you have never seen before. Dismukes et al. [DGK15] have an advice for you: "identify and analyze decision options" and "step back mentally from the moment-to-moment demands [...] to establish a high-level [...] mental model that guides actions". Oh, you learned all of your behaviors end to end and you do not know how to build a high-level mental model? Tough luck!

Not everything should be learned. Learning in robotics often aims at achieving the quality of human behavior that cannot be reached by other approaches. Humans are much better than robots at many tasks that require interpreting complex sensory data, involve noise and uncertainty, or fast and dynamic behavior. They are the best examples of a learning, physical agent that we know so far, but it might be hard to achieve better results than a human if we try to use the same design principles for robots. Humans

make errors all the time and the frequency of errors can even increase under external factors such as stress [DGK15]. While we do not think that robots are prone to stress, we think that in learned robotic behaviors often unpredictable failures might occur. A robot might encounter a situation that does not occur in the training set (distributional shift, see Amodei et al. [Amo+16]) or the agent learns continuously which means that it also forgets. Therefore, sometimes it makes sense to rely on logical reasoning and model-based approaches. Ironically, Dismukes et al. [DGK15] propose the same for humans to reduce errors under stress (it is the advice that we quoted in the previous paragraph). Humans, however, are weaker at strict logical reasoning and planning.

If a precise model of the world is available, planning and optimal control often generate new behaviors faster and do not require physical interaction with the real world before they provide a solution. For instance, collision avoidance based on distance sensors and planning or reactive behaviors can be close to perfect so that it is applicable in industrial scenarios [Gea+17]. If collision avoidance is learned, there is no guarantee for safety. Particularly, there will be no safe collision avoidance during the learning phase, in which imperfect behaviors will be explored on the real system. Tassa et al. [TET12a] show that, even if the model is not accurate, model predictive control (MPC) with a finite horizon can be used to generate intelligent and robust get-up and balancing behaviors. It has to be noted though, that optimal control and reinforcement learning are related [SBW92]. In this thesis we make the distinction between reinforcement learning that needs experience and optimal control that needs a model, although machine learning and optimal control can be combined [Lev+16; Eri+18].

Learning systems are not good at repetitive tasks and tasks that demand for high precision, for example, tasks that have to be executed in a factory. If the same car has to be produced several thousand times in precisely the same way, it is worth the effort to let a human design the process step by step. In a lot of situations it is even better to build specialized machines instead of using robots. Robots and behavior learning are only required if the system encounters changing requirements or environments.

Coordination of behaviors is a difficult task for machine learning at the moment. Whole-body control [SK06] is quite successful in this domain. It allows to prioritize tasks and solves everything online in a high frequency on the system. If, for example, an existing walking and object manipulation behavior should be combined so that the robot keeps its balance, whole-body control is the method of choice. Whole-body control is effective because it uses domain-specific knowledge: the Jacobian of the robot, which contains information about a kinematic chain. In order to exhibit similar behavior, a learned behavior would implicitly have to approximate the Jacobian. Configuring whole-body control, however, is challenging. Weighting and prioritizing subtasks such that the result "solves the task" is a difficult, manual task.

Perception for dynamic problems is challenging at the moment. It can be learned for static behaviors such as grasping [Lev+18] or visual servoing [Lev+16] but it is nearly impossible at the moment to learn a catching behavior for a ball end to end because the learned model has to solve difficult perception, tracking, and prediction problems while it must respond quickly. Birbach et al. [BFB11] impressively show how computer vision and state estimation without machine learning can be used to track ball trajectories with

an error of 1.5 cm in the predicted catch point. The perception takes about 25 ms and tracking about 10 ms per step. A ball catch rate of 80 % has been reached on a humanoid.

Learned behavior can show emergent properties. While this is sometimes good (for example, in the pancake flipping task [KCC10b]), it can also be disastrous. For example, in reinforcement learning or similar disciplines learning algorithms often exploit ill-posed problem definitions. This is called *reward hacking* [Amo+16, pages 7–11] and it is not necessarily immediately visible. This problem can be particularly challenging if the behavior should be used in a variety of different contexts and environments.

Interestingly, playing soccer is an exceptionally complex high-level behavior that robots are able to perform today without learning. It is often not even solved by methods that fall into the category of artificial intelligence. Hand-crafted behavior is the state of the art for about two decades. Röfer [Röf18] gives a reason for that: "In the domain of RoboCup, real-time requirements and limited computational resources often prevent the use of planning-based approaches". Between 2009 and 2017 three distinct teams won the RoboCup Standard Platform League (SPL), which is carried out every year: B-Human, UT Austin Villa, and rUNSWift. All of them used fixed behaviors. Few background information about the behaviors used by UT Austin Villa is available but the report accompanying their code release [Bar+13] suggests that behavior is hand-crafted. rUNSWift's behavior is hand-crafted and written in Python [Ash+15]. B-Human used XABSL [LRJ06] and uses CABSL [Röf18] to describe behaviors. Both languages are used to define hierarchical finite state machines for the robots' behavior. Only in 2018 a team using a *dynamic strategy*, Nao-Team HTWK, won the RoboCup SPL. They represented the problem of positioning players that are not close to the ball as an optimization problem and solve it [Mew14]. That, however, is only a part of the whole soccer behavior.

1.3. Limitations of Behavior Learning

1.3.1. Limited Versatility of Learned Skills

The works on bipedal walking are particularly interesting, since they allow a direct comparison of the application on real robots and the application in simulation and computer graphics. Peng et al. [Pen+17] learned bipedal walking in simulation on two levels: a low-level walking behavior and a high-level behavior that generates the walking direction. The high-level behavior incorporates information about the surrounding terrain and has been used to follow trails, dribble a soccer ball towards a target, and navigate through static and dynamic obstacles. The low-level behavior only knows about the internal state of the walker and the desired goal of the high-level behavior and was trained to be robust against disturbances and terrain variations. Furthermore, Peng et al. [Pen+18] demonstrate how imitation and reinforcement learning can be used to generate realistic acrobatic movements: performing a cartwheel, backflip, frontflip, roll, vault, dancing, kicking, punching, and standing up. Those skills are then combined to a complex sequence of behaviors. In comparison, learned biped walking behaviors on real robots are only tested in controlled environments in the lab [BF97; Mat+05; GPW06; Kor+11b; MB15].

Walking is just one example of how skills that have been learned on real robots are often not versatile. Another example is grasping: the currently most impressive work, published by Levine et al. [Lev+18], is applicable to a large variety of objects but only if the camera is in a certain angle to the objects and only vertical pinch grasps have been considered. Other behaviors, for example, tee-ball [PVS05; PS08b], pancake flipping [KCC10b], plugging in a power plug [Che+17a], flipping a light switch [Buc+11], do not even include the position of the manipulated object in their control loop. Many of the learned behaviors are hence still only applicable under controlled lab conditions.

1.3.2. Limited Variety of Considered Problems

In natural learning agents (also known as animals), there is evidence that the same learning mechanisms can be evolved and used to solve a variety of tasks:

> A major role of the early vertebrate CNS [central nervous system] involved the guidance of swimming based on receptors that accumulated information from a relatively long distance, mainly those for vision and olfaction. The original vertebrate motor system later adapted into the one that controls your reaching and pointing movements. [SW05, page 9]

In contrast, often the same simple problems with only minor variations are tackled again and again in behavior learning for robots with a large variety of different learning algorithms. Learning efforts often focus on grasping, walking, and batting. These problems are not solved yet ("Robot grasping is far from a solved problem." [JLD16]) and solving the exact same problem again is good for benchmarking. Yet, the variety of problems solved by learning is low. We should also try to solve a larger variety of problems to discover and tackle new challenges in behavior learning and to improve our set of tools. Examples are given in the outlook.

Most of the considered problems are only low-level motor skills. While this seems to be too simple at first, there is also a justification for it. Shadmehr and Wise [SW05, page 1] assume that motor learning, that is, learning of low-level behavior, uses the same basic mechanisms as higher forms of intelligence, for example, language and abstract reasoning. Nevertheless, the goal should be to demonstrate that learning is possible and useful at all levels of behavior and to use its full potential.

Given the current developments in behavior learning and computer vision, we expect that the next big steps will be made by deep learning (see Chapter 2) and by solving more complex perception problems. We emphasize, however, that for complex behaviors not only complex perception but also complex control is required. We should strive towards pushing the limits of robots' kinematic complexity as well as motion complexity.

1.3.3. Reasons for Current Limitations

What hinders robots from learning the same skills as humans with a similar performance these days? The main reasons are algorithmic, computational, and hardware problems.

Not many fields of artificial intelligence are as advanced as computer vision based on deep learning. In specific benchmarks computer vision is better than humans although

it is not as robust as a human, which has been demonstrated by adversarial examples [Sze+14] and overgeneralization [Jac+19]. In addition, semantic segmentation, tracking objects in videos, object detection with a large amount of classes are examples of active research topics in which humans are much better. Computer vision is one example of a domain which behavior learning builds upon. When we learn grasping [Lev+18] or visual servoing [Lev+16] end to end, we make use of the results from computer vision research. While we do not reach human-level performance in these areas, we can hardly surpass it in real-world behavior learning problems. Also reinforcement learning algorithms are not yet at the point where they are sample-efficient enough to learn complex behaviors from a reasonable amount of data. An impressive recent example is from OpenAI et al. [Ope+20], who learned in-hand manipulation to rotate a cube into any desired orientation. 100 years of experience were collected during training. Still the robustness of the skill is not comparable to an average human: on average 26.4 consecutive rotations succeed when 50 is the maximum length of an experiment. No human spent 100 years on learning exclusively in-hand manipulation and most of us reach a much better level of performance, although part of this success can be attributed to evolution.

Many state-of-the-art algorithms in machine learning have also high demands on processing power during prediction phase [Sil+16; Lev+18; Ope+20] which makes them slow in reaction time, maybe not even suitable for autonomous systems that have to budget with energy, and training on a robotic system might be infeasible.

Probably the main reason why not many researchers learn complex skills for robots in reality is that robots wear and break easily, which makes application of algorithms with low sample efficiency and unsafe exploration infeasible. In contrast, humans collect much more data, fail and fall all the time, and gain lots of negative experiences. There is probably not a single professional soccer match that has been played over the full length, in which no player is falling down unexpectedly, and yet most players are not seriously injured. Humans are colliding all the time with objects, when they move things around, for example, while eating at an overly full dinner table. The difference is that humans are flexible, soft, and lightweight. As already mentioned, they have about 244 DOF and 630 skeletal muscles [ZP12] and most of their body is soft, while a complex robot today has 61 DOF and consists mostly of stiff and rigid parts [Bar+16] that are at the same time fragile. Thus, we either develop more sample-efficient algorithms or we can build more flexible and robust robots. For example, Haddadin et al. [Had+09] propose to use elastic joints in the domain of robot soccer, which make robots more robust, collaboration or competition with humans safer, and they would enable higher maximum joint speeds. Controlling elastic joints is more complex though. In addition, humans have many sensors (tactile, acoustic, vestibular) that are used to recognize unexpected events and they can react accordingly: they learned to fall or to stop moving the arm before they pull down the bottle from the dining table.

While we do not have all these safety mechanisms and robust hardware yet, a good approach to behavior learning for robots is to use as few interactions with the environment as possible and to integrate prior knowledge, but we should do so by relying on intuitive human knowledge and universal principles such as knowledge about robot kinematics and physics rather than expert knowledge in machine learning.

1.4. Objectives

We now have an idea of what behaviors are, what behavior learning does, what it can do for robots, and what it cannot do. We also know about the current limitations of behavior learning. This leads us to the definition of the goals for this thesis that tackle the most important problems of behavior learning to make it a common tool in robotics:

1. Reduce required expert knowledge to learn new behaviors. Behavior learning algorithms should be easy to apply to a variety of systems and tasks without expert knowledge about the underlying algorithms.
2. Reduce the number of episodes required to learn non-trivial behaviors directly on real robots to a few hundred (100–300). Behavior learning should be sample-efficient enough to directly learn on a real robot.
3. Generalize behaviors over relevant parameters of the task. Behavior learning should not just learn a solution to one specific situation.
4. Evaluate on a variety of different robotic systems and tasks.

Referring to the behavior classification introduced in Section 1.1.2, the scope of this thesis is limited to low-level, active, discrete manipulation behaviors with an emphasis on action generation. Hence, we will mostly work with robotic arms and learn static and dynamic behaviors.

We develop algorithms with the goal to learn directly on real robots and avoid simulations; however, we will use simulations, as this is the best option to perform many repetitions of experiments to compare algorithms and evaluate their reliability. Sometimes we will even be able to transfer learned behaviors directly from simulation to reality, but usually there is a simulation-reality gap [JHH95]. Note that building a simulation would often require manual work unless a robot is able to build its own simulation from sensor measurements.

1.5. Contributions

This thesis makes the following contributions to behavior learning for robots:

- In Chapter 1 we discuss when behavior learning should be used.
- In Chapter 2 we review behavior learning for robots extensively.
- In Chapter 3 we discuss a novel procedure for an automatic mapping from human motion to robots. This procedure includes task-agnostic global and local trajectory optimization and task-specific refinement through policy search. We evaluate the task-agnostic part on four simulated and one real robotic systems based on 697 movements recorded from seven subjects. Furthermore, we compare target-system specific refinement in joint space and Cartesian space in several benchmarks of varying difficulty.

- In Chapter 4 we improve the sample efficiency of contextual policy search with various approaches: active learning, training set selection, surrogate models, and manifold learning. These improvements were evaluated on several benchmark problems, simulated robots, and real robots. We mostly consider throwing as a benchmark but also tackle the problem of grasping. The best compromise between sample efficiency and avoiding a hand-crafted solution is an extension of Bayesian optimization in combination with manifold learning from human demonstrations.
- In Section 4.2 we develop the positive upper boundary support vector estimation (PUBSVE), which is a new model to estimate upper boundaries of data. We use it to implement our approach to active training set selection.
- In Section 4.5 we develop an autoencoder to generate smooth trajectories.
- In Chapter 5 we discuss a framework for imitation and reinforcement learning, of which parts were designed and implemented in the scope of this thesis. We evaluate it with 240 throwing motions from ten subjects. Furthermore, we present applications to grasping and pulling a lever.
- In Chapter 6 we discuss the underlying software, which is designed to easily apply our approaches to new problems.

The algorithms that we discuss in this thesis work particularly well if reward is sparse and typically occurs at the end of an episode or if temporal credit assignment is difficult. Good examples of problems that can be solved well by our methods are grasping, throwing, and pulling a lever. Counter-examples are peg-in-a-hole, obstacle avoidance, and problems with multiple via points.

1.6. Summary

Many people have a misconception of the prevalence of machine learning for robot control. We call machine learning for robot control *behavior learning*. In this chapter, we define what a behavior is and how behaviors can be distinguished. Important questions that we discuss are why behavior learning for real robots is difficult, when it can be useful, and when it is not. Learned behaviors often have limited versatility and the considered problems are limited. One of those limitations is the recent focus on perception. We also see that there are several reasons for these limitations: algorithmic, computational, and hardware problems. The goals of this thesis mainly address algorithmic problems.

Related Publications

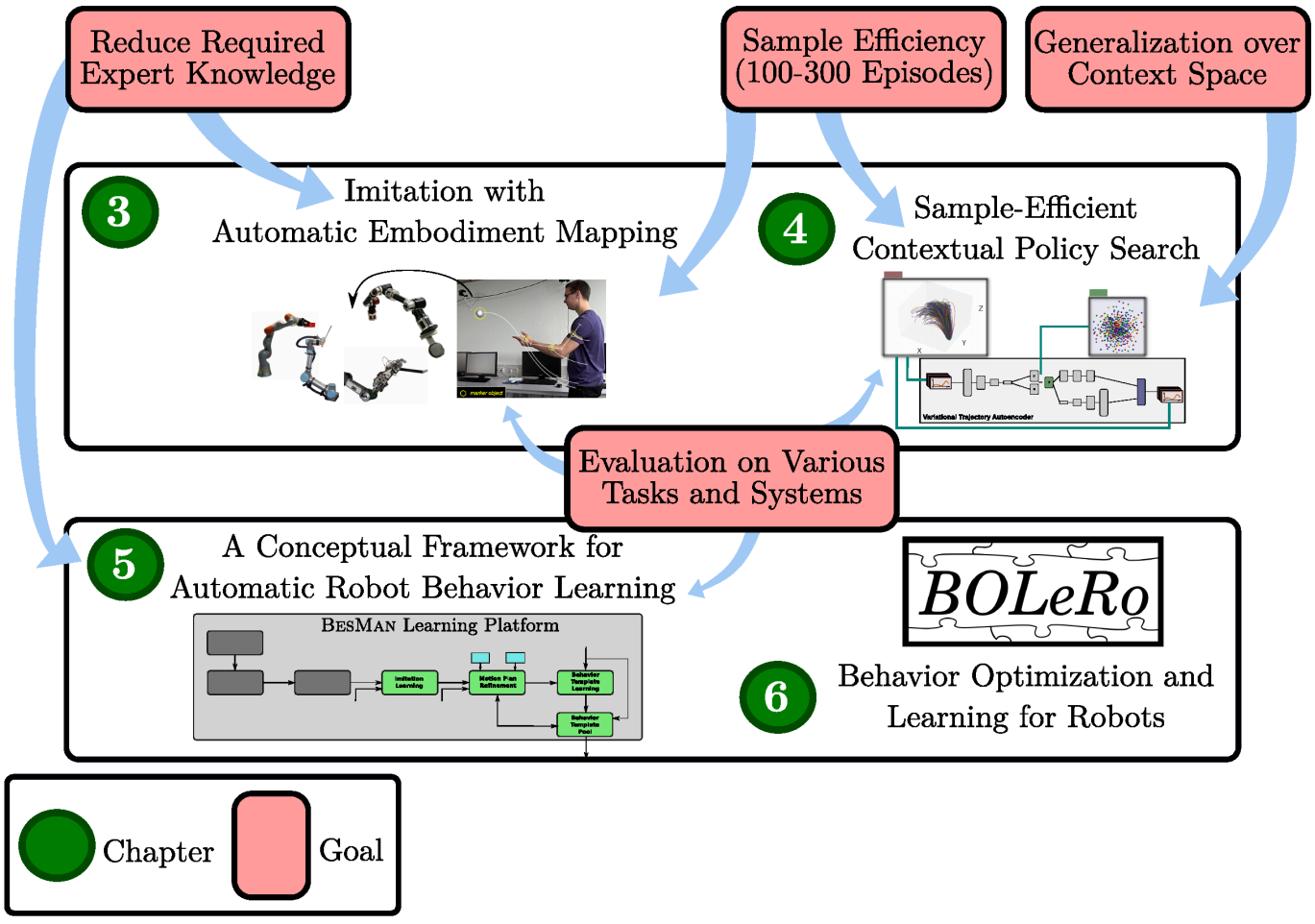
[Fab+20] Alexander Fabisch, Christoph Petzoldt, Marc Otto, and Frank Kirchner. "A Survey of Behavior Learning Applications in Robotics—State of the Art and Perspectives". In: *International Journal of Robotics Research* (2020). Submitted.

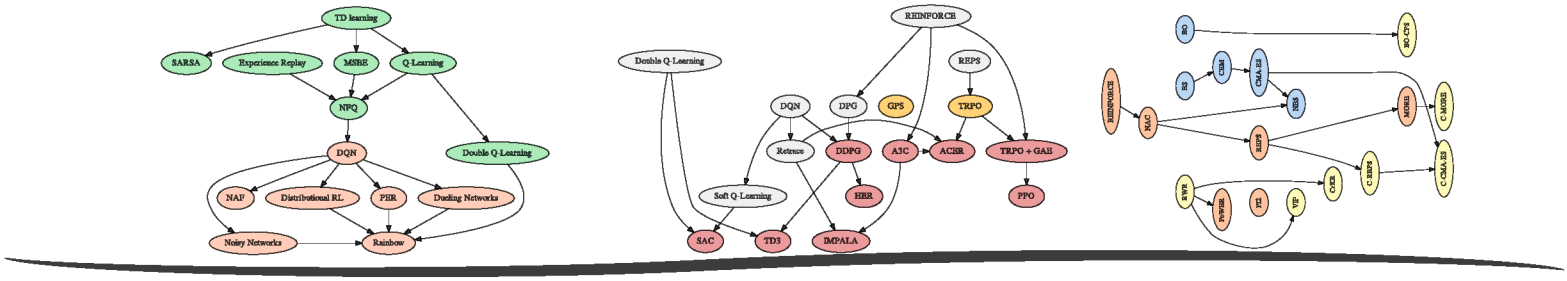
The discussion of behavior learning in this chapter is based on Fabisch et al. [Fab+20]. The complete publication is a joint work with the co-authors, but I contributed the analysis included in this thesis (Chapters 1, 2, and 8). Exceptions are marked accordingly.

1.7. Thesis Structure

In Chapter 2 *State of the Art* we will discuss behavior learning problems and algorithms to select a category of algorithms that will be used throughout this thesis. The main contributions of this thesis to the field are presented in Chapters 3 *Imitation with Automatic Embodiment Mapping* and 4 *Sample-Efficient Contextual Policy Search*. These will be embedded in a framework in Chapter 5 *A Conceptual Framework for Automatic Robot Behavior Learning*. The corresponding software BOLeRo will be presented in Chapter 6 *BOLeRo: Behavior Optimization and Learning for Robots*. The appendices contain complementary information. Appendix C *Overview of Mathematical Notation* and the glossary at the end of the document might be useful in the beginning.

Most chapters of this thesis have been published before. The corresponding publications are specified in marginal notes and at the end of each chapter the contributions of this thesis are discussed in detail.





Chapter 2.

State of the Art

This chapter summarizes the state of the art in behavior learning. We get an overview of applications and problems first and then take a closer look at algorithms and solutions. A detailed description of algorithms that are relevant for this thesis concludes the chapter.

2.1. Robotic Behavior Learning Problems

This section was published originally in [Fab+20] and has been revised.

Results in the area of machine learning are impressive but, as we have seen in Chapter 1, they can create a false impression. This leads us to the question: which behaviors have actually been learned?

We answer this question by giving examples and we only summarize the results here to limit the length of this chapter. A detailed discussion of each publication, analysis of problem characteristics, and categorization of the corresponding behavior can be found in Appendix A *Survey of Behavior Learning Problems*. We can draw a lot of interesting conclusions from this review regarding the questions when we should and when we should not learn behaviors. These have been summarized in Section 1.2. In this section we will give practical examples of behaviors that have been learned to get familiar with the topic before we dive into the discussion of algorithms.

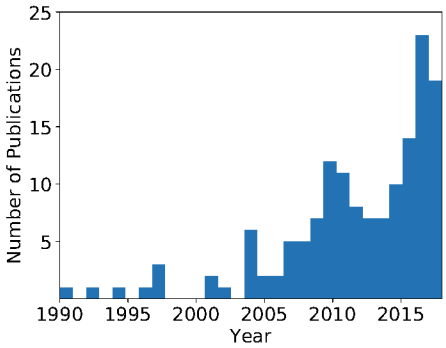


Figure 2.1.: Number of considered publications by years.

A histogram of analyzed papers by publication dates is shown in Figure 2.1. Although it is unlikely that every relevant work is included, it shows that the number of applications of behavior learning to robotic systems has been growing fast in the last 10 years. Figure 2.2 shows how individual behaviors that have been learned can be grouped by their domain of application. Some behaviors can of course be applied in several domains. These are elementary behaviors. Examples are walking and grasping. Table A.1 summarizes the behavior learning problems, corresponding publications, and their categorization.

We mostly consider works that meet the following criteria. (1) They were implemented on real robots because it is much more demanding to integrate and learn behaviors in

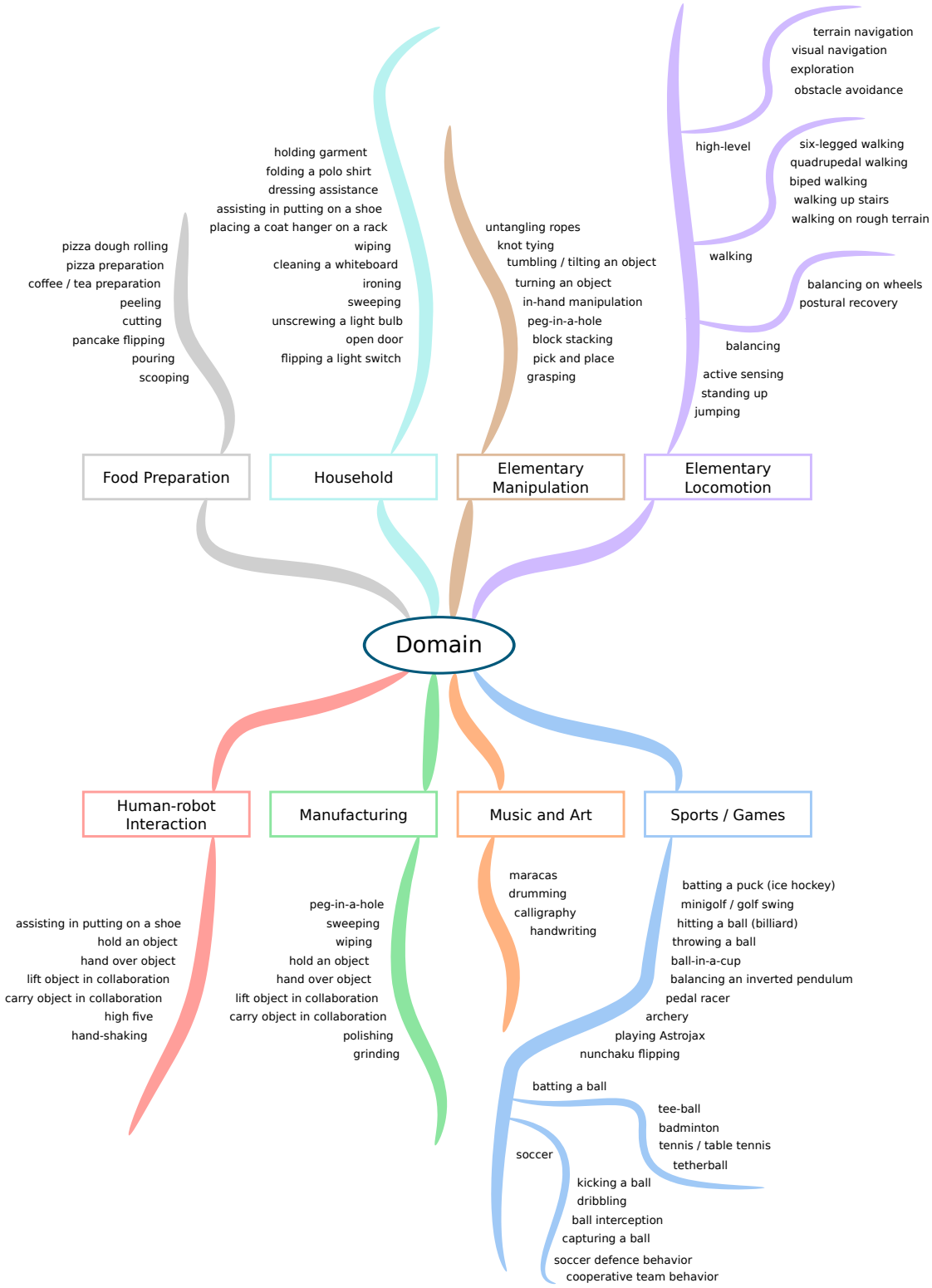


Figure 2.2.: Mind map of behavior learning applications ordered by domain. Some behaviors are assigned to multiple domains and most of the elementary behaviors could also belong to multiple domains.

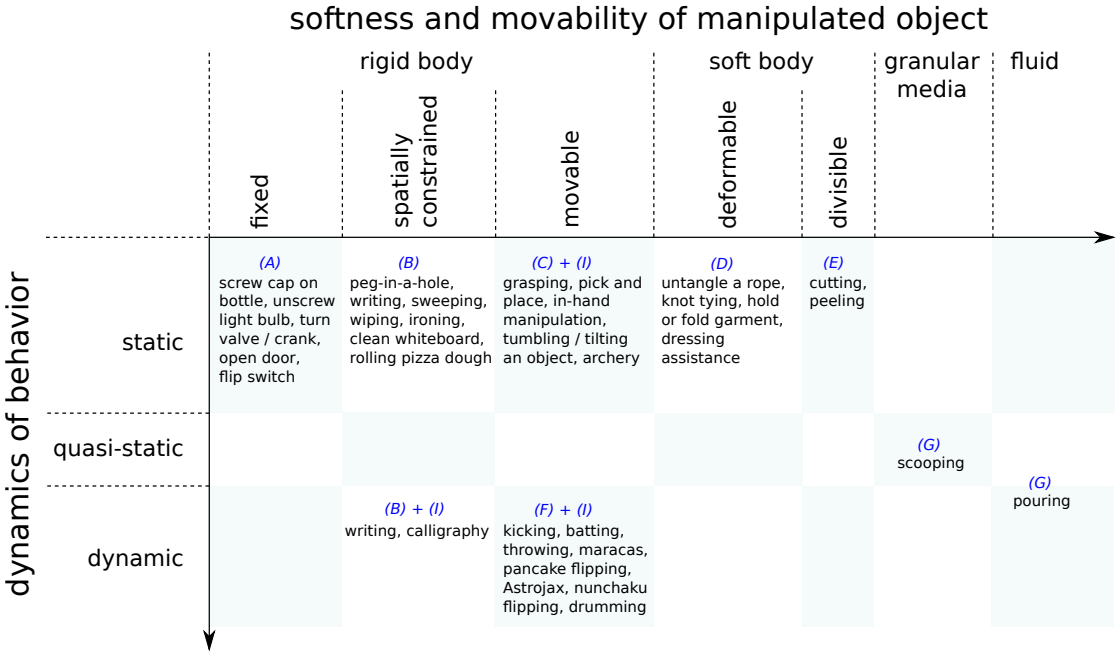


Figure 2.3.: Categorization of manipulation behaviors. Manipulation behaviors are categorized in two dimensions: softness and movability of the manipulated object and dynamics of the behavior. Blue letters indicate the corresponding subsections in Appendix A.1 *Manipulation Behaviors*.

a complex robotic system operating in the real world. (2) They were integrated in kinematically or sensorially complex robots, which includes humanoid robots or parts of humanoid robots such as legged robots or robotic arms. We only consider applications for unmanned aerial vehicles, autonomous underwater vehicles, or wheeled robots if their behaviors are relevant for humanoid robots. That excludes some recent work on deep reinforcement learning and early works that apply machine learning to robotic control such as Mahadevan and Connell [MC92], who learn a behavior to find and push a box with a wheeled robot. We also do not discuss behaviors that have only been demonstrated in simulation because of the simulation-reality gap [JHH95].

2.1.1. Manipulation Behaviors

Figure 2.3 shows the categorization of manipulation behaviors. Manipulation behaviors change the state of the robot's environment, hence, we categorized behaviors by the softness of the manipulated object and the dynamics of the behavior. This is similar to how Sanchez et al. [San+18] structured their survey about manipulation and sensing of deformable objects. We found this categorization to be useful to organize publications. It might not be easily applicable in all cases, however. For example, in case of a robot

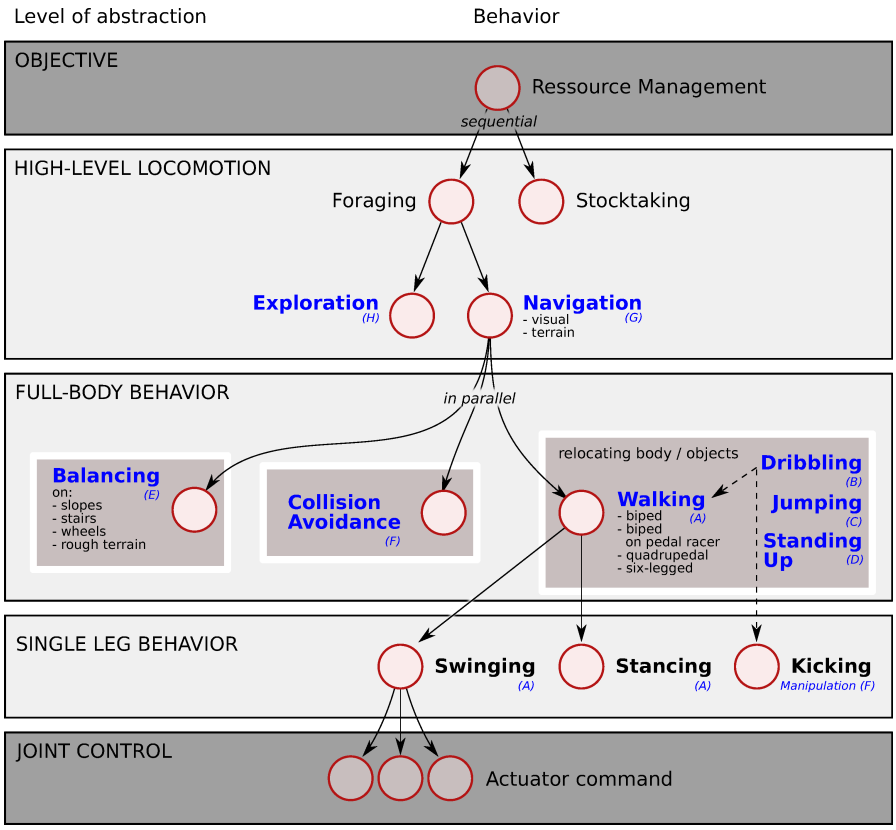


Figure 2.4.: Hierarchy of behaviors with focus on locomotion. Inspired by Arkin [Ark98, page 49]. For different levels of abstractions exemplary behaviors are presented. Concrete movements of the body, a single extremity or joint are found on lower levels in this hierarchy. While machine learning may be used on all levels and intersections, this work focuses on behavior learning above the level of joint control. Blue letters indicate subsections in Appendix A.2 *Locomotion Behaviors*. (Illustrated by Marc Otto in Fabisch et al. [Fab+20].)

that moves a catheter [Tib+14], we would have to answer the question if the catheter is the manipulated object or part of the robot. If the catheter is part of the robot, what would be the manipulated object?

2.1.2. Locomotion Behaviors

The design of locomotion behaviors is a challenge that increases with the kinematic complexity of the robot, its inherent stability, and the terrain to be traversed. Machine learning techniques can be used to provide solutions to locomotion problems, even though fundamental principles of robot locomotion are not yet fully understood [Agu+16].

Locomotion problems can be organized hierarchically based on the controlled entities (single or multiple legs, joints of the robot body) as shown in Figure 2.4. On the

lowest level, a proportional-integral-derivative (PID) controller may generate actuator commands to control the joints of a robot leg or the motors of its wheels to reach or maintain a certain position, velocity, or torque. By variation of its parameters, a joint controller can achieve useful reactive movements without knowledge of the kinematic structure. As an example, each joint can independently compensate for internal friction or a certain reflex can be triggered locally at joint level [Kue+14]. We exclude the level of joint control, as it is only modifying a given behavior generated on higher levels. Single leg behaviors, such as the swing movement, can be defined in the Cartesian space of the end-effector and thus require inverse kinematics and / or dynamics transferring the behavior's output into joint space. Behaviors that command the full body such as balancing or walking often use other behaviors that only control single legs. High-level locomotion behaviors concatenate, combine, and steer full-body behaviors. For example, navigation behavior for a humanoid robot controls the goal of a walking behavior. High-level behaviors could as well be controlled by other behaviors or overall objectives.

2.2. An Overview of Behavior Learning Approaches

There are many different approaches to behavior learning. Learning behaviors means learning sequential decision making, which is inherently more difficult than, for instance, supervised learning, because it violates the commonly used assumption of independent and identically distributed data. Samples depend on previous samples in a sequence. This often requires specialized approaches such as imitation learning or reinforcement learning, but also supervised learning, particularly self-supervised learning, can be used to learn behaviors. Furthermore, black-box optimization overlaps with reinforcement learning but is an independent field that has also been used for behavior generation.

Some highlights are among all the publications that we discuss. These are marked explicitly with this symbol on the margin, as they are relevant extensions of the repertoire of robotic behavior learning algorithms or problems that can be solved and they are crucial to understand the algorithmic development and technical challenges in the field. They also give a good impression of the advantages and disadvantages of each approach.

Although behavior learning is an incredibly interesting field, applications in robotics are often underwhelming. Although good results have already been presented in the nineties, progress in the last decades has been mostly made in controlled learning problems or with a huge amount of data and computational power—in controlled environments. A vast amount of algorithmic ideas has been explored. Nevertheless, we are still struggling to tune brittle algorithms that have many hyperparameters and do not hold up to their promises. Here, we will shed light on this field and its ideas.[1]

In the behavior learning literature we often find similar concepts but different mathematical notation. We now define the notation that we will use throughout this thesis. When we talk about behavior learning we mean that we want to learn a policy π of an *agent*. Agent is a synonym for robot for the scope of this thesis. The agent uses π to

[1] Some algorithms will be discussed in Appendix B because they are not directly relevant to this thesis.

Figure 2.5.: Kinesthetic teaching for the peg-in-a-hole problem with a UR5 robot arm.

derive actions that it takes in a specific state. The agent observes the state directly or indirectly. We will use the variable $\boldsymbol{x} \in \mathcal{X}$ for state and the variable $\boldsymbol{u} \in \mathcal{U}$ for action unlike, for example, Sutton and Barto [SB18]. Hence, a policy can, for instance, be defined as a function $\pi(\boldsymbol{x}) = \boldsymbol{u}$ (deterministic policy) or a distribution $\pi(\boldsymbol{x}|\boldsymbol{u})$ (stochastic policy). We will also see policies that define state space trajectories by a function $\pi(\boldsymbol{x}_t) = \boldsymbol{x}_{t+1}$ for discrete time steps $t \in \mathbb{N}$. These require low-level controllers that execute state transitions, that is, compute the action $\boldsymbol{u}$ from $\boldsymbol{x}_t$ and $\boldsymbol{x}_{t+1}$. Appendix C gives a more detailed overview of the mathematical notation.

2.2.1. Imitation Learning

Supervised learning can be used to learn the perception part of a behavior, the action part, or both. If actions are learned supervised, this is called imitation learning or programming by demonstration. Surveys of imitation learning have been written by Billard et al. [Bil+08], Argall et al. [Arg+09], and Osa et al. [Osa+18]. We will summarize aspects that are most relevant to this thesis.

2.2.1.1. Data Collection for Imitation Learning

If we want to apply imitation learning we have to collect a dataset that contains pairs of observed states or observed states and actions. This is mostly done by observing how a human operates a robot.

An operator can control a robot by kinesthetic teaching, which means the robot is in a compliant control mode and can be moved easily as illustrated in Figure 2.5. This requires a good model of the robot's dynamics and can not be done with all types of robots. Hence, kinesthetic teaching is mostly done with robotic arms. It is sometimes not

easy, however, to move the robot appropriately or even impossible to reach the precision and speed that the robot and the human would be capable of, if the dynamic model is not well calibrated, which even happens with commercial robot arms such as Universal Robots' UR5 or the Kuka iiwa.

An alternative to this approach is teleoperation of a robot. This can be assisted by human-computer interfaces. Zhang et al. [Zha+18a] present a system which uses a consumer-grade virtual reality hardware to let a human operate a PR2 robot in a virtual environment constructed from the perspective of the robot. They use this to teach manipulation behaviors. Similar work on a real system has been presented by Rakita et al. [RMG17] who use a 6-DOF controller for the pose of a UR5 robot arm.

The most natural way for a human to demonstrate behaviors is to perform the behavior themself without directly controlling a real or virtual robot. This can be tracked with a motion capture system and then be transferred to a robot. It is the best method to demonstrate challenging tasks such as throwing, which requires high acceleration, velocity, and smoothness, or assembly tasks, which require high precision and tactile feedback. Motion capture, however, poses the correspondence problem since the human does not directly control and observe the robotic target system.

2.2.1.2. Correspondence Problem

Demonstrated actions must be executable by the target system to apply imitation learning. Is is not possible to directly transfer joint angles from a human to a robot arm because it has different joints, degrees of freedom, and link lengths. The correspondence problem [ND02] consists of two subproblems [Arg+09]: finding the record mapping, which maps observations to a sequence of states and / or actions, and finding the embodiment mapping, which maps a recorded sequence of executable actions to the target system.

We have to find the record mapping $g_R : \mathcal{U}_T \rightarrow D$ from some not directly observable space $\mathcal{U}_T$ in which the teacher performs the demonstration (for example, joint angles of a human, muscle activity, applied torque) to a corresponding observation space D.

We have to find the embodiment mapping $g_E : D \rightarrow \mathcal{U}_L$ from the observations $d \in D$ to a corresponding action $\boldsymbol{u} \in \mathcal{U}_L$ that the learner has to perform to achieve a similar result. It is specific for the task and the target system.

Deriving the embodiment mapping, a processes that is also called motion retargeting, is a well-known problem in the computer graphics and animation community [Gle98]. In practice it is often solved manually and even individually for each character or even movement. Recently, Aberman et al. [Abe+19] present an approach based on neural networks for automatic 2D retargeting of human motion from images to other human-like characters. In the context of imitation learning for robots retargeting has been explored by Pollard et al. [Pol+02] and Michieletto et al. [MCM13] with a fixed mapping.

2.2.1.3. Imitation Learning for General Function Approximators

A lot of different policy representations have been investigated in robotics, from general function approximators (for instance, neural networks and Gaussian process regression) to more tailored representations like movement primitives and splines [KBP13].

Let us assume that we have an agent that directly or indirectly perceives the state $\boldsymbol{x}$ and has to predict an action $\boldsymbol{u}$. In imitation learning we assume that we have samples of pairs $(\boldsymbol{x}, \boldsymbol{u})$ provided by an expert. The problem of inferring a policy can be a standard supervised learning problem. In robotics we are mostly interested in $\boldsymbol{u} \in \mathbb{R}^n, n \in \mathbb{N}$ because the variables that we try to control (forces, torques, joint angles, Cartesian poses) are continuous. Hence, we do regression and we can use general function approximators to solve the problem. Typical examples are linear models or neural networks.

But consider that if we want to learn the behavior of an agent, we are mostly concerned with sequential decision making problems. Not just a single action but a sequence of actions has to be learned. In non-sequential regression problems it is often acceptable to have small errors or even a small amount of large errors. In imitation learning these errors add up. If we ignore the probabilistic nature of the state transitions for a moment, these errors will already lead the agent to states that might be different from the states that it saw during training. Standard regression algorithms assume independent and identically distributed training data. This is not the case in imitation learning, since we observe sequences of actions, in which each sample has an effect on the following samples. We can either make sure that the training data contain almost all states in which the agent might be or we need a specific training scheme. The state of the art is dataset aggregation (DAgger) that has been proposed by Ross et al. [RGB11]. DAgger uses an iterative data collection procedure: after training on an initial dataset provided by a teacher, the obtained behavior is executed and a teacher is again asked to give correct actions for observed states that have not been seen before. A new behavior is trained on the augmented dataset. This can be done for several iterations. A real-world application of DAgger has been presented by Ross et al. [Ros+13], who learn reactive obstacle avoidance in a forest for an autonomous aerial vehicle (a drone) from human control commands based on camera images. They note that although there exist devices for teleoperation it is still difficult to collect the correct control commands from a teacher. Human teachers could not correct the drone's actions in real time, hence, they replayed recorded data offline at a slower speed. Teachers also needed additional visual feedback to estimate the effect of their corrections. Crashing the drone during training had to be avoided by introducing an emergency mode in which the human operator takes complete control of the drone. A linear policy with predefined features of the camera images generates actions in this application.

For complex problems often large datasets are required. Scherzinger et al. [SRD19] learn force-based insertion with a robotic manipulator from approximately 1000 demonstrations. For a proof of concept of end-to-end imitation learning for autonomous driving from camera images to steering commands Bojarski et al. [Boj+16] record about 72 hours of training data; however, for more robust driving Xu et al. [Xu+17] learn from a dataset of 10,000 hours of driving.

2.2.1.4. Dynamical Movement Primitives

It is possible to avoid the instability of general function approximators with respect to distribution shifts by using tailored policy representations. In robotics we often use policies with guarantees for the produced trajectories. Popular examples are Dynamical Movement Primitives (DMPs) [Ijs+13; Pas+09] that represent state space trajectories.

DMPs are suited for imitation and reinforcement learning, and encode arbitrarily shapeable, goal-directed trajectories. Their advantages in comparison to other policies are:

1. They are stable trajectory representations. Slight errors in execution of the trajectory will not result in error accumulation as in general function approximators.
2. To reproduce a demonstrated movement, a one-shot learning algorithm can be used that determines the parameters of a DMP. Hence, imitation learning with DMPs is much simpler than it is for more general function approximators.
3. Movements can be easily adapted (even during execution). We can change the goal of the movement and positions of obstacles that should be avoided.

There are many variations of DMPs, which have in common that (1) they have an internal time variable (phase), which is defined by a so-called canonical system, (2) they can be adapted by tuning the weights of a forcing term, and (3) a transformation system generates goal-directed accelerations.

A canonical system generates the phase variable z which replaces explicit timing in DMPs. The values of z are generated by the first order differential equation $\tau \dot{z} = -\alpha z$, where z starts from 1 and approaches 0, τ is the duration of the movement primitive, α is some constant that has to be set such that z approaches 0 sufficiently fast.

Following Pastor et al. [Pas+09], the transformation system is a spring-damper system and generates a goal-directed motion that is controlled by a phase variable z and modified by a forcing term f. Two first order differential equations

$$\begin{aligned} \tau \dot{y} &= v \\ \tau \dot{v} &= K \underbrace{(g - y)}_{\text{distance to goal}} - D \overbrace{v}^{\text{velocity}} - K \underbrace{(g - y_0)}_{\text{scaling}} z + K \overbrace{f(z)}^{\text{learnable}} \end{aligned}$$

define a transformation, where $y, \dot{y} = v/\tau$, and $\ddot{y} = \dot{v}/\tau$ are position, velocity, and acceleration. y_0 is the start and g is the goal of the movement. v_t is an auxiliary variable. K and D are constants that have to be set for critical damping, that is $D = 2\sqrt{K}$ must be satisfied. Common values are $D = 20, K = 100$.

The forcing term is a linear model of the form

$$f(z) = z \cdot \frac{\sum_i \Phi_i(z) \cdot w_i}{\sum_i \Phi_i(z)}$$

with parameters $\boldsymbol{w} = (w_1, w_2, \ldots)^T$ that control the shape of the trajectory. Influence of the forcing term decays as the phase variable approaches 0. $\Phi_i(z) = \exp(-\frac{d_i}{2}(z - c_i)^2)$

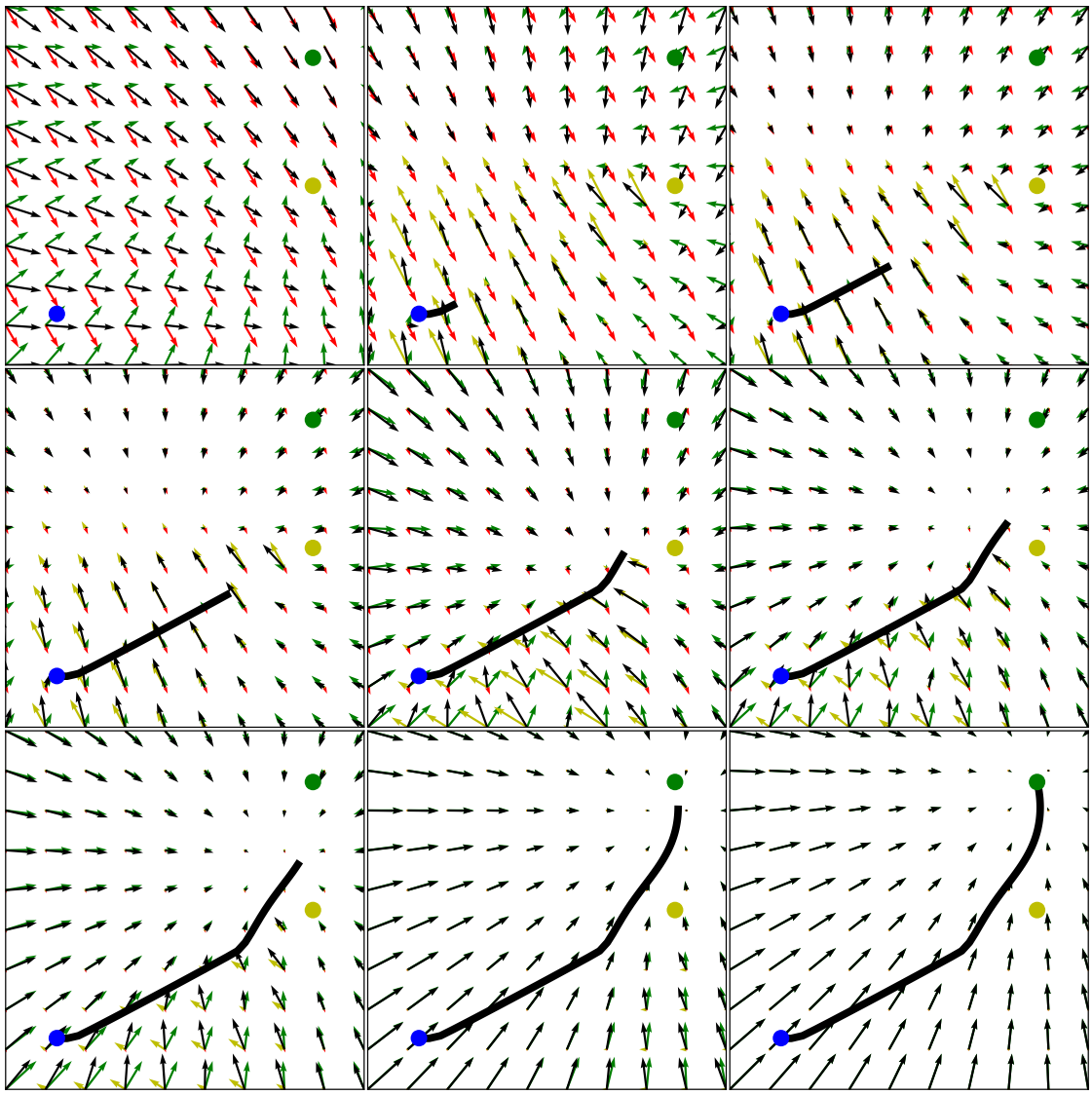


Figure 2.6.: Illustration of two-dimensional DMP as a potential field. The temporal evolution of a DMP is split up into 9 steps from left to right and top to bottom. The blue dot represents the start, the green dot the goal, and the yellow dot an obstacle. For each position in the two-dimensional space the DMP defines an acceleration (black arrows) that is generated by superimposing the forcing term (red arrows), the transformation system (green arrows), and obstacle avoidance (yellow arrows). The resulting trajectory is indicated by the black line. In the end the main influence comes from the transformation system, which guarantees that the goal is reached.

are radial basis functions with constant d_i (widths) and c_i (centers; equally spaced in log-phase domain).

So far we only described DMPs for one-dimensional task spaces. If we want to control multiple joints or positions in task space we will handle each dimension separately with a DMP. Figure 2.6 illustrates a two-dimensional DMP as a potential field with several superimposed components. For Cartesian trajectories we need to handle orientations differently because we cannot linearly interpolate between orientation representations. Averaging, for example, between two unit quaternions would result in a quaternion that does not have the norm one anymore and, hence, is not a valid orientation. Ude et al. [Ude+14] suggested a variant of DMPs that solves this problem for quaternions and improves a previously suggested approach by Pastor et al. [Pas+09].

The weights w_i can be learned by imitation or reinforcement learning. We can guarantee that the goal g will be reached at the end of the movement. Therefore, reinforcement learning algorithms will only explore movements that reach the goal. If this is a necessary condition for a good policy, the number of useless policies will be reduced.

In this thesis we often use a variant of DMPs that has been developed by Mülling et al. [MKP11; Mül+13], which allows us to specify a velocity at the end of the movement as a metaparameter. From a high-level perspective we can use these DMPs as policies

$$\boldsymbol{x}_{t+1} = \pi_{\boldsymbol{v},\boldsymbol{w}}(\boldsymbol{x}_t, t), \quad \boldsymbol{v} = (\boldsymbol{x}_0, \boldsymbol{g}, \dot{\boldsymbol{g}}, \tau)\,,$$

where $\boldsymbol{x}_t$ is the state (position, velocity, and acceleration) at time t, $\boldsymbol{w}$ are the weights of the forcing term and $\boldsymbol{v}$ are the following metaparameters: $\boldsymbol{x}_0$ is the initial state, $\boldsymbol{g}$ is the final position, $\dot{\boldsymbol{g}}$ the desired final velocity, and τ is the duration of the movement.

Note that this kind of policy defines a state space trajectory and requires a low-level controller that generates the appropriate actions such that the state transition from $\boldsymbol{x}_t$ to $\boldsymbol{x}_{t+1}$ is performed.

Apart from the standard formulation of DMPs there are several variations. Matsubara et al. [MHM10] introduce Stylistic Dynamic Movement Primitivess (SDMPs). These reduce the dimensionality of movement primitives learned from multiple demonstrations via singular value decomposition to identify and represent different styles of similar motions. SDMPs can interpolate between those styles by changing the style parameters.

Probabilistic movement primitivess (ProMPs) [Par+13] define distributions of trajectories that are learned from several demonstrations. They do not have an attractor point such as the goal in a DMP, but they can be conditioned on via points that lie within the distribution.

The drawback of movement primitives and similar trajectory generators is that they define state-space trajectories without directly specifying actions that ensure transition between the states. They require controllers to generate these actions. For instance, a joint position controller would be required if the state includes joint positions. Hence, we cannot easily integrate rich sensory information such as images, that are not directly controllable. Thus, movement primitives cannot continuously control based on this kind of sensor input [Haa+18a].

2.2.1.5. Other Trajectory Generators

Besides movement primitives and standard trajectory representations such as splines there are several other trajectory generators that are sometimes used in robotics. An example is the Stable Estimator of Dynamical Systems (SEDS) [KB11], an approach to represent demonstrated trajectories as a time-invariant dynamical system with the goal of the movement as attractor point. SEDS build on Gaussian mixture regression and add several constraints that have to be fulfilled during learning. Hence, they need a complex optimization procedure, which can be implemented for imitation learning but complicates reinforcement learning. An advantage of SEDS over standard DMPs is that they can encode multiple demonstrations. A similar approach has been presented by Lemme et al. [Lem+14] for an extreme learning machine, which is a neural network of which only the last layer is trained.

2.2.1.6. Generalizing Imitated Trajectories

The problem of learning more broadly applicable skills has been approached from different perspectives. DMPs themselves have been designed to generalize over some metaparameters such as the duration of the movement, the start position, and the final position. Additionally, some DMP variants are able to generalize over velocities at the end of the movement [MKP11; Mül+13]. We often want to be able to generalize over other parameters of a task though.

Task-Parameterized Gaussian Mixture Models (TP-GMMs) learn a probabilistic representation of multiple demonstrated trajectories with different linear task parameters such as Cartesian transformations [Cal16]. They generalize over these task parameters that could represent positions of objects in the environment or via points. They can also mix between task parameters and different demonstrations. The requirements on the task parameters are limiting, however. What if, for instance, we want to generalize ball throwing over the target positions? TP-GMMs cannot do this because they would require a model of the relation between the ball trajectory and the robot's motion and they would require this model to be linear.

Designing skills that generalize over more complex task parameters with a nonlinear, indirect relation to the movement primitive is not straightforward. This is why various machine learning techniques have been used to infer so-called upper-level policies. Ude et al. [Ude+10] used a form of self-imitation to generalize the task of throwing a ball at a desired target position. First, they generate a number of throws with different DMP parameterizations (τ, $\boldsymbol{g}$, $\boldsymbol{w}$, and release times of the ball) and measure the final ball positions. Then, they use a mixture of locally weighted regression [CD88] and Gaussian process regression [RW05] to learn mappings from the position that has been hit on the ground to corresponding parameters that generated the throw. Thus, the problem has been reduced to a regression problem. The same approach has been used for reaching tasks and drumming. Kronander et al. [KKB11] propose a similar method to play minigolf. As underlying policy representation SEDS [KB11] has been used and another set

of metaparameters is learned with Gaussian process regression and Gaussian mixture regression. Both methods can be regarded as generalized imitation learning algorithms.

2.2.2. Black-Box Optimization

In black-box optimization we do not know the correct actions. Instead, we have an objective function $f(\boldsymbol{\theta})$ that we can sample and which tells us how well a given parameter vector $\boldsymbol{\theta} \in \Theta \subseteq \mathbb{R}^n$ with $n \in \mathbb{N}$ performs. We cannot compute the gradient of f. We search for parameters $\boldsymbol{\theta}$ such that

$$\arg\min_{\boldsymbol{\theta}} f(\boldsymbol{\theta}).$$

We can apply this concept to behavior generation in robotics. To do this we have to define a cost function that evaluates a robot's behavior, we need a parameterized representation of a behavior, and then we minimize the cost by changing the parameters of the behavior. This approach is similar to policy search in reinforcement learning as we will see in the next section. Thus, we will adopt the terminology of reinforcement learning here, that is, we do not minimize cost but rather maximize return, the return will most likely be stochastic in the real world, and the behavior is a parameterized policy.

Objective functions have properties that we can use to distinguish them. An objective function is *separable* if we can obtain the optimum of each component of $\boldsymbol{\theta}$ independently of other components [Han+08]. An objective function is *ill-conditioned* if steps with the same length in different directions of the search space change the objective function value considerably differently (typically orders of magnitude differently) [Han+08]. An objective function is *multi-modal* if it has multiple local and / or global optima.

2.2.2.1. Evolution Strategy

One of the earliest concepts in this field that has been developed and is still used today is Evolution Strategy (ES) [Rec71]. Recently, Salimans et al. [Sal+17] showed that ES is still better than some state-of-the-art reinforcement learning algorithms in several reinforcement learning benchmarks. In its simplest form, ES uses an isotropic Gaussian search distribution with mean $\boldsymbol{\mu}$ and covariance $\sigma^2\boldsymbol{I}$ from which we can sample parameters

$$\boldsymbol{\theta} \sim \mathcal{N}(\boldsymbol{\mu}, \sigma^2\boldsymbol{I})$$

to evaluate the objective function. We will do this $N \in \mathbb{N}$ times, but we will use only the best samples (fewer than N) to estimate the new mean and variance of the search distribution. This completes one so-called generation and will be repeated until convergence. ES is a local search approach. It is not guaranteed that we will reach a global optimum, although chances increase with larger N and a larger initial variance. ES only changes the policy after each evaluation of the objective function (episode in reinforcement learning terms) and the search distribution is only updated after N episodes.

The Cross-Entropy Method (CEM) [Rub99] is a variant of ES that estimates a full covariance matrix and has been used for policy search [MRG03]. A state-of-the-art

variant of ES is Covariance Matrix Adaptation Evolution Strategies (CMA-ES) [HO01], which uses a full covariance matrix to define its exploration behavior and several advanced strategies to update the covariance matrix: (1) it maintains a separate scaling of the covariance that is updated based on the consistency of the direction of previous update steps, (2) it updates correlations between search space dimensions based on previous update steps, (3) it keeps information about correlations from previous iterations, and (4) it assigns individual weights to samples. The drawback of standard CMA-ES is that it stores the full covariance, hence, the space complexity is quadratic in the number of policy parameters. From time to time it also computes the Cholesky decomposition of the covariance matrix. This is done in a way such that the computational complexity of this operation over the whole optimization is on average quadratic per iteration, although it is in fact an operation of cubic computational complexity. Hence, standard CMA-ES is not the best option for a large number of parameters, that is, for more than 10,000. Deep neural networks sometimes have even more parameters. For these cases low-rank approximations of the covariance such as limited memory CMA-ES [Los17] could be used. Furthermore, black-box optimization problems with a low number of parameters are often better solved with other optimizers such as Nelder-Mead [NM65].

In optimization we can often trade function evaluation against more computational cost on the side of the optimizer with a surrogate model. A surrogate model approximates the objective function, that is, it represents a mapping from parameters to their expected function value. For CMA-ES a local ranking support vector machine [LSS10] and a global quadratic surrogate model [Han19] have been proposed.

CMA-ES uses many heuristics to update the search distribution. An alternative is Natural Evolution Strategies (NES) [Wie+14]. NES estimates the gradient of the expected fitness under the search distribution from samples, that is,

$$\boldsymbol{g} = \nabla_{\boldsymbol{\omega}} \mathbb{E}_{\boldsymbol{\theta} \sim \pi_{\boldsymbol{\omega}}(\boldsymbol{\theta})} \left[f(\boldsymbol{\theta}) \right] = \mathbb{E}_{\boldsymbol{\theta} \sim \pi_{\boldsymbol{\omega}}(\boldsymbol{\theta})} \left[f(\boldsymbol{\theta}) \nabla_{\boldsymbol{\omega}} \ln \pi_{\boldsymbol{\omega}}(\boldsymbol{\theta}) \right] \qquad \text{(log-derivative trick)}$$

is estimated from N samples $(\boldsymbol{\theta}_i, f(\boldsymbol{\theta}_i))$ by

$$\mathbb{E}_{\boldsymbol{\theta} \sim \pi_{\boldsymbol{\omega}}(\boldsymbol{\theta})} \left[f(\boldsymbol{\theta}) \nabla_{\boldsymbol{\omega}} \ln \pi_{\boldsymbol{\omega}}(\boldsymbol{\theta}) \right] \approx \frac{1}{N} \sum_{i=1}^{N} f(\boldsymbol{\theta}_i) \nabla_{\boldsymbol{\omega}} \ln \pi_{\boldsymbol{\omega}}(\boldsymbol{\theta}_i) \text{ with } \boldsymbol{\theta}_i \sim \pi_{\boldsymbol{\omega}},$$

for fixed parameters $\boldsymbol{\omega}$ of the search distribution $\pi_{\boldsymbol{\omega}}$, which are often mean and covariance of a Gaussian distribution. This is called the score function estimator of the gradient [Moh+19]. This gradient is not directly used to update the search distribution because this might be slow or unstable. We compute the natural gradient, that is,

$$\min_{\Delta \boldsymbol{\omega}} \mathbb{E}_{\boldsymbol{\theta} \sim \pi_{\boldsymbol{\omega} + \Delta \boldsymbol{\omega}}(\boldsymbol{\theta})} \left[f(\boldsymbol{\theta}) \right], \text{ subject to } D_{\mathrm{KL}}(\pi_{\boldsymbol{\omega} + \Delta \boldsymbol{\omega}} \parallel \pi_{\boldsymbol{\omega}}) = \epsilon,$$

with $\Delta \boldsymbol{\omega} = \boldsymbol{A} \boldsymbol{g}$ and a small increment ϵ. The constraint limits information loss measured by the Kullback-Leibler (KL) divergence between successive search distributions. It turns out that the matrix $\boldsymbol{A}$ that minimizes the above objective for small ϵ can be approximated by the inverse $\boldsymbol{F}^{-1}$ of the Fisher information matrix

$$\boldsymbol{F} = \mathbb{E}_{\boldsymbol{\theta} \sim \pi_{\boldsymbol{\omega}}} \left[\nabla_{\boldsymbol{\omega}} \ln \pi_{\boldsymbol{\omega}}(\boldsymbol{\theta}) \nabla_{\boldsymbol{\omega}} \ln \pi_{\boldsymbol{\omega}}(\boldsymbol{\theta})^T \right],$$

which we can estimate from the same samples that we obtained to compute the gradient. We can then update the parameters of the search distribution through

$$\boldsymbol{\omega} \leftarrow \boldsymbol{\omega} - \alpha \hat{\boldsymbol{F}}^{-1} \hat{\boldsymbol{g}},$$

with estimates of $\boldsymbol{F}$ and $\boldsymbol{g}$ and a learning rate α. This will be done until convergence. Although NES is not better than CMA-ES, it is theoretically more sound and has interesting similarities to reinforcement learning algorithms. Furthermore, Akimoto et al. [Aki+10] show that a specific parameterization of CMA-ES follows the natural gradient.

2.2.2.2. Bayesian Optimization

Bayesian Optimization (BO) [BCd10] is an approach to optimization that fully builds on a surrogate model, which not only predicts the expected function value but also its uncertainty in form of a standard deviation. These can be used to define a so-called acquisition function. A popular acquisition function is the Upper Confidence Bound (UCB): the expected function value plus the standard deviation multiplied by a constant, which is a hyperparameter of the algorithm. In BO we determine query points by maximizing the acquisition function with another optimization algorithm. It is, hence, costly but also sample-efficient and can be a global optimizer if the optimizer of the acquisition function searches globally. Dividing Rectangles (DIRECT) [JPS93] does this and it can be combined with a local optimizer for refinement. BO is particularly effective for a low number of parameters. It is computationally expensive and global optimization might be infeasible for a large number of parameters [SS19].

2.2.2.3. Gradient-Based Optimization with Finite Differences

It is possible to estimate the gradient of an objective function by finite differences and use gradient-based optimizers such as Broyden-Fletcher-Goldfarb-Shanno (BFGS) [Bro70; Fle70; Gol70; Sha70], Limited-memory BFGS (L-BFGS) [Noc80], or L-BFGS for Bound Constrained Optimization (L-BFGS-B) [Byr+95a]. If the gradient estimates are not accurate enough this often leads to premature convergence [Los17].

2.2.2.4. Black-Box Behavior Optimization

Episodic policy search algorithms are often similar to black-box optimization algorithms. Examples are the policy search algorithm Natural Actor-Critic (NAC) [PVS05] and the corresponding black-box optimization algorithm NES [Wie+14]. Hence, there are many applications of black-box optimization algorithms to behavior learning problems.

A lot of impressive results have been achieved with black-box optimization algorithms in reinforcement learning domains such as walking with a robot, which has been tackled with BO to optimize parameters of a finite state machine [Cal+16] and CMA-ES in combination with model-based control [Geh+14]. Another example is playing video games, which has been tackled with ES [Sal+17; CLH18; Fuk+19] and CMA-ES [HS18]. One of the earliest applications of CMA-ES in policy search was in pole balancing with fully and partially observable states [HI08; HI09].

2.2.3. Reinforcement Learning with Value Functions

Reinforcement learning [SB18] is neither supervised nor unsupervised. It is between both. Its goal is to infer a policy π that lets an agent select an appropriate action for each state. Correct actions are unknown but hints are given by a reward function that rates the actions of an agent. Surveys on reinforcement learning (RL) in robotics have been published by Kober et al. [KBP13] and Kormushev et al. [KCC13]; however, they do not include the latest developments.

Imitation learning maximizes $\int_{\mathcal{D}} p^{\pi}(\boldsymbol{x}, \boldsymbol{u}) d(\boldsymbol{x}, \boldsymbol{u})$, that is, the probability of the optimal actions of which we have a finite set of samples $(\boldsymbol{x}, \boldsymbol{u}) \in \mathcal{D}$ from a teacher. Reinforcement learning maximizes $\int_{\mathcal{X}} \int_{\mathcal{U}} p^{\pi}(\boldsymbol{x}, \boldsymbol{u}) r(\boldsymbol{x}, \boldsymbol{u}) d\boldsymbol{u}\, d\boldsymbol{x}$, that is, the probability of actions weighted by their reward [DH97].[2] In both cases $p^{\pi}(\boldsymbol{x}, \boldsymbol{u}) = \rho^{\pi}(\boldsymbol{x}) \pi(\boldsymbol{u}|\boldsymbol{x})$ with the state visitation distribution ρ^{π} of the policy π.

Reinforcement learning algorithms can be roughly divided into three categories: value-function based algorithms, policy search, and actor-critic algorithms, which combine both other approaches. Policy search algorithms try to infer the policy π directly by maximizing a given optimality criterion such as the expected return. Value-function based methods first learn a value function that estimates the expected return according to some optimality criterion. Under certain conditions the policy can be directly derived from the value function and we do not need to represent the policy explicitly. Actor-critic algorithms learn a value function and a separate policy. The value function is then a surrogate model for improvement of the policy. There are several main branches of current RL research. We will take a closer look at the most relevant ones for this thesis, that is, for the problem of learning behaviors for robots.

2.2.3.1. Value Function, Return, and Reward

In this section we will lay the foundation for more practical algorithms that we will discuss in the following sections. Reinforcement learning is a field that deals with optimal sequential decision making. A sequence of interactions with the environment that leads to a terminal state is called episode. The term value function refers to the state value function

$$V^{\pi}(\boldsymbol{x}_t) = \mathbb{E}_{\pi}\left[R_t\right],$$

the expectation of the return R_t of a policy π aggregated from state $\boldsymbol{x}_t$. It indicates how good it is for an agent that operates under the given policy to be in this state. The index t is used to indicate the position in the sequence of states, actions, or rewards. The return accumulates rewards that will be obtained in future steps. A frequently used model is the infinite horizon discounted return, that is,

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots$$

with a parameter $\gamma \in (0, 1]$ that reduces the influence of future rewards r_{t+k} exponentially. The reward signal informs the agent after each step about its previous performance. It

[2] Although this is a simplification that does neither apply to all imitation learning algorithm nor to all reinforcement learning algorithms.

is often not possible to directly relate a reward with a specific action though. Because we will often deal with episodes of fixed length in this thesis, we will also often use the infinite horizon model, that is,

$$R_t = r_{t+1} + r_{t+2} + r_{t+3} + \dots$$

To be precise, there are several types of value functions: the state value function $V^\pi(\boldsymbol{x})$, a state-action value function $Q^\pi(\boldsymbol{x}, \boldsymbol{u})$, in which the action $\boldsymbol{u}$ taken in state $\boldsymbol{x}$ is an argument and not necessarily selected according to π in state $\boldsymbol{x}$, and the advantage function $A^\pi(\boldsymbol{x}, \boldsymbol{u}) = Q^\pi(\boldsymbol{x}, \boldsymbol{u}) - V^\pi(\boldsymbol{x})$ that indicates how much better (or worse) it is to execute an action $\boldsymbol{u}$ in a state $\boldsymbol{x}$ in comparison to the average state value.

2.2.3.2. Markov Decision Process

The standard reinforcement learning formulation requires that the problem is a Markov decision process (MDP), that is, state transitions and rewards only depend on the last state and the last action of the agent (which have to be fully observable) but might be stochastic [SB18, Chapter 3]. With small modifications, many methods that we present here can also be applied in partially observable MDPs, in which we cannot directly observe the complete state.

2.2.3.3. Temporal-Difference Learning

Temporal-difference (TD) learning [Sut88] combines Monte Carlo estimation of a value function, that is, sampling a reward, with bootstrapping based on a previous (or initial) estimate of the value function. Iterating

$$\hat{V}_{k+1}(\boldsymbol{x}_t) = \hat{V}_k(\boldsymbol{x}_t) + \alpha \left[r_{t+1} + \gamma \hat{V}_k(\boldsymbol{x}_{t+1}) - \hat{V}_k(\boldsymbol{x}_t) \right]$$

will converge ($k \to \infty$) to the true state value function of the sampling policy if we visit every state infinitely often. The term $\delta = r_{t+1} + \gamma \hat{V}_k(\boldsymbol{x}_{t+1}) - \hat{V}_k(\boldsymbol{x}_t)$ is called temporal difference error. While pure Monte Carlo estimates allow updates of the value function only after a complete episode, TD learning updates the value function after each step in an episode.

If we want to improve the policy, one option is to approximate the state-action value function $\hat{Q}$ and interleave policy improvement based on the current estimate of $\hat{Q}$ with updating $\hat{Q}$. In Q-learning [Wat89], the foundation of many state of the art reinforcement learning algorithms, the policy selects actions based on the current best estimate of the state-action value function with $\pi(\boldsymbol{x}) = \arg\max_{\boldsymbol{u}} \hat{Q}(\boldsymbol{x}, \boldsymbol{u})$ and we try to iteratively converge to the state-action value function $Q^*(\boldsymbol{x}, \boldsymbol{u})$ of the best policy π^* with TD learning. Convergence is guaranteed, if we ensure that each state-action pair is visited infinitely often.

Q-learning is an off-policy reinforcement learning algorithm, as it uses a behavior policy β for exploration and a target policy π for which the state-action value function is approximated. We can use stochastic exploration in the behavior policy β. On-policy

algorithms do not make this distinction. They have one policy π for exploration and estimate a corresponding value function. The on-policy counterpart of Q-learning is State Action Reward State Action (SARSA) [RN94], which uses the same policy π for exploration and for estimation of the state-action value function.

Monte Carlo approaches (TD(1)) perform updates of the value function only at the end of an episode, but they update the value function for each visited state-action pair based on the Monte Carlo estimate of the expected return, that is, the measured return. One-step TD methods (SARSA and Q-learning; also TD(0)) update one state-action value per time step and are, thus, usually faster on stochastic tasks [SB18]. TD(λ) [Sut88] with $\lambda \in [0, 1]$ interpolates between the two approaches. For delayed rewards $\lambda \in (0, 1)$ learns usually faster than TD(0) methods.

Q-learning overestimates the state-action value function, as it uses the maximum state-action value as an approximation for the maximum expected state-action value, hence, Hasselt [Has10] propose Double Q-learning to alleviate this problem. Double Q-learning decouples the selection of the best action from computing its state-action value by using two separate estimates of the state-action value function.

2.2.3.4. Function Approximation

In its simplest form, reinforcement learning can approximate value functions for discrete state and action spaces with a table. When the state or action space is too large to store value functions in tables, it is a better option to approximate a value function with a function approximator. In the beginning this has been done mostly with models that are linear in their parameters because these tend to produce more consistent and stable predictions and their convergence can be guaranteed [Bai95; BB96]. Under the infinite horizon discounted return criterion we can approximate the state value function by minimizing the mean squared Bellman error (MSBE)

$$L(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} \left[r_i + \gamma \hat{V}_{\boldsymbol{\theta}}(\boldsymbol{x}_i') - \hat{V}_{\boldsymbol{\theta}}(\boldsymbol{x}_i) \right]^2$$

estimated from samples (x_i, r_i, x_i') with a previous estimate $\hat{V}_{\boldsymbol{\theta}}$ with respect to the function approximator's parameters $\boldsymbol{\theta}$ [Bai95], for instance, through gradient descent. This results in an update rule for the function approximator's parameters that is similar to the standard TD update for a tabular value function and can also be extended to complex function approximators such as neural networks at the cost of losing convergence guarantees.

2.2.3.5. Value Function Estimates

Estimating value functions may be prone to bias and variance. Assuming the infinite horizon discounted reward model, we can estimate $Q^{\pi}(\boldsymbol{x}_t, \boldsymbol{u}_t)$ with an approximation $\hat{V}^{\pi}(\boldsymbol{x})$ of the state value function via $\hat{Q}^{\pi}(\boldsymbol{x}_t, \boldsymbol{u}_t) = r_t + \gamma \hat{V}^{\pi}(\boldsymbol{x}_{t+1})$ or via Monte Carlo $\hat{Q}^{\pi}(\boldsymbol{x}_t, \boldsymbol{u}_t) = R_t = \Sigma_{t'=t}^{\infty} \gamma^{t'-t} r_{t'}$, where r_t and, hence, R_t are unbiased but have high

variance. An approximation $\hat{V}^{\pi}(\boldsymbol{x})$ of the value function with a function approximator introduces bias but no variance. A generalization of both cases is the n-step truncated Monte Carlo estimate $\hat{Q}^{\pi}(\boldsymbol{x}_t, \boldsymbol{u}_t) = \left[\Sigma_{t'=t}^{t+n-1} \gamma^{t'-t} r_{t'}\right] + \gamma^n \hat{V}^{\pi}(\boldsymbol{x}_{t+n})$ based on a function approximator $\hat{V}^{\pi}$, which allows to interpolate between both cases by setting the horizon n to balance bias and variance.

If we want to include old off-policy samples into estimates of V^{π} or Q^{π} we have to use importance sampling. The V-trace estimator [Esp+18] does this and is state of the art, as it improves on previous approaches such as the Retrace estimator [Mun+16]. V-trace uses truncated importance sampling to estimate V^{π} by

$$v_t = \hat{V}(\boldsymbol{x}_t) + \sum_{t'=t}^{t+n-1} \gamma^{t'-t} \left(\prod_{i=t}^{t'-1} c_i \right) \delta_{t'}, \tag{2.1}$$

where $\hat{V}$ is a neural network, $\delta_t = \rho_t(r_t + \gamma \hat{V}(\boldsymbol{x}_{t+1}) - \hat{V}(\boldsymbol{x}_t))$ is the temporal difference error, β is the behavior policy, π the target policy, and $\rho_t = \min(\overline{\rho}, \frac{\pi(\boldsymbol{u}_t|\boldsymbol{x}_t)}{\beta(\boldsymbol{u}_t|\boldsymbol{x}_t)})$ and $c_t = \min(\overline{c}, \frac{\pi(\boldsymbol{u}_t|\boldsymbol{x}_t)}{\beta(\boldsymbol{u}_t|\boldsymbol{x}_t)})$ are truncated weights for importance sampling. In the special case of $\pi = \beta$ (on-policy), with $\overline{c} \geq 1$, $c_i = 1$, $\rho_t = 1$, then

$$v_t = \left[\sum_{t'=t}^{t+n-1} \gamma^{t'-t} r_{t'} \right] + \gamma^n \hat{V}(\boldsymbol{x}_{t+n})$$

reduces to n-step truncated Monte Carlo estimate of V^{π}. If $\overline{\rho} \to \infty$ the fixed point of iteration is V^{π}. If $\overline{\rho} \to 0$ the fixed point is V^{β}. Otherwise the fixed point is between V^{π} and V^{β}. $\overline{c}$ controls how much δ_t impacts the update of the value function at a previous step, which can be used to reduce variance and does not change the solution. This kind of estimators will become relevant especially for policy gradient and actor-critic algorithms.

2.2.4. Policy Search with Movement Primitives

Although there are works that apply standard RL on real robotic systems, it has been struggling with challenging problems posed by the real world for a long time: continuous state and action spaces, complex dynamics, noisy sensors, sparse rewards, and above all the demand for sample efficiency. Policy search can use the strong inductive bias of movement primitives to learn directly on real robots.

In policy search, we maximize the expected return, that is, we optimize parameters $\boldsymbol{\theta} \in \mathbb{R}^n$ of a policy $\pi_{\boldsymbol{\theta}}$ through

$$\arg\max_{\boldsymbol{\theta}} \mathbb{E}\left[R(\boldsymbol{\theta})\right],$$

where $R(\boldsymbol{\theta})$ is the return that tells the agent how well the policy performed [DNP13]. We write $R(\boldsymbol{\theta})$ for $R_1 = r_1 + r_2 + \ldots$ because we can modify the expected return only through $\boldsymbol{\theta}$. Immediate rewards are stochastic but only depend on the policy, which we define through $\boldsymbol{\theta}$, and the environment dynamics, which we assume to be stochastic but fixed. Deisenroth et al. [DNP13] provide an introduction to policy search and Sigaud and Stulp [SS19] a recent survey.

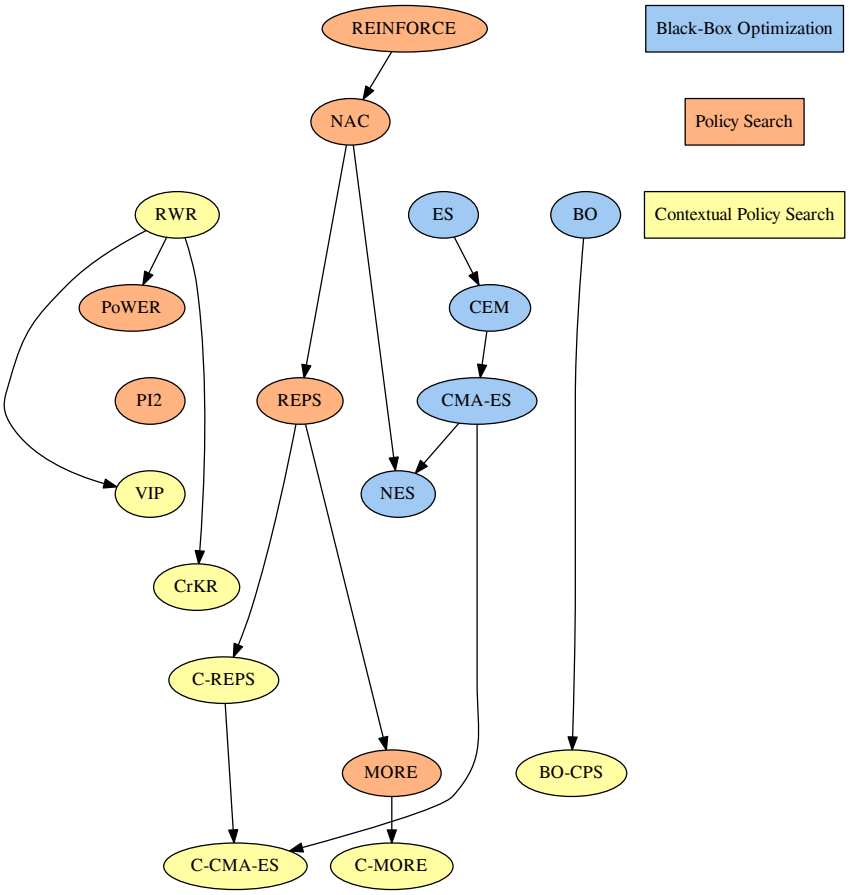


Figure 2.7.: Overview of policy search algorithms for movement primitives. Each ellipse represents an algorithm. We can categorize algorithms as black-box optimizers, direct policy search, and contextual policy search. Arrows indicate connections between algorithms (a strong influence of one algorithm on the development of the other). Algorithms above others in a chain have been developed earlier.

Movement primitives are common policy representations in robotics. Several policy search algorithms have been proposed that are specifically tailored to learn movement primitives, although most of them can also be applied to other policies. In DMPs, the policy parameters $\boldsymbol{\theta}$ could be the weights of the forcing term, metaparameters, a combination of both, or any combination of those of a sequence of multiple DMPs. The following policy search algorithms have been used mainly with movement primitives. Figure 2.7 shows connections between algorithms that we discuss in this section.

2.2.4.1. Policy Gradients

Peters et al. [PVS05] and Peters and Schaal [PS08a] presented an algorithmic milestone in reinforcement learning for robotic systems. They used a robot arm with 7 DOF to play tee-ball, a simplified version of baseball, where the ball is placed on a flexible shaft. Their solution combines imitation learning by kinesthetic teaching with a DMP and policy search, which is an approach that has been used in many following works. The goal was to hit the ball so that it flies as far as possible. The reward for policy search penalizes squared accelerations and rewards the distance. The distance is obtained from an estimated trajectory computed with trajectory samples that are measured with a vision system. An inverse dynamics controller has been used to execute motor commands. About 400 episodes were required to learn a successful batting behavior.

In their work, Peters et al. [PVS05] used NAC. Similarly to NES, NAC uses the score function estimator to determine the gradient of the return and estimates the Fisher information matrix to compute the natural gradient. In NAC, however, the exploration noise is different in each step of an episode, as we use a stochastic policy to generate actions. In NES we explore in the parameter space of a policy and change the parameters only after the end of an episode. In addition, an estimate of the value function with a linear function approximator is used to reduce the variance of the gradient. As many other gradient-based optimization approaches, NAC has a learning rate that has to be set. Heidrich-Meisner and Igel [HI08] show that CMA-ES outperforms NAC with neural network policies on a pole balancing problem. Only when the search distribution is initialized close to the optimum, NAC can be more sample-efficient than CMA-ES.

2.2.4.2. Reward-Weighted Self-Imitation

Reward-Weighted Regression (RWR) [PS07] and Policy Learning by Weighting Exploration with the Returns (PoWER) [KMP08; KP09; KP11] are based on the principle of reward-weighted self-imitation. RWR is an expectation-maximization approach that alternates between computing an expected reward (expectation step) and updating the policy and a reward transformation (maximization step). Specifically, RWR uses a weighted maximum likelihood approach to estimate a parameter-linear policy $\pi_{\boldsymbol{\theta}}(\boldsymbol{u}|\boldsymbol{x})$ that maps from nonlinear transformations of the state $\boldsymbol{\phi}(\boldsymbol{x})$ to actions $\boldsymbol{u}$. Sample weights are defined based on the reward r through normalization of $d(r) = \tau \exp(-\tau r)$ so that they sum up to 1, with τ estimated in the maximization step as $\tau \leftarrow \left(\sum_{i=1}^{N} d(r_i)\right) / \left(\sum_{i=1}^{N} d(r_i) r_i\right)$ based on $d(r_i)$ computed with the previous τ. The main advantage of RWR over NAC is that it does not have a learning rate.

In contrast to the reward-attracted RWR, Variational Inference for Policy Search (VIP) [Neu11] is a similar cost-averse algorithm. Neumann [Neu11] shows that RWR minimizes the KL divergence to the best possible policy in the expectation step. Since the KL divergence is asymmetric, he investigates the case of interchanged arguments. This leads to a more computationally expensive algorithm that can deal better with multi-modal objectives, as it focuses on one optimum.

PoWER adds state-dependent exploration noise to RWR. In addition, the best experiences with respect to their return are kept in memory and reused. A restriction of PoWER is that the reward is not scaled automatically but a fixed reward transformation has to ensure that rewards are positive.

PoWER has been used to solve challenging problems such as the game ball-in-a-cup, in which a ball is attached to a cup by a string and the robot has to catch the ball with the cup by moving only the cup. Even humans require a considerable amount of trials to solve the problem. Kober et al. [KMP08] and Kober and Peters [KP09] demonstrate that a successful behavior can be learned on a SARCOS arm and a Barret WAM. An approach similar to Peters et al. [PVS05] has been used: imitation learning with a DMP from motion capture or kinesthetic teaching and refinement through PoWER. In addition, the policy takes the ball position into consideration. A perceptual coupling is learned to mitigate the influence of minor perturbations of the end-effector that can have a strong influence on the ball trajectory. A successful behavior is learned after 75 episodes.

The problem of flipping a pancake with a pan has been solved by Kormushev et al. [KCC10b] with the same methods: a policy that is similar to a DMP is initialized from kinesthetic teaching and refined with PoWER. The behavior has been learned with a torque-controlled Barrett WAM arm with 7 DOF. The artificial pancake has a weight of 26 grams only, which makes its motion less predictable because it is susceptible to the influence of air flow. For refinement, a complex reward function has been designed that takes into account the trajectory of the pancake (flipping and catching), which is measured with a marker-based motion capture system. After 50 episodes, the first successful catch was recorded. A remarkable finding is that the learned behavior includes a useful aspect that has not directly been encoded in the reward function: it made a compliant vertical movement for catching the pancake that decreases the chance of the pancake bouncing off from the surface of the pan.

2.2.4.3. Restricting Information Loss

Peters et al. [PMA10] note that policy updates may result in information loss, which they reduce by enforcing the KL divergence of the previous and the next state-action distribution $p(\boldsymbol{x}, \boldsymbol{u})$ to be less than or equal a threshold ϵ, which is a hyperparameter of the proposed algorithm Relative Entropy Policy Search (REPS). REPS aims in each update at maximizing the expected reward of the new policy while bounding the KL divergence, that is,

$$\begin{aligned} \arg\max_{\boldsymbol{\theta}} \quad & \mathbb{E}_{\boldsymbol{x}\sim\rho^{\pi}(\boldsymbol{x}),\boldsymbol{u}\sim\pi_{\boldsymbol{\theta}}(\boldsymbol{u}|\boldsymbol{x})}\left[\pi_{\boldsymbol{\theta}}(\boldsymbol{u}|\boldsymbol{x}) r(\boldsymbol{x},\boldsymbol{u})\right] \\ \text{subject to} \quad & D_{\mathrm{KL}}(p_{\pi_{\boldsymbol{\theta}}}(\boldsymbol{x},\boldsymbol{u}) \parallel p_{\pi_{\boldsymbol{\theta}_{old}}}(\boldsymbol{x},\boldsymbol{u})) \leq \epsilon. \end{aligned}$$

The inequality constraint distinguishes REPS from NES and NAC that use an equality constraint for the KL divergence with an infinitesimal ϵ. The expectation and the KL divergence will be estimated from samples obtained with the old policy, which is a good enough approximation because each step is limited by ϵ.

To use a numerical optimizer, we also have to ensure that there is a stationary distribution over states (more precisely, state features) given the policy and that the policy and stationary distribution $\rho^{\pi}(\boldsymbol{x})$ are proper probability distributions. The resulting constraint optimization problem will be solved by the method of Lagrange multipliers with an optimizer such as L-BFGS-B. During optimization also a state value function will be approximated as $V_{\boldsymbol{v}}(\boldsymbol{x}) = \boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{v}$, where $\boldsymbol{v}$ are Lagrange multipliers and $\boldsymbol{\phi}$ computes arbitrary features of the state vector.

A main problem of episodic REPS in comparison to black-box optimizers such as CMA-ES or NES is its premature convergence because the entropy of the search distribution sometimes reduces too quickly [Abd+15]. In black-box optimization [HO01; Wie+14], according to Akimoto et al. [Aki+10], this is mitigated by using rank-based fitness shaping instead of fitness-proportionate updates of the search distribution (fitness is the same as return), that is, samples are ordered descending by their return and new monotonically decreasing return values are assigned to them. This also makes the optimizer invariant under rank-preserving transformations of the returns [Wie+14]. Note that all pure reinforcement learning algorithms that we discuss in this thesis use fitness-proportionate updates.

Nevertheless, REPS is an important algorithm that led to many follow-up works. Hierarchical REPS [DNP12a] is one extension of REPS that allows in situations, where several locally optimal behaviors exist, to learn all these optima simultaneously and is, thus, an improvement over VIP, which focuses only on one optimum.

A surrogate model has been introduced by Abdolmaleki et al. [Abd+15]. Their algorithm Model-based Relative Entropy Stochastic Search (MORE) learns a local quadratic surrogate model of the return. Bayesian dimensionality reduction is used to reduce the number of samples that is required to estimate the surrogate model in high-dimensional parameter spaces. The surrogate model is used to analytically limit the KL divergence. REPS could only compute a sample-based approximation, which requires more samples. Furthermore, MORE encourages higher entropy of the search distribution, which reduces the problem of premature convergence. MORE clearly outperforms PoWER, NES, and REPS in most presented experiments but CMA-ES is competitive in some experiments although it does not use a surrogate model.

2.2.4.4. Path Integrals

Policy Improvements with Path Integrals (PI^2) [TBS10b; TBS10a] is based on stochastic optimal control. PI^2 requires specification of an initial policy and a covariance matrix, which governs exploration in parameter space (a multiple of the identity matrix). Only the mean of the search distribution is adapted during learning and a new policy is sampled in each time step of an episode. PI^2 has also been used in the context of motion planning for manipulation [Kal+11a] to directly optimize end-effector trajectories.

Although PI^2 originates from a different field, parameter space exploration makes it similar to ES and its variants. This led to the development of a hybrid of PI^2 and CMA-ES [SS12] that automatically tunes the exploration noise of PI^2.

2.2.4.5. Parameterized Skills

Similarly to imitation learning, a way to generalize movement primitives in reinforcement learning is to parameterize them based on the task parameters, that is, we learn a mapping from task parameters $\boldsymbol{s} \in \mathcal{S}$ (usually $\mathcal{S} \subseteq \mathbb{R}^{n_s}$) to policy parameters $\boldsymbol{\theta}$. A more common name for $\boldsymbol{s}$ is context and we will use both terms interchangeably.

Silva et al. [SKB12] use PoWER to generate a training set of nearly optimal parameters $\{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots\}$ of low-level policies for several contexts $\{\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots\}$ for a regression algorithm. Thereupon, they infer a deterministic upper-level policy $\pi_{\boldsymbol{\omega}}(\boldsymbol{s}) = \boldsymbol{\theta}$, the so-called parameterized skill, which generalizes over the context space, with parameters $\boldsymbol{\omega}$. Silva et al. [SKB12] consider the problem of dart throwing and observe that this domain has the additional challenge of discontinuities in the mapping from task parameters to metaparameters of the policy. Such discontinuities occur if there are multiple solutions to a task with similar quality and if there are no constraints during reinforcement learning, but they are less likely when the examples have been generated by a human operator. To address this problem, Silva et al. [SKB12] use Isomap [TSL00], which extracts clusters that can be represented by a few parameters, and learn different upper-level policies for each cluster. Parameterized skills have been used to learn ball throwing for an iCub robot [da +14] on a target area of 90 cm × 90 cm with a policy that had 7 parameters to describe an overhand throw. In this work PI^2 was the underlying policy search algorithm. The threshold for successful policies was a distance of 6 cm to the target, which could be reached on average for 35 novel test targets with a training set of eight samples of successful throws that were learned for various other target positions.

Skill templates [Met+14] are an extension of parameterized skills that learn a stochastic upper-level policy $\pi_{\boldsymbol{\omega}}(\boldsymbol{\theta}|\boldsymbol{s})$, which includes an estimate of uncertainty that can be useful for subsequent adaptation of the policy through policy search in a new context.

An advantage of these approaches is that the upper level policy $\pi_{\boldsymbol{\omega}}(\boldsymbol{\theta}|\boldsymbol{s})$, which maps from contexts to parameters $\boldsymbol{\theta}$ of the low-level policy, can be an arbitrary function and, hence, complicated regression methods can be used for generalization, which can deal with discontinuities. On the other hand, a functional relationship between context and control-policy parameters $\boldsymbol{\theta}$ as in a deterministic policy might not always be adequate; for instance, there might be multiple optima in the space of $\boldsymbol{\theta}$ for a single context (for example, think of fore- and backhand strokes in tennis). In addition, the approach requires to learn close-to-optimal parameters for the control policy in a context to generate just one training example for regression. Consequently, all other experience collected while learning these close-to-optimal parameters is not used for generalization over the context space.

2.2.4.6. Contextual Policy Search

Several algorithms have been proposed that learn the upper-level policy $\pi_{\boldsymbol{\omega}}(\boldsymbol{\theta}|\boldsymbol{s})$ without separating learning in an RL and a regression part. These approaches allow transferring experience between different contexts even if the behavior in these contexts is suboptimal. Thus, these approaches can make use of all collected experience to be more

sample-efficient, that is, they can learn a close-to-optimal upper-level policy with fewer episodes. Furthermore, the upper-level policy π_ω can be stochastic, which means it can be implemented by a conditional probability distribution. This has the advantage that the agent's exploratory behavior is explicitly modeled and can be adapted by the learning algorithm on the upper level.

This category of algorithms is called contextual policy search, in which we seek to optimize

$$\arg\max_{\omega} \int_{\mathcal{S}} p(s) \int_{\mathbb{R}^n} \pi_\omega(\theta|s)\mathbb{E}\left[R(s,\theta)\right] d\theta ds,$$

where $s \in \mathcal{S}$ is a context, π_ω is a stochastic upper-level policy parameterized by ω that defines a distribution of policy parameters $\theta \in \mathbb{R}^n$ for a given context [DNP13]. The return R is extended to take into account the context. It is again stochastic because returns and environment dynamics are stochastic. During the learning process, we optimize ω, observe the current context s_i, and select $\theta_i \sim \pi_\omega(\theta|s)$.

RWR [PS07] and VIP [Neu11] directly support contextual policy search. They can learn a stochastic policy similar to the mapping from task parameters to policy metaparameters in the regression setting. Kober et al. [Kob+12] extend this to non-parametric policies in an approach that they called Cost-Regularized Kernel Regression (CrKR), which models the upper-level policy with a Gaussian process and assumes the outputs to be independent. CrKR has been used to learn throwing movements as well as table tennis.

Mülling et al. [MKP11; Mül+13] learn table tennis with a Barrett WAM arm. Particularly challenging in this task is the advanced perception and state estimation problem: behaviors have to take an estimate of the future ball trajectory into account when generating movements that determine where, when, and how the robot hits the ball. A vision system with an extended Kalman filter [Kal60] tracks the ball at 60 Hz and a simplified model that neglects spin predicts ball trajectories. 25 striking movements have been learned from kinesthetic teaching to form a library of movement primitives with a modified DMP version that allows to set a final velocity as a metaparameter. Desired position, velocity, and orientation of the racket are computed analytically for an estimated ball trajectory and a given target on the opponent's court, and are then given as metaparameters to the DMP. In addition, based on these task parameters, a gating network computes a weighted average of the repertoire of striking movements to form a mixture of movement primitives. The resulting behavior is refined with CrKR and a reward function that encourages minimization of the distance between the desired goal on the opponent's court and the actual point where the ball hits the table. In a final experiment, a human played against the robot, serving balls on an area of 0.8 m × 0.6 m. Up to nine balls were returned in a row by the robot. Initially the robot was able to return 74.4 % of the balls and after playing one hour the robot was able to return 88 %.

Extending standard policy search algorithms to the contextual setting is often straightforward. For instance, Contextual Relative Entropy Policy Search (C-REPS) [Kup+13] is an extension of REPS. One of the key advantages of C-REPS over similar methods is that it takes into account that different contexts might have different reward distribu-

tions and computes a baseline to normalize the reward of each context. This baseline approximates a context value function.

Because episodic REPS can be used for black-box optimization, C-REPS is a template for the extension of other methods. CMA-ES [HO01] has been extended to Contextual Covariance Matrix Adaptation Evolution Strategies (C-CMA-ES) [Abd+17a], MORE [Abd+15] to Contextual Model-based Relative Entropy Stochastic Search (C-MORE) [Tan+17], and BO [BCd10] to Bayesian Optimization for Contextual Policy Search (BO-CPS) [MFH15]. There is also a hybrid of C-REPS and CMA-ES [Abd+19]. Since contextual policy search is a main focus of this thesis we will give a more detailed overview of contextual policy search algorithms in Section 2.3.

2.2.5. Deep Reinforcement Learning with Value Functions

Deep reinforcement learning is a new field that makes use of the results from deep learning. Although there are only a few applications of deep reinforcement learning in robotics, results of these methods are interesting for behaviors that involve difficult perception problems. Specifically, it allows to learn complex end-to-end mappings from sensor measurements to actuator commands or expected returns. A recent survey of deep reinforcement learning has been published by Arulkumaran et al. [Aru+17] and a survey of deep learning for robotic perception and control by Tai and Liu [TL16].

Deep RL with value functions specifically approximates either the state value function $V(\boldsymbol{x})$, the state-action value function $Q(\boldsymbol{x}, \boldsymbol{u})$, or the advantage function $A(\boldsymbol{x}, \boldsymbol{u})$ with a neural network. Neural networks allow us to handle complex state spaces with high dimensionality or continuous values efficiently. Handling continuous action spaces, however, is more difficult. An overview of algorithms for RL with value functions that we discussed previously or will discuss here is shown in Figure 2.8.

The first success of reinforcement learning with neural networks was an agent that played backgammon with TD(λ) at the level of good human players [Tes95]. Convergence with a neural network as a function approximator for the value function [Tes92] is not guaranteed though.

For a long time this was the only prominent example that showed how neural networks can be used for reinforcement learning. Reinforcement learning is difficult with a complex function approximator such as a neural network, as they suffer from catastrophic forgetting [MC89; Rat90], that is, they prefer memorizing recent training samples over older training samples. This is particularly critical for supervised online learning but also in reinforcement learning, where the distribution of visited state-action pairs changes continually while the policy improves. This motivated Riedmiller [Rie05] and Riedmiller et al. [RMD07] to use experience replay [Lin92] to reduce the problem of catastrophic forgetting in Q-learning with a neural network. The algorithm Neural Fitted Q Iteration (NFQ) accumulates a set $\mathcal{D}$ of experiences $(\boldsymbol{x}_i, \boldsymbol{u}_i, r_i, \boldsymbol{x}'_i)$. Then NFQ constructs a loss

$$L(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} \left[r_i + \gamma \arg\max_{\boldsymbol{u}'} \hat{Q}(\boldsymbol{x}'_i, \boldsymbol{u}') - \hat{Q}(\boldsymbol{x}_i, \boldsymbol{u}_i) \right]^2$$

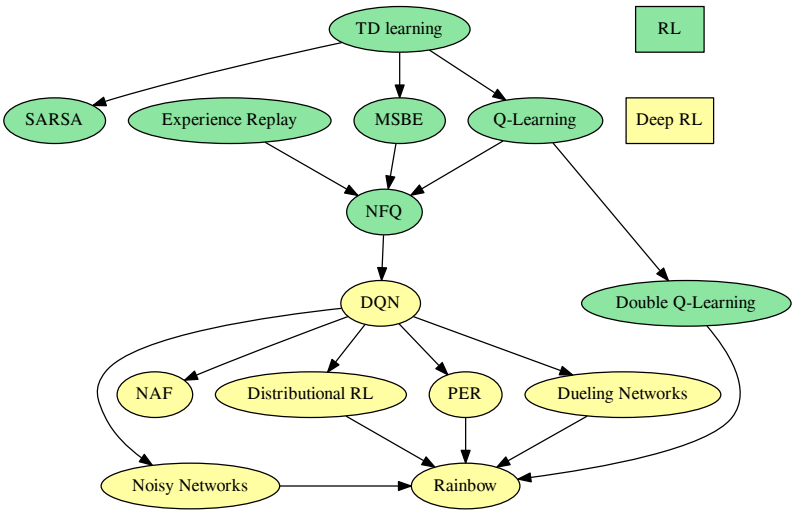


Figure 2.8.: Overview of reinforcement learning algorithms with value functions. Each ellipse represents an idea or algorithm that has been published. We can distinguish between standard RL and deep RL algorithms. Arrows indicate connections between ideas (a strong influence of one publication on the other). Ideas above others in a chain have been developed earlier.

with a previous estimate of the state-action value function $\hat{Q}$. This is the MSBE for Q-learning. Then the loss is used to train the state-action value network in a variant of batch gradient descent until convergence. NFQ stores all experiences. For difficult problems that require a lot of experience and large neural networks neither storing all samples nor batch training until convergence are feasible.

New concepts had to be developed to enable Q-learning with deeper architectures and to solve more complex problems such as video games of which we only observe visual inputs. Learning the state-action value function with a neural network is a fragile process in this domain. We sample from a high-dimensional state space, measure possibly stochastic rewards, approximate the expected return with a complex function approximator $\hat{Q}^*_{\boldsymbol{\theta}}$ with a large amount of parameters $\boldsymbol{\theta}$ on a small subset of all possible state-action pairs using a stochastic optimization procedure while the target that we approximate with the optimization depends on the current estimate $\hat{Q}^*_{\boldsymbol{\theta}}$ which changes during this optimization. Mnih et al. [Mni+15] mitigate the latter problem by introducing a target network $\hat{Q}^*_{\boldsymbol{\theta}^-}$ of which the parameters $\boldsymbol{\theta}^-$ are less frequently updated. The target network is used to compute the temporal difference errors for the update of $\hat{Q}^*_{\boldsymbol{\theta}}$. Then a variant of stochastic gradient descent uses small batches of experience from a fixed-size *replay buffer* for this update. The proposed algorithm is called Deep Q Networks (DQN) and is able to reach and surpass human level of control on many problems from the Arcade

Learning Environment (ALE), a frequently used benchmark environment that provides an interface to hundreds of Atari 2600 games [Bel+13].

Various extensions that increase the performance of the final policy derived from DQN and the sample efficiency have been proposed. Double Q-learning can be combined with DQN [HGS16]. Prioritized experience replay (PER) [Sch+16a] uses the replay buffer more intelligently. A dueling network architecture separates the state-action value function into the state value function and an advantage function to improve generalization across actions [Wan+16c]. Noisy networks represent a learnable exploration strategy which improves standard exploration [For+18]. Distributional reinforcement learning learns a value distribution instead of a value function [BDM17]. Rainbow combines all of these improvements [Hes+18]. The reported learning curves of Rainbow for Atari games show the performance for up to 200 million steps. While the observation space for video games is high-dimensional and complex, the action space contains only a few discrete actions.

Most algorithms that are derived from Q-learning require discrete action spaces because we have to be able to find the best action for a given state to update the state-action value function and to determine the policy. Gu et al. [Gu+16] extend Q-learning to continuous actions. They separate the state-action value function into state value function and advantage function. The advantage function is enforced to be quadratic with respect to the state vector by predicting parameters of this quadratic function with a neural network. We can compute the optimum of the quadratic function analytically. This algorithm is called Normalized Advantage Function (NAF). Restricting the advantage function to be quadratic is a strong inductive bias. Nevertheless, this algorithm has been used successfully by Gu et al. [Gu+17] to learn how to open a door with a robot arm.

We can use deep reinforcement learning with value functions and large amount of experience to learn behaviors for complex state spaces. With a strong inductive bias we can extend it to handle continuous action spaces. A more promising approach to handle complex, continuous action spaces, however, is to represent a policy by a neural network and use policy gradients to train them, which can be supported by value functions.

2.2.6. Deep Reinforcement Learning with Policy Gradients

Deep policy gradient algorithms represent the policy $\pi_{\boldsymbol{\theta}}$ by a neural network with weights $\boldsymbol{\theta}$. An overview of the algorithms that we will discuss can be found in Figure 2.9.

We can distinguish between deterministic and stochastic policies. A deterministic policy $\boldsymbol{u} = \pi_{\boldsymbol{\theta}}(\boldsymbol{x})$ can be represented directly by a neural network. A stochastic policy $\pi_{\boldsymbol{\theta}}(\boldsymbol{u}|\boldsymbol{x})$ would be a probability density function, that is, $\int_{\mathcal{U}} \pi_{\boldsymbol{\theta}}(\boldsymbol{u}|\boldsymbol{x}) d\boldsymbol{u} = 1$ and $\pi_{\boldsymbol{\theta}}(\boldsymbol{u}|\boldsymbol{x}) > 0$. We can guarantee that a neural network fulfills these conditions by restricting $\pi_{\boldsymbol{\theta}}$ to a specific parametric distribution such as a Gaussian distribution. In this case $\pi(\boldsymbol{u}|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{u}|\mu(\boldsymbol{x}), \sigma(\boldsymbol{x}))$ with the distribution's parameters $\mu(\boldsymbol{x}), \sigma(\boldsymbol{x})$ being the outputs of the neural network. We can sample from this distribution with $\boldsymbol{u} = \mu(\boldsymbol{x}) + \sigma(\boldsymbol{x}) \circ \boldsymbol{\epsilon}$, where $\boldsymbol{\epsilon}$ is sampled from a standard normal distribution.

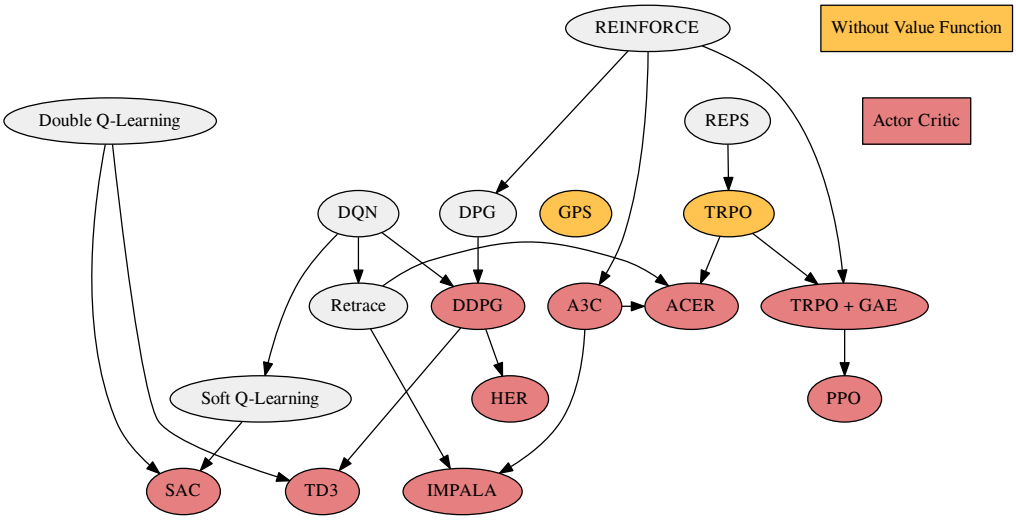


Figure 2.9.: Overview of policy gradient algorithms. Each ellipse represents an idea or algorithm that has been published. We can distinguish between algorithms without value function approximation and actor critic algorithms. Arrows indicate connections between ideas (a strong influence of one publication on the other). Ideas above others in a chain have been developed earlier.

2.2.6.1. Policy Gradient Theorems

Policy gradient algorithms directly maximize the expected return $\mathbb{E}[R(\boldsymbol{\theta})]$ with gradient-based optimization, for instance, gradient ascent. The expectation is computed over the states $\boldsymbol{x} \in \mathcal{X}$ that are distributed according to $\boldsymbol{x} \sim \rho^{\pi}$ and the actions $\boldsymbol{u} \in \mathcal{U}$ that are selected by the policy $\boldsymbol{u} \sim \pi_{\boldsymbol{\theta}}$. That is, we can write

$$\mathbb{E}[R(\boldsymbol{\theta})] = \int_{\mathcal{X}} \rho^{\pi}(\boldsymbol{x}) \int_{\mathcal{U}} \pi_{\boldsymbol{\theta}}(\boldsymbol{u}|\boldsymbol{x}) r(\boldsymbol{x}, \boldsymbol{u}) d\boldsymbol{u}\, d\boldsymbol{x}$$

with the reward function $r(\boldsymbol{x}, \boldsymbol{u})$. The reward function and the state distribution are assumed to be unknown and, thus, can only be sampled. The policy gradient theorem eliminates this problem. There are different versions of it for stochastic and deterministic policies as well as for on-policy and off-policy algorithms.

Sutton et al. [Sut+00] derive the policy gradient theorem for stochastic policies and on-policy algorithms, which (combined with the log-derivative trick [Wil92]) is

$$\nabla_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{x} \sim \rho^{\pi}, \boldsymbol{u} \sim \pi_{\boldsymbol{\theta}}}[R(\boldsymbol{\theta})] = \mathbb{E}_{\boldsymbol{x} \sim \rho^{\pi}, \boldsymbol{u} \sim \pi_{\boldsymbol{\theta}}}[\nabla_{\boldsymbol{\theta}} \ln \pi_{\boldsymbol{\theta}}(\boldsymbol{u}|\boldsymbol{x}) Q^{\pi}(\boldsymbol{x}, \boldsymbol{u})] .$$

Hence, the gradient does not depend on the derivative of the state visitation distribution and the expectation can be estimated with sampled state-action pairs and an estimate of

the state-action value function Q^π. An unbiased estimate of Q^π is the measured return R. We will omit the subscripts of the expectation in the following.

Similarly, Degris et al. [DWS12] derived the policy gradient theorem for stochastic policies and off-policy algorithms, which states

$$\nabla_{\boldsymbol{\theta}} \mathbb{E}\left[R(\boldsymbol{\theta})\right] \approx \mathbb{E}\left[\frac{\pi_{\boldsymbol{\theta}}(\boldsymbol{u}|\boldsymbol{x})}{\beta(\boldsymbol{u}|\boldsymbol{x})} \nabla_{\boldsymbol{\theta}} \ln \pi_{\boldsymbol{\theta}}(\boldsymbol{u}|\boldsymbol{x}) Q^\pi(\boldsymbol{x}, \boldsymbol{u})\right],$$

where β is the behavior policy for exploration and π is the target policy. $\frac{\pi_{\boldsymbol{\theta}}(\boldsymbol{u}|\boldsymbol{x})}{\beta(\boldsymbol{u}|\boldsymbol{x})}$ is the ratio for importance sampling, since states and actions are distributed according to β but we compute the gradient with respect to the parameters of π. Note that this is an approximation even without sampling state-action pairs, which has to be done, too.

Silver et al. [Sil+14] introduce the policy gradient theorem for deterministic policies and off-policy algorithms, which states

$$\nabla_{\boldsymbol{\theta}} \mathbb{E}\left[R(\boldsymbol{\theta})\right] \approx \mathbb{E}\left[\nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(\boldsymbol{x}) \nabla_{\pi_{\boldsymbol{\theta}}(\boldsymbol{x})} Q^\pi(\boldsymbol{x}, \pi_{\boldsymbol{\theta}}(\boldsymbol{x}))\right],$$

where states are distributed according to some behavior policy β.

2.2.6.2. Policy Gradients with Advantage Function

Williams [Wil92] introduces REINFORCE (abbreviation of Reward Increment = Non-negative Factor × Offset Reinforcement × Characteristic Eligibility), a type of policy gradient algorithm that uses the Monte Carlo estimate of Q^π. In addition, Williams [Wil92] introduces a baseline in the policy gradient and proves that the baseline does not add a bias to the estimate but can be used to reduce the variance, hence, the stochastic policy gradient for on-policy algorithms is often used in the form

$$\nabla_{\boldsymbol{\theta}} \mathbb{E}\left[R(\boldsymbol{\theta})\right] = \mathbb{E}\left[\nabla_{\boldsymbol{\theta}} \ln \pi_{\boldsymbol{\theta}}(\boldsymbol{u}|\boldsymbol{x}) \left(Q^\pi(\boldsymbol{x}, \boldsymbol{u}) - b(\boldsymbol{x})\right)\right],$$

with a baseline $b(\boldsymbol{x})$, which is often the state value function so that we can write

$$\nabla_{\boldsymbol{\theta}} \mathbb{E}\left[R(\boldsymbol{\theta})\right] = \mathbb{E}\left[\nabla_{\boldsymbol{\theta}} \ln \pi_{\boldsymbol{\theta}}(\boldsymbol{u}|\boldsymbol{x}) A^\pi(\boldsymbol{x}, \boldsymbol{u})\right],$$

by definition of the advantage function $A^\pi(\boldsymbol{x}, \boldsymbol{u}) = Q^\pi(\boldsymbol{x}, \boldsymbol{u}) - V^\pi(\boldsymbol{x})$.

2.2.6.3. Deterministic Policy Gradients

Deterministic Policy Gradients (DPG) uses the deterministic policy gradient theorem. It is an off-policy actor-critic algorithm. Deep Deterministic Policy Gradients (DDPG) extends DPG to deep neural networks and uses DQN to estimate Q^π of the deterministic target policy π [Lil+16]. DDPG is outlined in Algorithm 1. Note that optimization is performed with a variant of stochastic gradient descent and gradients are computed by automatic differentiation. DDPG is the only algorithm that we show in detail in this section to get an impression of how a policy gradient algorithm works.

Algorithm 1 Deep Deterministic Policy Gradients

Randomly initialize critic network $Q(\boldsymbol{x}, \boldsymbol{u}|\theta^Q)$ and actor $\pi(\boldsymbol{x}|\theta^\pi)$
Initialize target networks Q' and π' with weights $\theta^{Q'}, \theta^{\pi'}$
Initialize replay buffer D, distribution $\mathcal{N}$ for action exploration
while not converged **do**
    while episode not terminated **do**
        Observe state $\boldsymbol{x}$ and execute action $\boldsymbol{u} = \pi(\boldsymbol{x}_t|\theta^\pi) + \mathcal{N}$
        Observe reward r and new state $\boldsymbol{x}'$
        Store transition $(\boldsymbol{x}, \boldsymbol{u}, r, \boldsymbol{x}')$ in D
        Sample batch of N transitions $(\boldsymbol{x}_i, \boldsymbol{u}_i, r_i, \boldsymbol{x}_{i+1})$ from D
        Set $y_i = r_i + \gamma Q'(\boldsymbol{x}_{i+1}, \pi'(\boldsymbol{x}_{i+1}|\theta^{\pi'})|\theta^{Q'})$
        Update critic with $\arg\min_{\theta^Q} \frac{1}{N} \sum_i (y_i - Q(\boldsymbol{x}_i, \boldsymbol{u}_i|\theta^Q))^2$
        Update actor with $\arg\max_{\theta^\pi} \frac{1}{N} \sum_i Q(\boldsymbol{x}_i, \pi(\boldsymbol{x}_i|\theta^\pi))$
        $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$, $\theta^{\pi'} \leftarrow \tau\theta^\pi + (1-\tau)\theta^{\pi'}$
    end while
end while

Lillicrap et al. [Lil+16] demonstrate that DDPG handles high-dimensional state or observation spaces such as images well, which DPG with a linear function approximator could not. DDPG has been extended to Twin Delayed Deep DPG (TD3), which adds tricks such as double Q-learning [FHM18] to improve the performance and sample efficiency; however, for relatively simple walking problems from OpenAI Gym [Bro+16] the required steps to reach a good performance is in the order of 100,000.

Andrychowicz et al. [And+17] develop Hindsight Experience Replay (HER) based on DDPG. They address problems with sparse binary rewards that are only meaningful if a goal was reached. They investigated robotic manipulation tasks such as pushing an object to a given target position or picking up an object and placing it at a target position. In these problems the reward is parameterized by the goal, which can be exploited by replaying experience as if it was collected during an episode with a different goal. This setting is similar to contextual policy search. Andrychowicz et al. [And+17] consider the problem of pushing an object to a goal. Object and goal can be placed at arbitrary positions on a table in front of the robot. The robot has a budget of 50 steps to reach the goal so that it receives a reward. A goal is reached if the object is within 7 cm of the goal. HER needs about 70 updates to reach around 90 % success rate in this task while each update requires 800 episodes which results in a total number of 56,000 episodes. HER is considered to be a state of the art in reinforcement learning for simulated robots.

2.2.6.4. Trust Region Methods

Even small steps in the parameter space of a policy can have large impact on outputs of the policy. Therefore, policy gradient algorithms can be unstable, as performance might even degrade during training. Trust Region Policy Optimization (TRPO) mitigates this problem by taking the largest step that improves the policy on a batch of samples

[Sch+15c]. Although TRPO is an on-policy algorithm, it uses an objective for the policy that includes importance sampling to maximize the surrogate reward

$$\mathbb{E}\left[\frac{\pi_{\boldsymbol{\theta}}(\boldsymbol{u}|\boldsymbol{x})}{\pi_{\boldsymbol{\theta}_{old}}(\boldsymbol{u}|\boldsymbol{x})} Q^{\pi_{\boldsymbol{\theta}_{old}}}(\boldsymbol{x},\boldsymbol{u})\right] = \mathbb{E}\left[Q^{\pi_{\boldsymbol{\theta}}}\right]$$

with a Monte Carlo estimate of the state-action value function and furthermore restricts the update step such that the expected Kullback-Leibler divergence D_{KL} between the old and the new policy is below a threshold ϵ, that is,

$$\mathbb{E}\left[D_{\mathrm{KL}}(\pi_{\theta_{old}}(\cdot|\boldsymbol{x}) \parallel \pi_{\theta}(\cdot|\boldsymbol{x}))\right] \leq \epsilon.$$

This notation means that the KL divergence is computed over the action distribution of a fixed state, while the expected value is computed over the distribution of the state. This procedure is not straightforward and involves approximations such as the linearization of the objective and quadratic approximation of the constraint and a line search to ensure that performance increases monotonously. Although the basic idea is similar to REPS [PMA10], TRPO bounds the conditional distribution $p(\boldsymbol{u}|\boldsymbol{x})$, whereas REPS bounds $p(\boldsymbol{x},\boldsymbol{u})$ and performs a more costly nonlinear optimization in every step.

Duan et al. [Dua+16] made an empirical comparison of several reinforcement learning algorithms for neural networks in continuous control problems. These algorithms include REINFORCE [Wil92], a variant of the natural policy gradient [PS08b], RWR [PS07], REPS [PMA10], TRPO [Sch+15c], CEM [Rub99; MRG03], CMA-ES [HO01], and DDPG [Lil+16]. Note that only neural networks as a policy representation have been used. The main focus of the evaluation is on walking problems with varying complexity up to a full humanoid mechanism. They found that TRPO mostly outperforms the other algorithms. This finding is remarkable because TRPO performs updates after a couple of episodes and does not learn a value function while DDPG continuously learns during each episode and because the original paper of Lillicrap et al. [Lil+16] on DDPG does not include a comparison to TRPO for exactly those reasons. This finding has been confirmed by Ha et al. [HKY18] who learned crawling behaviors for simple mechanisms and observe that the learning progress of TRPO is more stable. Furthermore, Mahmood et al. [Mah+18] performed extensive evaluation on various robotic systems and show that TRPO consistently outperforms DDPG.

Schulman et al. [Sch+16c] extend TRPO to an actor-critic algorithm and use an estimate of the advantage function instead of the state-action value function in the objective for the policy. They introduce Generalized Advantage Estimation (GAE), a method to trade off bias and variance in estimation of the advantage function with a single parameter $\lambda \in [0,1]$. Let $\hat{V}$ be an approximate value function, the temporal difference error is defined as

$$\delta_t^{\hat{V}} = r_t + \gamma \hat{V}(\boldsymbol{x}_{t+1}) - \hat{V}(\boldsymbol{x}_t).$$

If V is the real value function, then

$$\begin{aligned}\mathbb{E}\left[\delta_t^V\right] &= \mathbb{E}\left[r_t + \gamma V(\boldsymbol{x}_{t+1}) - V(\boldsymbol{x}_t)\right] \\ &= \mathbb{E}\left[Q(\boldsymbol{x}_t,\boldsymbol{u}_t) - V(\boldsymbol{x}_t)\right] = A(\boldsymbol{x}_t,\boldsymbol{u}_t),\end{aligned}$$

hence, δ_t^V is an estimator of the advantage. We can take the discounted sum of n temporal difference errors

$$\hat{A}_t^{(n)} = \sum_{l=0}^{n-1} \gamma^l \delta_{t+l}^{\hat{V}} = -\hat{V}(\boldsymbol{x}_t) + r_t + \gamma r_{t+1} + \ldots + \gamma^n \hat{V}(\boldsymbol{x}_{t+n}),$$

which is the n-step truncated Monte Carlo estimate of the advantage and is biased when $\hat{V}(\boldsymbol{x}_{t+n})$ is biased. The bias becomes smaller for $n \to \infty$. $-\hat{V}(\boldsymbol{x}_t)$ can be ignored because it is the same for every action. For $n \to \infty$,

$$\lim_{n\to\infty} \hat{A}_t^{(n)} = \lim_{n\to\infty} \sum_{l=0}^{n} \gamma^l \delta_{t+l}^{\hat{V}} = \lim_{n\to\infty} \left[\sum_{l=0}^{n} \gamma^l r_{t+l} \right] - \hat{V}(\boldsymbol{x}_t) = R_t - \hat{V}(\boldsymbol{x}_t)$$

reduces to the Monte Carlo estimate. $\hat{A}_t^{(1)} = \delta_t^{\hat{V}}$ reduces to the temporal difference error. The construction of the generalized advantage estimate is similar to TD(λ):

$$\hat{A}_t^{GAE(\gamma,\lambda)} = (1-\lambda)\left(\hat{A}_t^{(1)} + \lambda \hat{A}_t^{(2)} + \lambda^2 \hat{A}_t^{(3)} + \ldots\right) = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}^{\hat{V}},$$

with $\hat{A}_t^{GAE(\gamma,0)} = \delta_t^{\hat{V}}$ (low variance, high bias) and $\hat{A}_t^{GAE(\gamma,1)} = \left[\sum_{l=0}^{\infty} \gamma^l r_{t+l}\right] - \hat{V}(\boldsymbol{x}_t)$ (high variance, no bias) as edge cases. λ controls the compromise between bias and variance.

Hwangbo et al. [Hwa+19] present a real-world application of TRPO [Sch+15c] with GAE [Sch+16c]. They learn walking for a quadrupedal robot from simulation and focus on making the simulation as realistic as possible by integrating a learned model of the robot's actuators. The simulation runs much faster than real time and training of a behavior takes 4 hours of wall-clock time or 9 days of simulated time. The behavior that was deployed on the real robot followed commands more accurately, was more energy-efficient, and was faster than the previous, engineered walking controllers that often took months of development effort. Nevertheless, there were some difficulties that had to be solved to generate realistic walking behaviors. A walking behavior for a real robot should use as low torques as possible. Encoding both objectives—moving forward and moving with low torques—requires balancing between them. If penalties for torques are too high, the robot will stand still. If they are too low, they will not result in walking behaviors that can be executed on a real robot. This makes it complicated to define a single combined objective that can be used to learn the behavior. Hence, Hwangbo et al. [Hwa+19] use a curriculum of increasing difficulty. They first learn to solve the main objective and then continuously increase the weights of constraints such as energy efficiency.

Proximal Policy Optimization (PPO) [Sch+17b] simplifies the optimization problem in comparison to TRPO by replacing the optimization constraint with a clipping function, which is much easier to implement while it matches the performance of TRPO. PPO uses

$$\mathbb{E}\left[\min\left(\rho_\theta(\boldsymbol{x},\boldsymbol{u})\hat{A}(\boldsymbol{x},\boldsymbol{u}), \underbrace{\text{clip}(\rho_\theta(\boldsymbol{x},\boldsymbol{u}), 1-\epsilon, 1+\epsilon)}_{\text{clipped ratio}} \hat{A}(\boldsymbol{x},\boldsymbol{u})\right)\right]$$

as the objective for the policy, where $\rho_\theta(\boldsymbol{x}, \boldsymbol{u}) = \frac{\pi_\theta(\boldsymbol{x},\boldsymbol{u})}{\pi_{\theta_{old}}(\boldsymbol{x},\boldsymbol{u})}$ and ϵ is a clipping parameter that is often set to values between 0.1 and 0.3. PPO also can be combined with GAE [Sch+16c].

2.2.6.5. Distributed Policy Gradients

A major focus of deep reinforcement learning research has been distributed computation, which is straightforward for environments that can be simulated fully. PPO has been used to learn in-hand manipulation with a complex human-like robotic hand [Ope+20], basic movements to solve Rubik's cube with the same hand [Ope+19b], play the video game Dota 2 [Ope+19a], learn complex walking behaviors from simple rewards in complex environments [Hee+17], and to compete in a two-team hide-and-seek game in which we can observe emergent tool use [Bak+19]. While all of these applications represent extraordinary achievements in terms of what can be learned, they all required massive amount of data, engineering effort, and computational resources. This is one of the main directions of research in deep reinforcement learning at the moment.

OpenAI et al. [Ope+20] learn in-hand manipulation behaviors for a complex robotic hand with 20 actuated degrees of freedom, that is, they learn how to turn a cube from one orientation to another specified orientation. For this application three cameras were placed around the hand to observe the orientation of the cube. Camera images were given directly as input to the policy network. In addition, a motion capture system was required to determine the state of the robotic hand. Still 100 years of simulated experience were required to train policies that are robust enough to be executed on a real hand. So much simulated experience was required because of an approach that is called domain randomization, which was used to mitigate the simulation-reality gap: many different variations of the simulation with different physics parameters and visual appearance were used to let the reality seem to be just another variation of the simulation. Although in the best trial the robot was able to turn the cube 50 times consecutively, the median number of consecutive successful rotations was 13 in 10 trials with a maximum number of 50 rotations. In their follow-up work OpenAI et al. [Ope+19b] used more than 13 thousand years of simulated experience to learn manipulation skills for Rubik's cube. While the achieved results are unprecedented and it is important to explore the capabilities of these algorithms, we should not believe that these skills can be learned on a real robot.

Another extreme example that illustrates the enormous potential of neural networks to represent complex behaviors but also requires huge amount of data and computational resources is the work of OpenAI et al. [Ope+19a] who learn to play the video game Dota 2 and win against professional human players. For this application 45,000 years of game experience have been used. To give an impression of how much computational resources are required for these kind of applications OpenAI [Ope18] write that they used 128,000 CPU cores to collect experience and 256 P100 GPUs to run and train neural networks.

A deep RL algorithm that was explicitly designed for distributed training is Asynchronous Advantage Actor Critic (A3C) [Mni+16]. It computes the n-step truncated

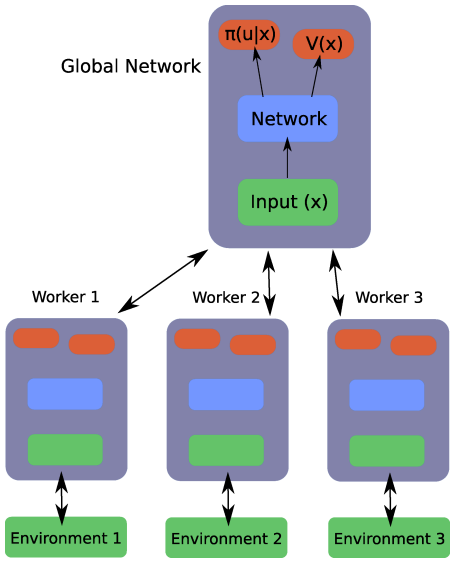


Figure 2.10.: Distributed architecture of A3C. Based on diagram from Juliani [Jul16].

Monte Carlo estimate of the advantage function with an approximation of the state value function from a neural network to compute the stochastic on-policy gradient. A3C adds an entropy bonus to the objective of the policy to avoid premature convergence. The target value for the value function is the n-step truncated Monte Carlo estimate of the state value. The networks for the state value function and the policy share a common core. Figure 2.10 shows the distributed architecture of A3C with a central server that updates network parameters with the gradients that it receives from workers that collect experiences in instances of the environment. The parameter copies of the workers are regularly updated with parameters from the global network on the central server. A3C has been evaluated in various Atari 2600 games [Bel+13] and has been shown to clearly outperform DQN with respect to wall-clock training time and final performance.

Wang et al. [Wan+16a] improve the sample efficiency of A3C with off-policy samples from old policies for the policy gradient estimate in the algorithm Actor Critic with Experience Replay (ACER). ACER builds on the Retrace estimator [Mun+16] for the advantage function. In addition, the update step is limited by a trust region approach similar to TRPO [Sch+15c].

Espeholt et al. [Esp+18] take the distribution one step further and not only distribute workers that interact with the environment but they also distribute gradient computations in the algorithm Importance Weighted Actor-Learner Architecture (IMPALA). Workers only send observations $(\boldsymbol{x}_t, \boldsymbol{u}_t, r_t, \boldsymbol{x}_{t+1})$ that they collected during interaction, which avoids the communication overhead of gradients from large neural networks as illustrated in Figure 2.11. Now observations might correspond to old policies, hence, they are off-policy samples. Therefore value function estimates are computed based on V-trace and additionally the truncated importance sampling weights from V-trace are

used in the policy gradient. Similarly to A3C an estimate of the advantage function is used so that we estimate the policy gradient

$$\mathbb{E}\left[\rho_t \nabla_{\boldsymbol{\theta}} \ln \pi_{\boldsymbol{\theta}}(\boldsymbol{u}_t|\boldsymbol{x}_t) \quad \overbrace{\hat{A}(\boldsymbol{x}_t, \boldsymbol{u}_t)}^{r_t+\gamma v_{t+1}-\hat{V}(\boldsymbol{x}_t)}\right],$$

where v_{t+1} is estimated by V-trace, ρ_t is the truncated importance sampling weight from V-trace, and a neural network represents $\hat{V}$. IMPALA has been evaluated initially in multi-task scenarios such as learning multiple Atari games from ALE. It was on average found to perform comparable to A3C instances with shallow neural networks that have been trained on individual tasks only. When IMPALA is trained specifically for one task, it clearly outperforms A3C [Esp+18]. After this first success IMPALA has been used in combination with various extensions and complex network architectures in AlphaStar, which was able to beat professional human players at the video game StarCraft II [Vin+19], and the FTW agent, which was able to beat strong humans players in capture the flag in the video game Quake 3 [Jad+19]. Both games were played with images as inputs to the policy, which can be handled well by neural networks. Both applications require a huge amount of training data and computational resources though. Vinyals et al. [Vin+19] report that AlphaStar instantiates 12 agents that compete against each other during training. Each agent runs 16,000 concurrent matches on an equivalent of roughly 4,200 CPU cores and makes predictions on Tensor Processing Units (TPUs), which are specialized hardware components for neural networks. Furthermore, there is a central 128-core TPU learner responsible for each agent.

State of the art in the ALE, however, at the time of writing this thesis is achieved by Agent57 [Bad+20], a value-function based approach that is optimized for the domain. It uses curiosity-driven exploration and reaches human-level performance on all 57 investigated Atari games and the median score over all 57 games is 1933.49 % of an average human's score. 78 billion steps are required during training to surpass human-level performance in the most difficult game. Assuming a frame rate of 50 frames per second this would result in almost 50 years of experience.

2.2.6.6. Soft Actor Critic

A novel approach to policy gradient algorithms comes from maximum entropy reinforcement learning, which encourages policies to exhibit a diverse distribution and maximize the return, that is, we seek to maximize

$$\mathbb{E}\left[R + \alpha \mathcal{H}(\pi(\cdot|\boldsymbol{x}))\right],$$

with the temperature α as a hyperparameter that controls the compromise between maximizing the return R and the entropy $\mathcal{H}$ of the policy. Haarnoja et al. [Haa+18a] investigate soft Q-learning for real-world robot manipulation tasks. The (soft) state-action value function in soft Q-learning approximates the expectation from the definition

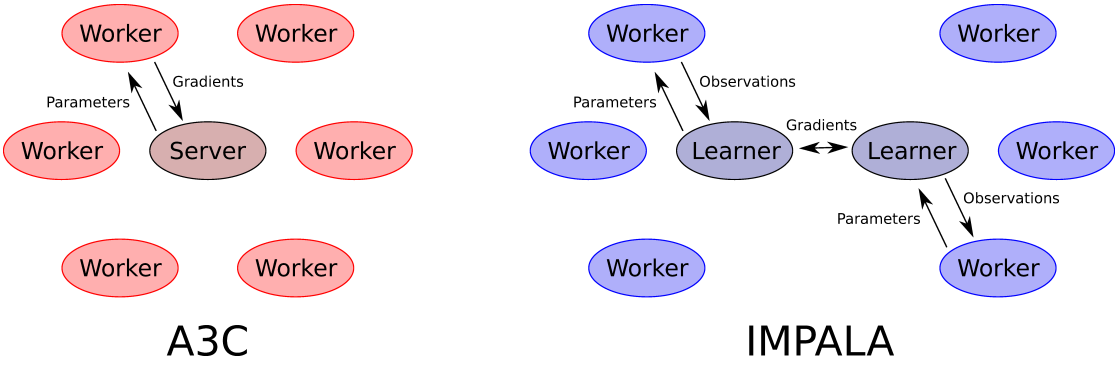


Figure 2.11.: Comparison of distributed architectures of A3C and IMPALA. Based on diagram of Soyer et al. [SPE18].

above. Haarnoja et al. [Haa+18b] extend this to Soft Actor-Critic (SAC), an off-policy actor-critic algorithm based on maximum entropy reinforcement learning. The objective of the policy update looks completely different from other policy gradient algorithms as the goal is to minimize the expected KL divergence between the new policy and the softmax of the soft state-action value function, that is,

$$\arg\min_{\boldsymbol{\theta}} \mathbb{E}\left[D_{\mathrm{KL}}\left(\pi_{\boldsymbol{\theta}}(\cdot|\boldsymbol{x}) \,\middle\|\, \frac{\exp(Q(\boldsymbol{x},\cdot))}{Z(\boldsymbol{x})}\right)\right],$$

where Q is approximated by a neural network through soft Q-learning and Z can be ignored, as it cancels out in the policy gradient and is never explicitly evaluated. Instead of the score function estimator, which is used in the stochastic policy gradient theorem, SAC uses the reparameterization trick (also known as pathwise gradient estimator [Moh+19]) to directly compute the gradient of the objective with respect to policy parameters. The policy $\pi_{\boldsymbol{\theta}}$ is represented by a neural network that takes a random noise vector $\boldsymbol{\epsilon}$ and a state $\boldsymbol{x}$ to predict an action. The noise can be sampled, for instance, from an isotropic Gaussian. When we estimate the policy gradient, we consider samples of $\boldsymbol{\epsilon}$ to be constants. With this trick we can represent and learn complicated policies such as multi-modal distributions. Later versions of the algorithm [Haa+18c; Haa+19] change the policy optimization objective to a constraint optimization problem with a lower limit for the entropy, which results in an objective that will be solved with dual gradient ascent. In terms of sample efficiency, SAC outperforms methods such as DDPG, PPO, TD3 in Open AI gym walking problems. SAC has been used to learn walking with a simple, real quadrupedal robot Haarnoja et al. [Haa+18c; Haa+19] within 160,000 environment steps, which amount to about 2 hours. Ha et al. [Ha+20] show that this specific application can be fully automated with a system that recovers the robot from failure states.

2.2.6.7. Guided Policy Search

Guided Policy Search (GPS) [LK13] is not a typical policy gradient algorithm. It is model-based and turns policy optimization into a supervised learning problem. GPS alternates between fitting a local linear model with Gaussian noise of the dynamics, trajectory optimization with iterative linear-quadratic-Gaussian regulators (iLQGs) [TL05; TET12a] with the model of the dynamics, and policy learning. A neural network is trained to mimic the optimized trajectories in the policy learning step by minimizing the KL divergence between guiding policies, which are obtained through trajectory optimization, and the neural network policy.

Levine et al. [Lev+16] show that GPS can learn behaviors that use raw camera images to compute corresponding motor torques (visual servoing) end to end. They use the 7 DOF arm of a PR2 robot to learn a variety of manipulation behaviors: hanging a coat hanger on a clothes rack, inserting a block into a shape sorting cube, fitting the claw of a toy hammer under a nail, and screwing a cap on a water bottle. A Convolutional Neural Network (CNN) controls the arm's movements at 20 Hz based on the visual input from a monocular RGB camera with a resolution of 240x240 pixels. A complicated training process involving several phases is required. The first layer of the CNN is initialized from a neural network that has been pretrained on the ImageNet dataset [Den+09]. In a second step, the image processing part of the neural network is initialized by training a pose regression CNN to predict 3D points that define the target objects involved in the task. GPS is used to train the final policy. The whole state of the system is observed while the local dynamic model is trained. The iLQG uses the full system state to obtain guiding policies. These guiding policies are used to train the neural network policy in a supervised setting but only with a partially observable state; the neural network only observes images from the robot's camera. As proposed by Levine and Abbeel [LA14] the objective of policy training includes a constraint on the distribution between successive policies to stabilize learning. The KL divergence between successive trajectory distributions is limited by a constant ϵ, which is similar to REPS. Depending on the task, the whole training process for a new behavior takes 3 to 4 hours with 156 to 288 episodes on the robot. This achievement relied on additional pretraining procedures and a fully observable state space during training, which makes the process of generating new behaviors difficult.

2.2.6.8. Criticism

Deep neural networks are powerful function approximators. Their complexity makes them vulnerable to adversarial examples [Sze+14] and overgeneralization [Jac+19]. Thus, a main problem of deep reinforcement learning is its sample efficiency. For example, HER [And+17] needs about 40,000 episodes to learn how to push a puck on a table to multiple goals with approximately 95 % success rate.

Nonetheless, in many Open AI gym [Bro+16] environments, particularly walking problems, deep RL algorithms do not actually use deep neural networks. For example, Lillicrap et al. [Lil+16] use a policy network composed of two hidden layers with 400 and 300

nodes if no visual input is used. Schulman et al. [Sch+17b] use two hidden layers with 64 nodes per layer. Schulman et al. [Sch+15c] use only one hidden layer with 30 nodes. Duan et al. [Dua+16] use 3 hidden layers with 100, 50, and 25 nodes. This suggests that parameter-linear models with a nonlinear feature projection could achieve similar performance. This has been demonstrated by Rajeswaran et al. [Raj+17b] for policy gradient algorithms. They show that radial basis functions as features can be used to even generalize better in these benchmarks. Mania et al. [MGR18] simplify the methods to solve these problems even further: they use a variant of random search [Mát65] with linear policies and show competitive performance in learning gaits for simple simulated mechanisms in comparison to TRPO and outperform SAC [Haa+18b], DDPG [Lil+16], and PPO [Sch+17b].

Engstrom et al. [Eng+20b] investigate whether the theoretical framework of TRPO and PPO is actually explaining how these trust region policy gradient algorithms work. They found that in practice PPO implementations make use of a lot of tricks that are not theoretically justified but are important. An example of such a trick is normalization of rewards to a preset range, for example, $[-5, 5]$. Furthermore, Ilyas et al. [Ily+18] find that PPO and TRPO operate with poor estimates of the policy gradient, as they use too few state-action samples per update. Using a value function as a baseline only slightly reduces the variance of the gradient estimate and often the approximation of the value function is poor, too. In addition, the trust region is not enforced properly by PPO, which makes it a bad approximation of TRPO.

Mania et al. [MGR18] also find that the reported performances in the benchmarks were susceptible to changes in the random seed and point out that the number of experiments that were run with deep reinforcement learning algorithms are low, which makes results questionable. In fact, Islam et al. [Isl+17] show that standard policy gradient algorithms such as DDPG and TRPO are fragile with respect to hyperparameter choices (such as network architecture, batch size). Hence, it is easy to improve in comparison to these algorithms if their parameters were not perfectly tuned, which results in unfair comparison. DDPG even gives different results when rewards are scaled with a constant factor. Islam et al. [Isl+17] show that even the variance over different random seeds is so large that it is not enough to perform five runs of the same experiment to show performance differences between algorithms, but many published works do not include performance evaluation over more than five runs. Mahmood et al. [Mah+18] evaluate algorithms such as TRPO, PPO, and DDPG in several simple robotic problems and also found that the performance of each algorithms strongly depends on the hyperparameters. Henderson et al. [HRP18] emphasize that optimizers and their hyperparameters are critical in particular. Henderson et al. [Hen+19] point out that not only random seeds and environment properties but also the implementation and hyperparameters can complicate reproduction of results of policy gradient algorithms.

There is not an established evaluation procedure in the deep reinforcement learning community [CSO18] and it is hard to tell if a method is robust enough to be worth the effort of implementing it for roboticists. Colas et al. [CSO18] at least give a practical hint

on the number of runs for an experiment to compute robust estimates of the standard deviation of the algorithm's performance: they suggest 20 or more.

2.2.7. Self-Supervised Learning

Self-supervised learning is supervised learning with a process that automatically generates desired outputs that should be approximated. Applications in behavior learning for robots so far are rare, although Levine et al. [Lev+18] show an impressive application in robotic grasping from raw monocular RGB camera images with a 7 DOF robot arm. Grasp success can be verified automatically in this application. The behavior is not learned end to end, but a CNN has been learned to predict the success of a motion command for a given camera image (and the camera image before the behavior is started). The behavior goes through a sequence of ten waypoints defined by the Cartesian end-effector position and the rotation of the 2-finger gripper around the z-axis. A motion command is selected in each step by an optimizer (CEM) based on the predicted success of the motion command. Although the whole approach is inspired by and similar to the work of Pinto and Gupta [PG16], the remarkable fact about this work is that a total amount of more than 800,000 plus 900,000 grasps collected in two datasets have been performed to train the grasp success prediction model and a maximum of 14 robots has been used in parallel to collect the data. A large variety of objects has been used to test the learned grasping behavior.

2.2.8. Discussion

Neural networks are a powerful framework to design function approximators that can represent complex value functions and policies. The benefit of such a powerful general function approximator is that it allows a conceptually easy integration of memory through recurrent layers or integration of sensors like cameras that might require complex processing architectures with convolutional layers.

If we want to train large neural network policies for continuous control, policy gradients and exploration in action space are most efficient [Dua+16]. Nevertheless, we have to be careful with policy gradient approaches, as they are brittle and have many critical hyperparameters that we have to tune. In the literature they are often not evaluated correctly with too few repetitions or not optimally tuned baseline algorithms.

Recent success of deep learning in the research community mainly comes from large datasets and huge computational resources, however, what makes deep learning really practically appealing is its enormous success with transfer learning. If we have a good pretrained model that generates useful features we can easily adapt it with only a few samples to a new but similar task. This has been shown in computer vision [Lin+17] and machine translation [Dev+19]. We did not see similar practically useful results of deep reinforcement learning for robotics so far. Transfer learning with pretrained models is difficult in this domain since the kinematic structure and the sensors that have been used in behavior learning are so diverse that transfer is hardly possible. Sometimes it is even difficult for similar robots [Lev+18]. Nonetheless, it will be a promising field

for the future. It will be interesting to see how transfer learning can be used with deep reinforcement learning, imitation learning, and self-supervised learning in the robotic domain or how we can encode robotic priors in neural networks.

Recently, there is a renewed interest in simple black-box optimizers such as ES [Rec71] in deep reinforcement learning. The works of Salimans et al. [Sal+17], Chrabaszcz et al. [CLH18], Ha and Schmidhuber [HS18], and Fuks et al. [Fuk+19] suggest that traditional black-box optimization can be competitive to policy gradient algorithms and value-function based reinforcement learning (particularly in the Atari domain [Bel+13]) even for larger neural networks although only the accumulated reward at the end of an episode is used. Song et al. [Son+20] state that ES only uses the total reward and not any information about individual rewards or state transitions is not always a weakness but it also can lead to more stability in the learning process. Salimans et al. [Sal+17] find that exploration in parameter space as it is done by ES has an advantage over exploration in action space as this is done by policy gradient algorithms for long episodes in which actions have a long lasting effect because the variance of the policy gradient often increases with the number of steps. Deisenroth et al. [DNP13] make similar observations and add that exploration in action space might lead to action sequences that might not be reproducible by a greedy policy without exploration noise, which affects the quality of the policy update.

In deep RL we often assume a direct sensor-actuator coupling: a current sensor measurement is fed into a neural network and the network generates an actuator command (for example, joint angles or torques) that is executed. This is a reactive behavior. In this thesis we will solve problems in which we have one sensor measurement from which we predict a sequence of actuator commands. Discrete and dynamic manipulation behaviors such as batting [PVS05; Mül+13], pancake flipping [KCC10b], or throwing [Gam+10] fall into this category.

We will focus on algorithms that already work well in this setting: policy search algorithms with movement primitives. We will select these categories of algorithms as a foundation for this thesis. Policy search combined with imitation learning has been proven to work reliably on real systems, as it often leads to a more stable learning process in comparison to other reinforcement learning approaches. Movement primitives are appealing since just one demonstration is enough to learn, for instance, a DMP that we can directly refine by policy search. Using a neural network as policy representation would complicate the integration of prior knowledge from non-experts by imitation learning because specific imitation learning procedures or many demonstrations would be required. Specifically, we will use contextual policy search algorithms to generalize behaviors over parameters of the tasks.

Movement primitives can also be refined by black-box optimization and since we will often encounter episodic settings in which the final reward is most relevant or considerably more relevant (ball-throwing or grasping) we will use these methods, too.

We will explicitly not learn state transition models. They can improve sample efficiency drastically as demonstrated by GPS [Lev+16]; however, they come with hyperparameters that are often difficult to tune and they could introduce bias that might lead to a worse

final performance than model-free algorithms. This decision will not limit the usefulness of the algorithms that we develop since they can be combined with state transition models to further increase sample efficiency. A brief overview of model-based reinforcement learning can be found in Appendix B.

2.3. A Detailed Overview of Contextual Policy Search

In policy search, the objective is to find parameters $\boldsymbol{\theta}$ of a policy $\pi_{\boldsymbol{\theta}}$ to maximize the expected return. A problem with this formulation is that it is not applicable to settings in which different tasks, which result in different returns for the same policy $\pi_{\boldsymbol{\theta}}$, are imposed on the agent. For instance, when an agent tries to throw a ball to different target positions (the tasks parameters), the same policy obtains different returns depending on the target position. To address such multi-task settings, we consider the problem of learning policies for contextual RL problems, that is, we assume that similar tasks are distinguished by context vectors $\boldsymbol{s} \in \mathcal{S} \subseteq \mathbb{R}^{n_s}$.

If we want to use state-space trajectory generators such as movement primitives it is obvious that we must separate context and state information, as a movement primitive directly modifies the state without indirection through actions. This eliminates the freedom of integrating arbitrary context information in the state representation, which we typically have in MDPs, because the state space of movement primitives can only include directly controllable variables. Although context information could be integrated in the state vector of a standard MDP, a state that describes the current situation, in which the agent is, and can be modified by the agent's actions, is conceptually different from a context, which is not directly under the agent's control. This led to similar extensions of the MDP concept in the absence of movement primitives. An example is the contextual MDP presented by Hallak et al. [HCM15] and Modi and Tewari [MT19]. In deep reinforcement learning a similar concept has been introduced by Schaul et al. [Sch+15a], who use DQN to learn value functions that are parameterized not only by a state but also by a goal vector. Although our context can be a goal, it can be something entirely different that describes the variations among different but similar reinforcement learning problems. The only restriction that we have is that contexts must be n_s-dimensional real vectors. Therefore, we use the terms goal, task, and context mostly interchangeably.

In contextual policy search we learn an *upper-level policy* $\pi_{\boldsymbol{\omega}}(\boldsymbol{\theta}|\boldsymbol{s})$, which is parameterized by $\boldsymbol{\omega}$ and defines a probability distribution over the parameters $\boldsymbol{\theta}$ of the actual control policy $\pi_{\boldsymbol{\theta}}$. The upper-level policy should be selected according to

$$\arg\max_{\boldsymbol{\omega}} \int_{\mathcal{S}} p(\boldsymbol{s}) \int_{\mathbb{R}^n} \pi_{\boldsymbol{\omega}}(\boldsymbol{\theta}|\boldsymbol{s}) \mathbb{E}\left[R(\boldsymbol{s},\boldsymbol{\theta})\right] \; d\boldsymbol{\theta} d\boldsymbol{s},$$

where $\mathbb{E}\left[R(\boldsymbol{s},\boldsymbol{\theta})\right]$ is the expected return of policy $\pi_{\boldsymbol{\theta}}$ in context $\boldsymbol{s}$ and $p(\boldsymbol{s})$ is the probability density function of the context distribution. This is an extension to the original policy search formulation.

Contextual policy search algorithms are iterative and a typical iteration includes:

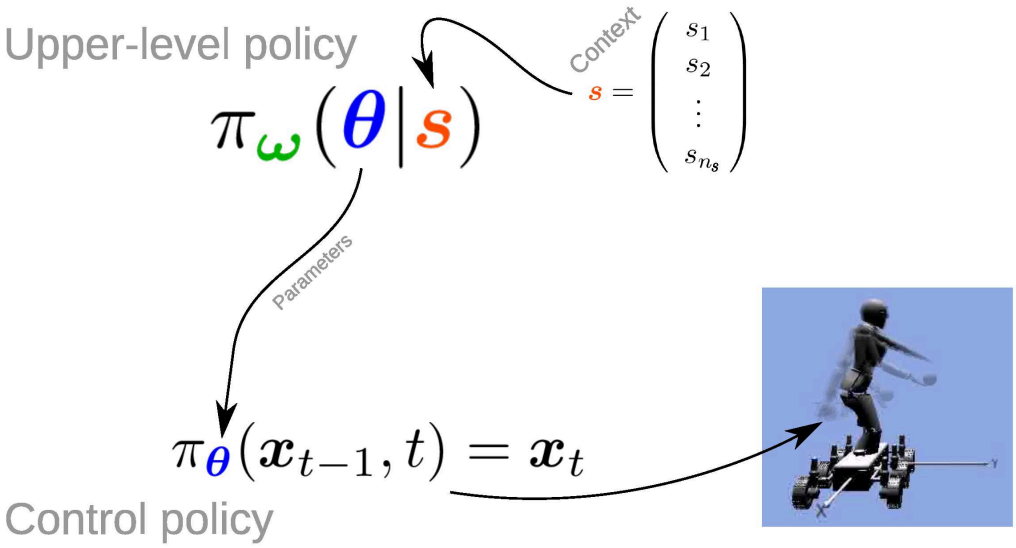


Figure 2.12.: Illustration of experience collection in contextual policy search. We observe a context based on which we sample control policy parameters from the upper-level policy. The control policy can be a time-dependent movement primitive such as a DMP. It will be executed on the robot to sample a return.

1. Sample a dataset of corresponding contexts $\boldsymbol{s}$, control policy parameters $\boldsymbol{\theta}$, and returns $R(\boldsymbol{s}, \boldsymbol{\theta})$ based on the current upper-level policy $\pi_\omega(\boldsymbol{\theta}|\boldsymbol{s})$ (see Figure 2.12).
2. Compute weights for each sample $(\boldsymbol{s}, \boldsymbol{\theta})$ based on $R(\boldsymbol{s}, \boldsymbol{\theta})$.
3. Update $\pi_\omega(\boldsymbol{\theta}|\boldsymbol{s})$ with the collected dataset based on *weighted regression*.

We will discuss details in the following subsections. In Section 2.3.2 we will analyze differences in calculation of sample weights that will be used in weighted regression to learn a stochastic upper-level policy $\pi_\omega(\boldsymbol{\theta}|\boldsymbol{s})$, hence, the underlying regression algorithm has to support sample weights and must be able to predict a (co)variance so that we can sample parameters for the control policy. We will see in Sections 2.3.3 and 2.3.4 that there is a broad class of algorithms that can be used for this purpose. Section 2.3.5 contains detailed descriptions of two concrete contextual policy search algorithms. First of all, however, we will discuss similarities to black-box optimization.

2.3.1. Contextual Black-Box Optimization

A general and deterministic problem formulation that is similar to contextual policy search is

$$\arg\min_g \int_S f_{\boldsymbol{s}}\left(g(\boldsymbol{s})\right) d\boldsymbol{s},$$

where f_s is a parameterized objective function and we want to find an optimal function $g(s)$. We call this the contextual black-box optimization problem. This, of course, is an extremely difficult problem, which is relaxed by restricting the problem to a parameterized class of functions g_ω (often linear functions with a nonlinear projection of the context such as polynomials). The optimization problem becomes

$$\arg\min_{\omega} \int_{\mathbb{S}} f_s(g_\omega(s))ds.$$

The challenge of contextual black-box optimization in comparison to black-box optimization is that the true objective function is an integral over all possible context. We can only compute a sample-based approximation. Contextual policy search and contextual black-box optimization correspond to each other like policy search and black-box optimization. Contextual black-box optimization can benefit from the ideas of black-box optimization as contextual policy search can benefit from policy search.

2.3.2. Computing Weights from Returns

Each contextual policy search algorithm has a different way of computing sample weights. RWR [PS07] computes the weights based on the return by $d_i = \tau \exp(-\tau R_i)/Z$ and sets Z so that the weights sum up to one. In each iteration the hyperparameter is updated by $\tau = \left(\Sigma_{i=1}^{N} d_i\right) / \left(\Sigma_{i=1}^{N} d_i R_i\right)$, which requires $R > 0$. RWR's kernel-based extension CrKR [Kob+12] uses returns as sample weights, which requires $R > 0$.

VIP [Neu11] transforms rewards to probabilities by $p_i = \exp\left((R(s_i, \theta_i) - V(s_i)) \,/\, \eta\right)$ and iteratively obtains sample weights d_i and a scaling factor η through gradient descent so that the KL divergence is minimized between the search distribution and the reward-weighted trajectory distribution p_R estimated by $p_R(i) = d_i p_i$. VIP normalizes weights so that their maximum is 1 and estimates the baseline $V(s)$ from previous samples.

C-REPS [Kup+13] computes weights by $d_i = \exp\left((R(s_i, \theta_i) - V(s_i)) \,/\, \eta\right) \,/\, Z$, where the baseline $V(s)$ and the parameter η are optimized so that the KL divergence between the old and the new policy in an update is less than a specified hyperparameter. Z normalizes all weights so that they sum up to one.

C-CMA-ES [Abd+17a] estimates a context value function $V(s)$ to compute the context advantage function $\hat{A}(s_i, \theta_i) = R(s_i, \theta_i) - \hat{V}(s_i)$. The advantage values will be used to order samples and their weights $d_i = \max(0, (\ln \mu + 0.5) - \ln(i))/Z$ are defined based on each sample's rank $i \in 1, \ldots, \mu, \ldots, N$. Z normalizes weights so that they sum up to one.

In most approaches we see similar patterns: in more recent algorithms a baseline $V(s)$ is subtracted from rewards, rewards are transformed with an exponential function before they will be processed further to make them strictly positive, the argument of the exponential function includes an adaptive scaling factor (η or τ) that can be used to put more or less emphasis on the best samples, and weights of a dataset are normalized.

2.3.3. Weighted Regression

Weighted regression is the basis of most contextual policy search algorithms. We will summarize existing results and show similarities between algorithms. Note that deriva-

tions are not rigorous and formally complete, as we will skip steps and focus on the most interesting parts to get an overview of the foundations of contextual policy search (although we can say the same about original publications on contextual policy search).

The goal of regression is to approximate an unknown latent function $f : \mathbb{R}^D \to \mathbb{R}^F$. In linear regression we assume Gaussian measurement noise, that is, we collect a dataset $\mathcal{D}$ of N samples $(\boldsymbol{x}, \boldsymbol{y}) \in \mathbb{R}^D \times \mathbb{R}^F$ from the true function, where $\boldsymbol{y} = f(\boldsymbol{x}) + \boldsymbol{\epsilon}$ and $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$. Note that $\boldsymbol{x}$ in this section is not a state but just an arbitrary real vector and input of the model. We will actually use the context vector $\boldsymbol{s}$ as input in the following sections, but here we stick to the standard notation, which uses $\boldsymbol{x}$ for feature vectors (inputs), since the algorithms that we discuss in this section are general regression algorithms.

In linear regression we seek to maximize the likelihood of the model given the data. This is equivalent to minimizing the negative log-likelihood or the sum of squared errors. To find the maximum likelihood estimate for f, we have to assume a parametric form of f, in this case a linear function. With one output dimension ($F = 1$) this reduces to

$$f_{\boldsymbol{w}}(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}),$$

where $\boldsymbol{\phi}(\boldsymbol{x}) \in \mathbb{R}^{D'}$ is a nonlinear feature transformation and the hypothesis space is fully defined by all possible $\boldsymbol{w} \in \mathbb{R}^{D'}$ and $\boldsymbol{\Sigma} = \sigma^2 \in \mathbb{R}$. The probability of the dataset given the hypothesis can hence be computed as

$$p(\mathcal{D}|\boldsymbol{w}, \sigma^2) = \prod_{(\boldsymbol{x},\boldsymbol{y}) \in \mathcal{D}} \mathcal{N}(\boldsymbol{y}|\boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}), \sigma^2),$$

which directly translates to $p(\mathcal{D}|\boldsymbol{w}, \sigma^2) = \mathcal{L}(\boldsymbol{w}, \sigma^2|\mathcal{D})$, the likelihood of the hypothesis given the dataset. The maximum likelihood estimate is a solution to the optimization problem

$$\arg\max_{\boldsymbol{w}, \sigma^2} \mathcal{L}(\boldsymbol{w}, \sigma^2|\mathcal{D}).$$

Since the negative logarithm is a monotonic function, we can simplify the problem and find the solution by minimizing the negative log-likelihood

$$\arg\min_{\boldsymbol{w}, \sigma^2} -\ln(\mathcal{L}(\boldsymbol{w}, \sigma^2|\mathcal{D})).$$

After a lot of simplifications the negative log-likelihood almost reduces to the sum of squared errors (see Bishop [Bis06, page 141, Equation 3.11])

$$-\ln \mathcal{L}(\boldsymbol{w}, \sigma^2|\mathcal{D}) = \left[\frac{N}{2} \ln(2\pi) + \frac{N}{2} \ln(\sigma^2) + \frac{1}{2\sigma^2} \sum_{(\boldsymbol{x},\boldsymbol{y}) \in \mathcal{D}} \left(\boldsymbol{y} - \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}) \right)^2 \right].$$

We have to find the zero of the derivative to compute the optimum. To prove that it is a minimum we have to show that the second derivative (Hessian matrix) is positive definite. We are at the moment only interested in the solution for $\boldsymbol{w}$, which we can find with normal equations. For more complex nonlinear models such as neural networks this

Table 2.1.: Comparison of weighted regression. $\boldsymbol{\Phi}$: design matrix; $\boldsymbol{K} = \boldsymbol{\Phi}\boldsymbol{\Phi}^T$: Gram matrix; $\boldsymbol{y}$ targets; $\boldsymbol{D}$: diagonal sample weight matrix; $\boldsymbol{w}, \boldsymbol{\alpha}$: model parameters; λ: regularization coefficient

Algorithm	Objective	Solution
Linear Regression	$\min_{\boldsymbol{w}} \frac{1}{2} (\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{w})^T (\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{w})$	$\boldsymbol{w} = \left(\boldsymbol{\Phi}^T\boldsymbol{\Phi}\right)^{-1} \boldsymbol{\Phi}^T\boldsymbol{y}$
Weighted Linear Regression	$\min_{\boldsymbol{w}} \frac{1}{2} (\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{w})^T \boldsymbol{D} (\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{w})$	$\boldsymbol{w} = \left(\boldsymbol{\Phi}^T\boldsymbol{D}\boldsymbol{\Phi}\right)^{-1} \boldsymbol{\Phi}^T\boldsymbol{D}\boldsymbol{y}$
Ridge Regression	$\min_{\boldsymbol{w}} \frac{1}{2} (\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{w})^T (\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{w}) + \frac{\lambda}{2}\boldsymbol{w}^T\boldsymbol{w}$	$\boldsymbol{w} = \left(\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \lambda\boldsymbol{I}\right)^{-1} \boldsymbol{\Phi}^T\boldsymbol{y}$
Weighted Ridge Regression	$\min_{\boldsymbol{w}} \frac{1}{2} (\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{w})^T \boldsymbol{D} (\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{w}) + \frac{\lambda}{2}\boldsymbol{w}^T\boldsymbol{w}$	$\boldsymbol{w} = \left(\boldsymbol{\Phi}^T\boldsymbol{D}\boldsymbol{\Phi} + \lambda\boldsymbol{I}\right)^{-1} \boldsymbol{\Phi}^T\boldsymbol{D}\boldsymbol{y}$
Kernel Regression	$\min_{\boldsymbol{\alpha}} \frac{1}{2} (\boldsymbol{y} - \boldsymbol{K}\boldsymbol{\alpha})^T (\boldsymbol{y} - \boldsymbol{K}\boldsymbol{\alpha})$	$\boldsymbol{\alpha} = \boldsymbol{K}^{-1}\boldsymbol{y}$
Weighted Kernel Regression	$\min_{\boldsymbol{\alpha}} \frac{1}{2} (\boldsymbol{y} - \boldsymbol{K}\boldsymbol{\alpha})^T \boldsymbol{D} (\boldsymbol{y} - \boldsymbol{K}\boldsymbol{\alpha})$	$\boldsymbol{\alpha} = (\boldsymbol{D}\boldsymbol{K})^{-1} \boldsymbol{D}\boldsymbol{y}$
Kernel Ridge Regression	$\min_{\boldsymbol{\alpha}} \frac{1}{2} (\boldsymbol{y} - \boldsymbol{K}\boldsymbol{\alpha})^T (\boldsymbol{y} - \boldsymbol{K}\boldsymbol{\alpha}) + \frac{\lambda}{2}\boldsymbol{\alpha}^T\boldsymbol{K}\boldsymbol{\alpha}$	$\boldsymbol{\alpha} = \left(\boldsymbol{K}^{-1} + \lambda\boldsymbol{I}\right)\boldsymbol{y}$
Weighted Kernel Ridge Regression	$\min_{\boldsymbol{\alpha}} \frac{1}{2} (\boldsymbol{y} - \boldsymbol{K}\boldsymbol{\alpha})^T \boldsymbol{D} (\boldsymbol{y} - \boldsymbol{K}\boldsymbol{\alpha}) + \frac{\lambda}{2}\boldsymbol{\alpha}^T\boldsymbol{K}\boldsymbol{\alpha}$	$\boldsymbol{\alpha} = (\boldsymbol{D}\boldsymbol{K} + \lambda\boldsymbol{I})^{-1} \boldsymbol{D}\boldsymbol{y}$
Cost-regularized Kernel Regression	$\min_{\boldsymbol{\alpha}} \frac{1}{2} (\boldsymbol{y} - \boldsymbol{K}\boldsymbol{\alpha})^T (\boldsymbol{y} - \boldsymbol{K}\boldsymbol{\alpha}) + \frac{\lambda}{2}\boldsymbol{\alpha}^T\boldsymbol{\Phi}\boldsymbol{D}^{-1}\boldsymbol{\Phi}^T\boldsymbol{\alpha}$	$\boldsymbol{\alpha} = (\boldsymbol{K} + \lambda\boldsymbol{D}^{-1})^{-1}\boldsymbol{y}$

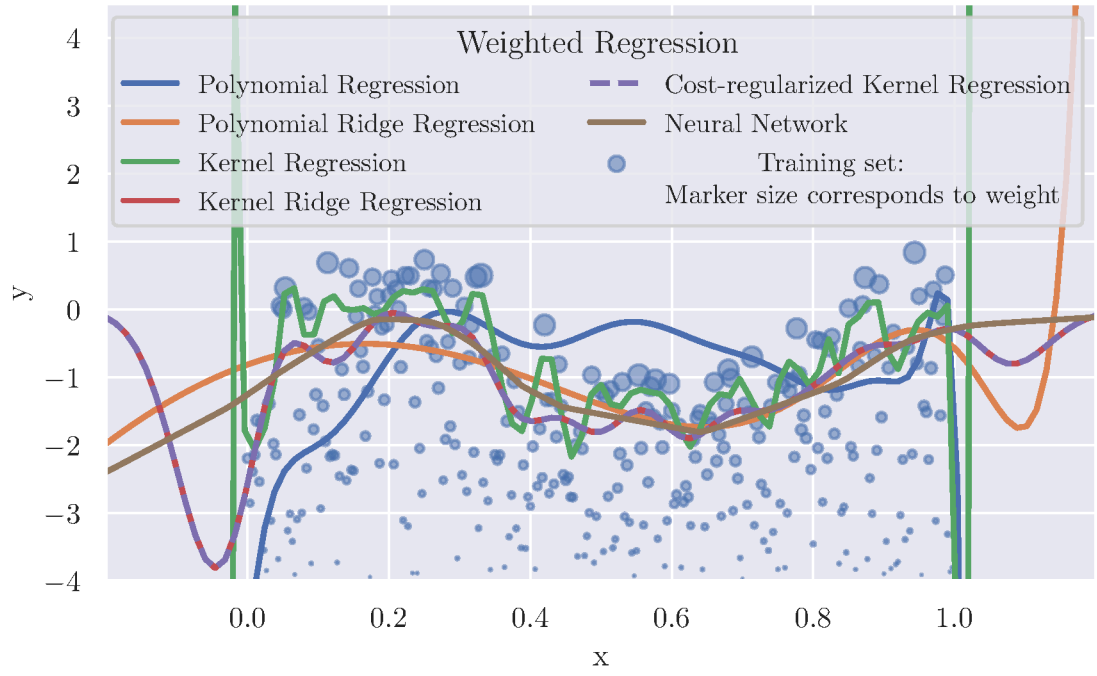


Figure 2.13.: Comparison of weighted linear regression methods. Methods with Tikhonov regularization have a regularization coefficient $\lambda = 10^{-4}$, polynomial regression uses 20 degrees, kernel-based methods use an RBF kernel with $\gamma = 100$, and the neural network has 500 and 500 hidden units.

might not be trivial and it might be infeasible to find the global optimum of the objective function. At this point we can introduce sample weights in the objective to weight each sample's contribution to the total sum of squared errors.

See Table 2.1 for an overview of algorithms, objectives, and solutions for variations of least squares regression and their counterparts that include sample weights. Note that we made some simplifications here. We assume one-dimensional outputs. Hence, we can write all outputs from the training set in a single vector $\boldsymbol{y} \in \mathbb{R}^N$. We can also create a design matrix $\boldsymbol{\Phi}$ that contains the n-th projected feature vector $\boldsymbol{\phi}(\boldsymbol{x}_n)^T$ in its n-th row. Sample weights are organized in a diagonal matrix $\boldsymbol{D}$.

From linear models we can derive nonlinear methods through nonlinear feature projection or kernel methods. Kernel methods replace the weight vector $\boldsymbol{w}$ by $\boldsymbol{\Phi}^T \boldsymbol{\alpha}$ to introduce scalar products that can be replaced by kernels. We can also apply Tikhonov regularization, that is, we can penalize the Euclidean norm of the weight vector $\boldsymbol{w}$. Thus, we do not just maximize the likelihood but we also have a prior on the distribution of $\boldsymbol{w}$ and, hence, compute the maximum a posteriori (MAP) estimate [Mur12, pages 225–227].

Figure 2.13 compares several weighted regression methods on a simple one-dimensional weighted regression problem. It also contains a neural network that minimizes the loss that we will discuss in Section 2.3.4.

Sample weights can be integrated in two different ways in kernel methods: they can be used directly to mitigate the error contribution of samples with low weight or they can be used in kernel methods to apply different regularization coefficients per sample.[3] The latter has been introduced by Kober et al. [Kob+12] in the algorithm CrKR. Since the derivation is interesting and the most complicated one of the presented methods, we will discuss it in Appendix D *Derivation of Cost-Regularized Kernel Regression.*

2.3.4. Regression with Uncertainty Estimation

Figure 2.14 compares three different approaches to uncertainty estimation. For simplicity we omit sample weights in these plots. A detailed discussion on categories of uncertainty estimates has been published by Kendall and Gal [KG17]. We briefly summarize it here and apply it to regression models for contextual policy search.

Epistemic and aleatoric uncertainty can be distinguished. Epistemic uncertainty represents uncertainty in model parameters. We can reduce it by collecting more data. Aleatoric uncertainty represents observation noise. If we observe multiple output values for the same input, the variance of these output values will be an aleatoric estimate. Furthermore, we distinguish homoscedastic and heteroscedastic aleatoric uncertainty. Homoscedastic uncertainty is equal for the whole input domain, that is, the predicted mean depends on the input but the covariance matrix does not (Gaussian: $\mathcal{N}(\boldsymbol{\mu}(\boldsymbol{x}), \boldsymbol{\Sigma})$). Heteroscedastic uncertainty depends on the input (Gaussian: $\mathcal{N}(\boldsymbol{\mu}(\boldsymbol{x}), \boldsymbol{\Sigma}(\boldsymbol{x}))$). We can estimate a full covariance matrix, a diagonal covariance matrix, or isotropic uncertainty ($\boldsymbol{\Sigma} = \sigma^2 \boldsymbol{I}$).

Let us take a look at contextual policy search algorithms and analyze their uncertainty prediction properties. RWR [PS07] is the first algorithm that we can categorize as contextual policy search. In its original form RWR uses a probabilistic upper-level policy. The model is $\mathcal{N}(\boldsymbol{y}|\boldsymbol{\mu}(\boldsymbol{x}), \sigma^2 \boldsymbol{I})$, where $\boldsymbol{\mu}(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x})$ with $\boldsymbol{w}, \boldsymbol{\phi}(\boldsymbol{x}) \in \mathbb{R}^{D'}$ and $\boldsymbol{y} \in \mathbb{R}^F$. The covariance is isotropic as its variance is the same for all output dimensions and there are no correlations. The distribution represents aleatoric uncertainty, that is, the weighted variance of samples. It is obtained through weighted maximum likelihood, that is,

$$\sigma^2 = \frac{1}{F} \mathrm{Tr}\left[(\boldsymbol{Y} - \boldsymbol{\Phi W})^T \boldsymbol{D} (\boldsymbol{Y} - \boldsymbol{\Phi W}) \right],$$

where all weights sum up to one ($\boldsymbol{Tr}\left[\boldsymbol{D}\right] = 1$). The uncertainty estimate is homoscedastic since the variance does not depend on the input vector.

C-REPS [Kup+13] uses a similar kind of uncertainty estimate based on weighted maximum likelihood for the full covariance

$$\boldsymbol{\Sigma} = c \cdot (\boldsymbol{Y} - \boldsymbol{\Phi W})^T \boldsymbol{D} (\boldsymbol{Y} - \boldsymbol{\Phi W}),$$

[3] This leads to the same result as Figure 2.13 suggests. Compare Weighted Kernel Ridge Regression and Cost-regularized Kernel Regression.

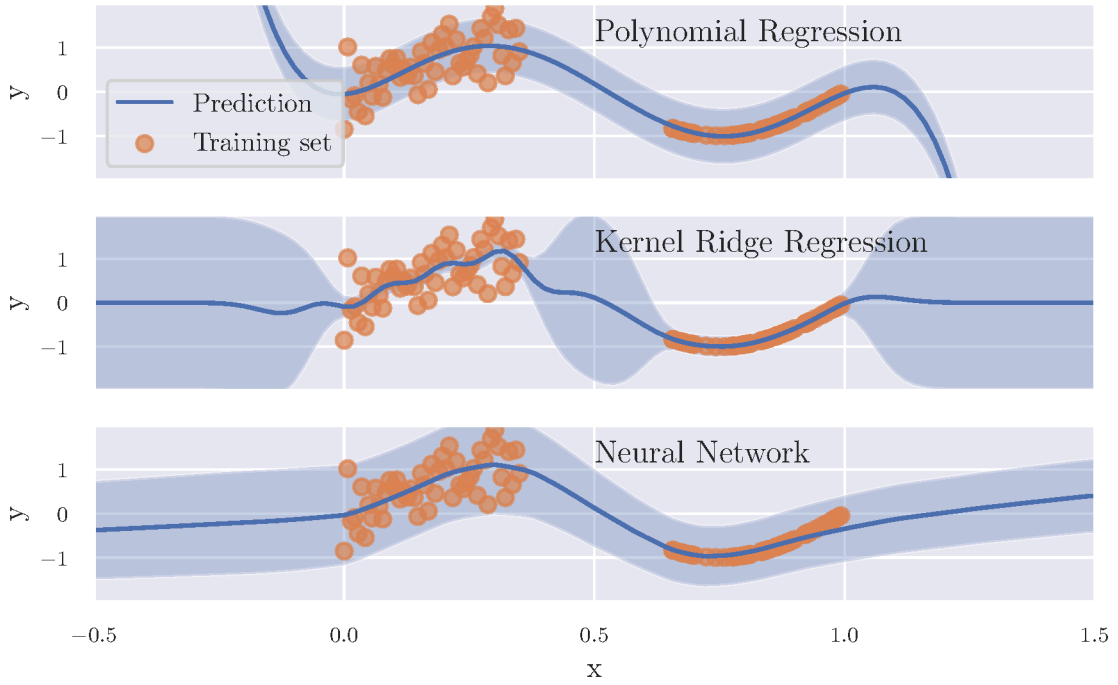


Figure 2.14.: Comparison of several uncertainty estimates. The transparent area indicates the 95 % confidence interval ($\mu \pm 1.96\sigma$) of a prediction.

with $c = 1\,/\,(1 - Tr\,[\boldsymbol{DD}])$ to obtain an unbiased estimate [DNP13] (requires $Tr\,[\boldsymbol{D}] = 1$).

VIP [Neu11] also uses the weighted maximum likelihood estimate of the covariance but incorporates previous covariances in the update. The influence of those is determined by weights that will be optimized through gradient descent.

C-CMA-ES [Abd+17a] uses weighted maximum likelihood with additional tricks to find a good covariance matrix for exploration. It computes an average of the previous covariance, an estimate based on previous search directions, and the weighted maximum likelihood estimate. Furthermore, the covariance matrix is scaled separately.

CrKR [Kob+12] predicts the variance with

$$\sigma^2 = k(\boldsymbol{x}, \boldsymbol{x}) + \lambda - \boldsymbol{k}^T \left(\boldsymbol{K} + \lambda \boldsymbol{D}^{-1}\right)^{-1} \boldsymbol{k}.$$

For the RBF kernel the term $k(\boldsymbol{x}, \boldsymbol{x})$ is 1 and λ is a hyperparameter that is constant. Hence, only the term $-\boldsymbol{k}^T \left(\boldsymbol{K} + \lambda \boldsymbol{D}^{-1}\right)^{-1} \boldsymbol{k}$ is a non-constant term. It reduces the variance based on the values of the kernels. Since we use the RBF kernel, the closer $\boldsymbol{x}$ is to the training data, the lower is the variance. This is a form of epistemic uncertainty prediction. Gaussian process regression also estimates homoscedastic aleatoric uncertainty by adding a regularization kernel and optimizing its coefficients.

According to Kendall and Gal [KG17] we can predict heteroscedastic aleatoric uncertainty with a neural network that has two outputs $\mu_{\boldsymbol{w}}(\boldsymbol{x}), \sigma_{\boldsymbol{w}}(\boldsymbol{x})$ by minimizing negative log-likelihood. We include sample weights d_i ($\sum_{i=1}^{N} d_i = 1$) in the loss and extend it to

$$L(\boldsymbol{w}) = \frac{1}{2} \sum_{i=1}^{N} d_i \left(\ln \sigma_{\boldsymbol{w}}^2(\boldsymbol{x}_i) + \frac{||\mu_{\boldsymbol{w}}(\boldsymbol{x}_i) - \boldsymbol{y}_i||^2}{\sigma_{\boldsymbol{w}}^2(\boldsymbol{x}_i)} \right).$$

Uncertainty estimates influence the exploration behavior of the agent in policy parameter space. Why do algorithms use fundamentally different approaches to uncertainty estimation—aleatoric and epistemic uncertainty? Epistemic uncertainty explicitly fosters exploration of previously unexplored regions, which is the most reasonable approach. Aleatoric uncertainty estimates would actually not be enough to guarantee sufficient exploration and avoid premature convergence, however, because each training example has a weight that is proportional to its return, aleatoric uncertainty increases more along the directions to examples with high returns. This is still not sufficient for some algorithms, hence, VIP and C-CMA-ES use additional modifications of the uncertainty update to control exploration. In practice we also modify C-REPS to include regularization in covariance estimation.

2.3.5. Contextual Policy Search Algorithms

After pointing out similarities of contextual policy search algorithms, let us take a closer look at specific algorithms that we will use in the following chapters.

2.3.5.1. Contextual Relative Entropy Policy Search

This section was published originally in [FM14] and has been revised.

C-REPS allows any policy which can be learned by weighted maximum likelihood. A frequently used variant is $\pi_{\boldsymbol{\omega}}(\boldsymbol{\theta}|\boldsymbol{s}) = \mathcal{N}\left(\boldsymbol{\theta}|\boldsymbol{W}^T\boldsymbol{\phi}(\boldsymbol{s}), \boldsymbol{\Sigma}\right)$, where $\boldsymbol{\phi}(\boldsymbol{s})$ contains the linear and quadratic terms of the context vector $\boldsymbol{s}$, $\boldsymbol{W}$ is a weight matrix, and $\boldsymbol{\Sigma}$ the covariance of the stochastic policy. C-REPS has been explained in detail by Deisenroth et al. [DNP13] and Kupcsik et al. [Kup+13].

We summarize C-REPS in Algorithm 2. A stochastic policy $\pi_{\boldsymbol{\omega}}(\boldsymbol{\theta}|\boldsymbol{s})$ is used to generate low-level controllers $\pi_{\boldsymbol{\theta}}$ for a context $\boldsymbol{s}$. We use a Gaussian policy with linear mean [DNP13] so that $\boldsymbol{\omega} = (\boldsymbol{W}, \boldsymbol{\Sigma})$. After collecting N experience tuples $(\boldsymbol{s}_i, \boldsymbol{\theta}_i, R(\boldsymbol{s}_i, \boldsymbol{\theta}_i))$ (lines 2–6), C-REPS determines a weight d_i for each experience based on the context-dependent baseline $V(\boldsymbol{s}) = \boldsymbol{v}^T\boldsymbol{\phi}(\boldsymbol{s})$ (lines 7–8). With the obtained weights, C-REPS updates the policy by weighted maximum likelihood (lines 14–15). Note that this solution extends the solutions from Section 2.3.3 to multiple output dimensions.

Implementing C-REPS is not straightforward. In our experiments and for our reward functions it was sometimes numerically unstable. To improve the reproducibility of the results, we explain some of the modifications that we made in addition to the log-sum-exp trick (in line 7) that is already mentioned by Deisenroth et al. [DNP13].

The weighted least squares problem is sometimes ill-conditioned. Hence, we added the regularization term $\lambda\boldsymbol{I}$ so that

$$\boldsymbol{W}_{\text{new}} \leftarrow (\boldsymbol{\Phi}^T\boldsymbol{D}\boldsymbol{\Phi} + \lambda\boldsymbol{I})^{-1}\boldsymbol{\Phi}^T\boldsymbol{D}\boldsymbol{\Theta}.$$

Algorithm 2 C-REPS

Require: ϵ: maximum Kullback-Leibler divergence between two successive policy distributions; $\phi(\boldsymbol{s})$: extracts features from the context $\boldsymbol{s}$; N: number of samples per update; $\boldsymbol{W}$: initial weights of upper level policy; $\boldsymbol{\Sigma}$: initial covariance of upper level policy distribution

while not converged **do**
  for $i \in \{1, \ldots, N\}$ **do**
    Observe $\boldsymbol{s}_i$ ▷ Draw from context distribution $p(\boldsymbol{s})$
    $\boldsymbol{\theta}_i \sim \mathcal{N}(\boldsymbol{W}^T \phi(\boldsymbol{s}_i), \boldsymbol{\Sigma})$ ▷ Draw policy parameters
    Obtain $R(\boldsymbol{s}_i, \boldsymbol{\theta}_i)$ ▷ Return of policy $\pi_{\boldsymbol{\theta}_i}$ in environment with context $\boldsymbol{s}_i$
  end for
  Solve Lagrangian dual problem $[\eta, \boldsymbol{v}] = \arg\min_{\eta', \boldsymbol{v}'} g(\eta', \boldsymbol{v}')$, subject to $\eta > \eta_{\min}$

$$g(\eta, \boldsymbol{v}) = \eta\epsilon + \boldsymbol{v}^T \hat{\phi} + \eta \ln \left(\sum_{i=1}^{N} \frac{1}{N} \exp\left(\frac{R(\boldsymbol{s}_i, \boldsymbol{\theta}_i) - \boldsymbol{v}^T \phi(\boldsymbol{s}_i)}{\eta} \right) \right)$$

▷ $\hat{\phi}$ are average context features.

  $\boldsymbol{D}_{ij} \leftarrow \frac{\delta_{ij}}{Z} \exp\left(\frac{R(\boldsymbol{s}_i, \boldsymbol{\theta}_i) - \boldsymbol{v}^T \phi(\boldsymbol{s}_i)}{\eta} \right)$
  ▷ Z is chosen so that weights sum up to one, $\boldsymbol{D}$ is a diagonal matrix
  for $i \in \{1, \ldots, N\}$ **do** ▷ Prepare policy update
    $\boldsymbol{\Phi}_i = \phi(\boldsymbol{s}_i)^T$
    $\boldsymbol{\Theta}_i = \boldsymbol{\theta}_i^T$
  end for
  $\boldsymbol{W} \leftarrow (\boldsymbol{\Phi}^T \boldsymbol{D} \boldsymbol{\Phi})^{-1} \boldsymbol{\Phi}^T \boldsymbol{D} \boldsymbol{\Theta}$ ▷ Update policy mean
  $\boldsymbol{\Sigma} \leftarrow \frac{1}{1 - Tr[\boldsymbol{D}\boldsymbol{D}]} (\boldsymbol{\Theta} - \boldsymbol{\Phi}\boldsymbol{W})^T \boldsymbol{D} (\boldsymbol{\Theta} - \boldsymbol{\Phi}\boldsymbol{W})$ ▷ Update exploration covariance
end while

Towards the end of the learning process, η sometimes becomes tiny which causes numerical problems with the original formulation of C-REPS, which uses $\eta_{\min} = 0$. It is helpful to use another lower bound $\eta_{\min}$ so that $\eta > \eta_{\min}$. We suggest $\eta_{\min} \in \left[10^{-8}, 10^{-4}\right]$.

For large returns the weights $d_i = \boldsymbol{D}_{ii}$ are sometimes so big that the result of the exponential function cannot be represented properly with 64-bit floating-point numbers. Fortunately, d_i is a softmax term, which can be implemented numerically stable as

$$d_i = \frac{\exp\left(\frac{R(\boldsymbol{s}_i, \boldsymbol{\theta}_i) - \boldsymbol{v}^T \phi(\boldsymbol{s}_i)}{\eta} \right)}{\sum_j \exp\left(\frac{R(\boldsymbol{s}_j, \boldsymbol{\theta}_j) - \boldsymbol{v}^T \phi(\boldsymbol{s}_j)}{\eta} \right)} = \frac{\exp\left(\frac{R(\boldsymbol{s}_i, \boldsymbol{\theta}_i) - \boldsymbol{v}^T \phi(\boldsymbol{s}_i)}{\eta} - m \right)}{\sum_j \exp\left(\frac{R(\boldsymbol{s}_j, \boldsymbol{\theta}_j) - \boldsymbol{v}^T \phi(\boldsymbol{s}_j)}{\eta} - m \right)},$$

where $m = \max_i \left(R(\boldsymbol{s}_i, \boldsymbol{\theta}_i) - \boldsymbol{v}^T \phi(\boldsymbol{s}_i) \right) / \eta$ so that the maximum argument of an exponential function in this computation is 0.

2.3.5.2. Contextual Covariance Matrix Adaptation Evolution Strategies

This section was published originally in [Fab19a] and has been revised.

C-CMA-ES only has a few critical hyperparameters for which good default values are known. It does not suffer from premature convergence like C-REPS. C-CMA-ES is based on CMA-ES [HO01]. The original publication does not give a complete and correct listing

Algorithm 3 C-CMA-ES

Require: N: update frequency; μ: number of samples used for the update; $\phi(\boldsymbol{s}), \psi(\boldsymbol{s})$: context transformations; λ: regularization coefficient; n: parameter dimension; n_s: context dimension; σ^0: step size; t: step counter, initially 1; search distribution is initialized to $\boldsymbol{W}^0 = \boldsymbol{\Sigma}^0 = \boldsymbol{I}$

$\boldsymbol{D}_{ij} \leftarrow \frac{\delta_{ij}}{Z} \max(0, (\ln \mu + 0.5) - \ln(i))$

▷ Diagonal sample weight matrix $\boldsymbol{D} \in \mathbb{R}^{N \times N}$, Z is chosen so that weights sum up to one

$\mu_{eff} \leftarrow \frac{1}{\sum_{i=1}^{N} \boldsymbol{D}_{ii}^2}$

$c_1 \leftarrow 2/\left((n + n_s + 1.3)^2 + \mu_{eff}\right)$

$c_\mu \leftarrow \min\left(1 - c_1, \frac{2\left(\mu_{eff} - 2 + \frac{1}{\mu_{eff}}\right)}{(n+n_s+2)^2 + \mu_{eff}}\right)$

$c_c \leftarrow \frac{4 + \frac{\mu_{eff}}{n+n_s}}{4 + n + n_s + 2\frac{\mu_{eff}}{n+n_s}}$

$c_\sigma \leftarrow \frac{\mu_{eff} + 2}{n + n_s + \mu_{eff} + 5}$

$d_\sigma \leftarrow 1 + 2\max\left(0, \sqrt{\frac{(\mu_{eff} - 1)}{(n+n_s+1)}} - 1\right) + c_\sigma + \ln(n + n_s + 1)$

$\mathbb{E}||\mathcal{N}(0, \boldsymbol{I})|| \leftarrow \sqrt{n}\left(1 - \frac{1}{4n} + \frac{1}{21n^2}\right)$

while not converged **do**

  for $i \in \{1, \ldots, N\}$ **do**

    Observe $\boldsymbol{s}_i$ ▷ Draw from context distribution $p(\boldsymbol{s})$

    $\boldsymbol{\theta}_i \sim \mathcal{N}(\boldsymbol{W}^{tT} \phi(\boldsymbol{s}_i), (\sigma^t)^2 \boldsymbol{\Sigma}^t)$ ▷ Draw policy parameters

    Obtain $R(\boldsymbol{s}_i, \boldsymbol{\theta}_i)$ ▷ Return of policy $\pi_{\boldsymbol{\theta}_i}$ in environment with context $\boldsymbol{s}_i$

  end for

  Build $\boldsymbol{\Phi}$, $\boldsymbol{\Psi}$, $\boldsymbol{\Theta}$, and $\boldsymbol{R}$, where $\boldsymbol{\Phi}_i = \phi(\boldsymbol{s}_i)^T$, $\boldsymbol{\Psi}_i = \psi(\boldsymbol{s}_i)^T$, $\boldsymbol{\Theta}_i = \boldsymbol{\theta}_i^T$, $\boldsymbol{R}_i = R(\boldsymbol{s}_i, \boldsymbol{\theta}_i)$

  $\boldsymbol{B}^t \leftarrow \left(\boldsymbol{\Psi}^T \boldsymbol{\Psi} + \lambda \boldsymbol{I}\right)^{-1} \boldsymbol{\Psi}^T \boldsymbol{R}$ ▷ Baseline

  for $i \in \{1, \ldots, N\}$ **do**

    $\hat{A}(\boldsymbol{s}_i, \boldsymbol{\theta}_i) \leftarrow R(\boldsymbol{s}_i, \boldsymbol{\theta}_i) - \boldsymbol{B}^{tT} \psi(\boldsymbol{s}_i)$

  end for

  Order $\left[\left(\boldsymbol{s}_1, \boldsymbol{\theta}_1, \hat{A}(\boldsymbol{s}_1, \boldsymbol{\theta}_1)\right), \ldots\right]$ descending by advantage values $\hat{A}(\boldsymbol{s}_i, \boldsymbol{\theta}_i)$

  $\boldsymbol{W}^{t+1} \leftarrow \left(\boldsymbol{\Phi}^T \boldsymbol{D} \boldsymbol{\Phi} + \lambda \boldsymbol{I}\right)^{-1} \boldsymbol{\Phi}^T \boldsymbol{D} \boldsymbol{\Theta}$

  $\hat{\boldsymbol{\phi}} = \frac{1}{N} \sum_{i=1}^{N} \phi(\boldsymbol{s}_i)$; $\boldsymbol{y} = \frac{\boldsymbol{W}^{t+1}\hat{\boldsymbol{\phi}} - \boldsymbol{W}^t \hat{\boldsymbol{\phi}}}{\sigma^t}$ ▷ σ-normalized step

  $\boldsymbol{p}_\sigma^{t+1} \leftarrow (1 - c_\sigma)\, \boldsymbol{p}_\sigma^t + \sqrt{c_\sigma(2 - c_\sigma)\mu_{eff}}\left(\boldsymbol{\Sigma}^t\right)^{-\frac{1}{2}} \boldsymbol{y}$ ▷ Step size evolution path

  $h_\sigma \leftarrow \begin{cases} 1 & \text{if } \frac{||\boldsymbol{p}_\sigma^{t+1}||^2}{n\sqrt{1-(1-c_\sigma)^{2t}}} < 2 + \frac{4}{n+1} \\ 0 & \text{otherwise} \end{cases}$

  $\boldsymbol{p}_c^{t+1} \leftarrow (1 - c_c)\, \boldsymbol{p}_c^t + h_\sigma \sqrt{c_c(2 - c_c)\mu_{eff}}\, \boldsymbol{y}$ ▷ Covariance evolution path

  $c_{1a} \leftarrow c_1\left(1 - (1 - h_\sigma) c_c(2 - c_c)\right)$

  $\boldsymbol{S} \leftarrow \sum_{i=1}^{N} \left(\boldsymbol{\theta}_i - \boldsymbol{W}^t \phi(\boldsymbol{s}_i)\right) \frac{\boldsymbol{D}_{ii}}{\sigma^{t\,2}} \left(\boldsymbol{\theta}_i - \boldsymbol{W}^t \phi(\boldsymbol{s}_i)\right)^T$

  $\boldsymbol{\Sigma}^{t+1} \leftarrow (1 - c_{1a} - c_\mu)\boldsymbol{\Sigma}^t + c_1 \boldsymbol{p}_c^{t+1} \boldsymbol{p}_c^{t+1\,T} + c_\mu \boldsymbol{S}$

  $\sigma^{t+1} \leftarrow \sigma^t \exp\left(\frac{c_\sigma}{d_\sigma}\left(\frac{||\boldsymbol{p}_\sigma^{t+1}||}{\mathbb{E}||\mathcal{N}(0, \boldsymbol{I})||} - 1\right)\right)$

  $t \leftarrow t + 1$

end while

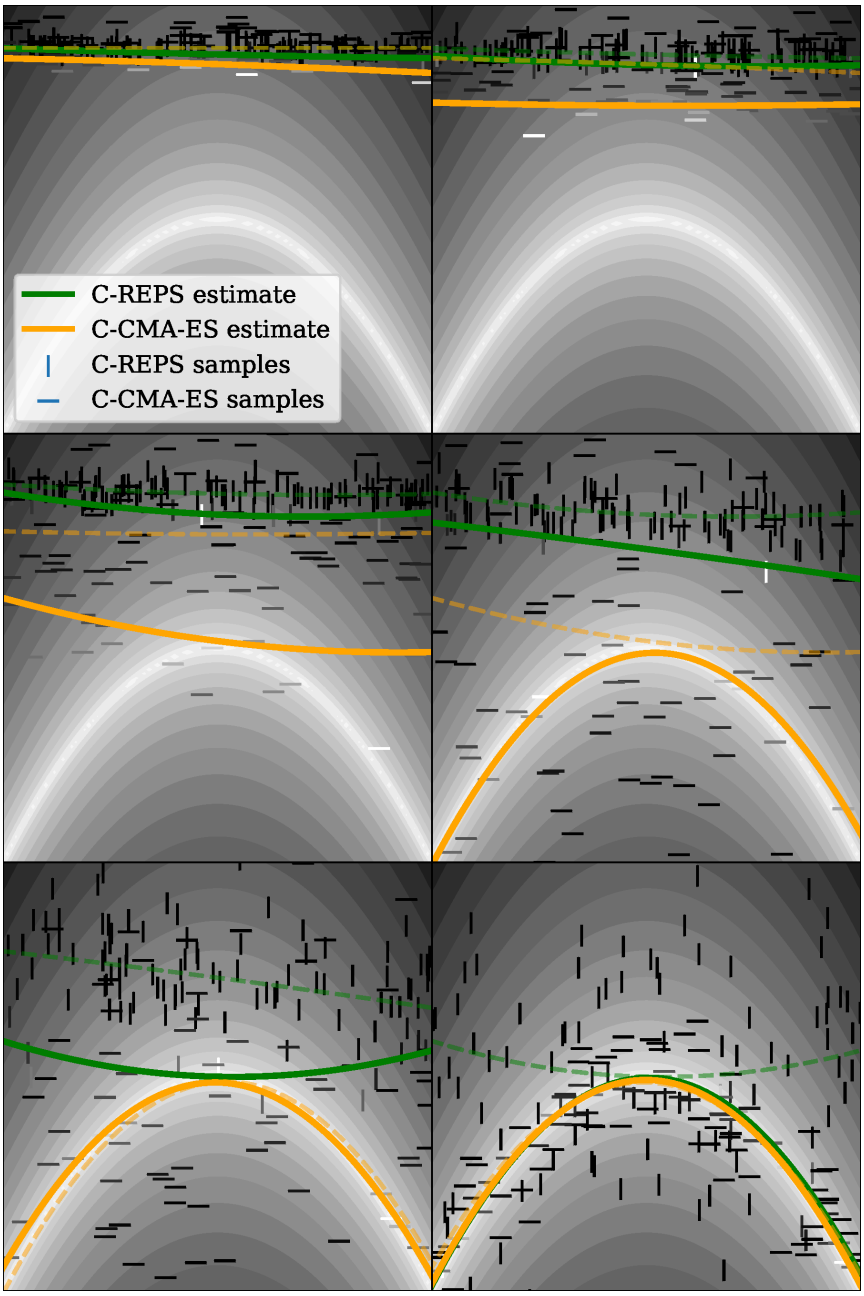


Figure 2.15.: Comparison of C-REPS and C-CMA-ES in a simple contextual problem in six generations. Background color indicates objective values. The optimum is a quadratic function in the valley. The x-axis represents context and the y-axis parameter. 100 samples per generation are used to move the search distribution's mean function from dashed to solid lines.

of the algorithm, which we do in Algorithm 3. Abdolmaleki et al. [Abd+17a] provide a more detailed explanation of the algorithm.

Note that the number of samples for an update, which is also called population size, is written as N here, but the literature about CMA-ES uses λ. As we do not want to confuse it with the regularization parameter λ we will call it N.

A main difference to CMA-ES is that C-CMA-ES uses a context-dependent baseline (line 17) to compute the advantage (lines 18–20) and determine the order of samples (line 21). Furthermore, similar to C-REPS it performs weighted regression to estimate the weight matrix (line 22), although the covariance matrix update is more complex.

The weight computation (line 1) is from CMA-ES. C-CMA-ES, like CMA-ES, separates the step size σ from the covariance. It estimates the evolution path (lines 24–26), which is an approximation of the direction of previous update steps, to adapt the step size (line 30) and obtain an additional rank-1 update of the covariance. Hence, three ingredients are used for the covariance matrix update in C-CMA-ES: the rank-1 update based on the evolution path, a weighted maximum likelihood update like in C-REPS, and the old covariance matrix is not immediately forgotten but decayed exponentially (lines 27–29). These update steps are done with step-size normalized exploration noise.

Figure 2.15 illustrates how C-CMA-ES compares with C-REPS in a simple contextual optimization problem. The initial variance of the search distribution was set intentionally low to demonstrate that C-CMA-ES quickly adapts its step size, whereas C-REPS restricts the maximum Kullback-Leibler divergence between successive search distributions which results in slow adaptation.

2.3.6. Benchmarks

Episodic policy search and black-box optimization are similar. In the black-box optimization community several benchmarks have been proposed. The most important benchmark suite is Comparing Continuous Optimisers (COCO) [Han+16; Fin+19], which provides 24 noiseless objective functions that can be used to analyze the behavior of black-box optimizers in a controlled setting. Those objective functions include five separable functions, four functions with moderate conditioning, five functions with high conditioning, five multi-modal functions with global adequate structure, and five multi-modal functions with weak global structure. They can be instantiated for arbitrary parameter dimensions greater than 1. Two examples with two parameters are shown in Figure 2.16.

As we discussed previously, we can extend black-box optimization with context parameters, that is, we can optimize $\arg\min_{\boldsymbol{\omega}} \int_{\mathbb{S}} f_{\boldsymbol{s}}(g_{\boldsymbol{\omega}}(\boldsymbol{s}))d\boldsymbol{s}$ with respect to parameters of an upper-level policy $g_{\boldsymbol{\omega}}$ that maps the context to the optimum of the parameterized objective function $f_{\boldsymbol{s}}$. If we want to transfer the idea of benchmark functions to contextual function optimization, we have to define parameterized objective functions $f_{\boldsymbol{s}}$. In this thesis, we will modify existing benchmark functions from COCO. We apply a context-based transformation h of the parameter vector, that is, $h(\boldsymbol{s}, \boldsymbol{\theta}) = \boldsymbol{\theta}'$, before it is given to the original objective function f, hence, the contextual benchmark function is $f_{\boldsymbol{s}}(\boldsymbol{\theta}) = f(h(\boldsymbol{s}, \boldsymbol{\theta}))$. This approach has been later on used by Abdolmaleki et al. [Abd+17a] as well. Specifically Abdolmaleki et al. [Abd+17a] proposed $f_{\boldsymbol{s}}(\boldsymbol{\theta}) = f(\boldsymbol{\theta}+\boldsymbol{G}\boldsymbol{s})$, with a matrix $\boldsymbol{G}$ of components that are sampled independent and identically distributed

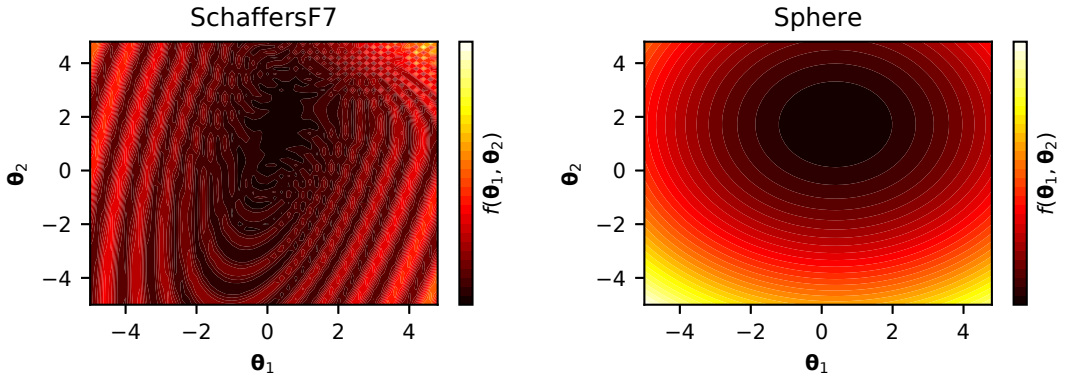


Figure 2.16.: Two object functions from the COCO benchmark: Schaffer's F7 and Sphere.

(iid) from a standard normal distribution. We can extend this to $f_{\boldsymbol{s}}(\boldsymbol{\theta}) = f(\boldsymbol{G}\boldsymbol{\phi}(\boldsymbol{s}))$, with a transformation $\boldsymbol{\phi}$ of the context vector. If $\boldsymbol{\phi}\left((s_1, s_2)^T\right) = \left(1, s_1, s_2, s_1 s_2, s_1^2, s_2^2\right)^T$, we would be able to represent any quadratic function of $\boldsymbol{s} = (s_1, s_2)^T$. The effect of the transformation h is to make the optimum of the objective context-dependent. The contextual optimizer has to compensate for h while it samples in individual contexts.

2.3.7. Discussion

We have seen that there are strong connections between black-box optimization and policy search. C-REPS has been the most important contextual policy search algorithm for several years; however, it has been shown that it suffers from premature convergence [Abd+17b]. Recently, novel approaches have been proposed: C-CMA-ES, C-MORE, and BO-CPS can be considered state of the art in contextual policy search.

C-CMA-ES is similar to C-REPS and it is computationally more efficient than BO-CPS with respect to the number of parameters that have to be optimized. For high-dimensional, redundant context vectors such as camera images C-MORE is particularly effective because it uses a dimensionality reduction method. Since we are more interested in low-dimensional context vectors in this thesis, we will mostly focus on C-REPS and C-CMA-ES. As most contextual policy search algorithms are local search approaches, they need a good initial search distribution, which is often obtained by imitation learning.

BO-CPS is different, as it is a global optimizer and is only efficient enough for a small number of control policy parameters. It is the most sample-efficient algorithm though. BO-CPS will have an important role in this thesis and will be presented in detail in Section 4.4.

2.4. Summary

We give a broad overview of behavior learning methods and problems in robotics. Although there are several survey papers on various aspects, such a complete and broad overview is new.

We identify policy gradient algorithms for neural networks as a promising field of research, especially for high-dimensional sensor data, but regard current approaches as too brittle and difficult to tune. Policy search with movement primitives is a field with a lot of potential for robust and sample-efficient methods that can be initialized from imitation of human demonstration. These methods will form the basis for the next chapters. Natural and highly optimized demonstrations from humans can be obtained best from motion capture data, however, the embodiment mapping has to be defined. Contextual policy search is a way to generalize behaviors over several task parameters or the context. It is, however, either not sample efficient enough or not applicable to complex movements. We have to address this issue.

Related Publications

[Fab+20] Alexander Fabisch, Christoph Petzoldt, Marc Otto, and Frank Kirchner. "A Survey of Behavior Learning Applications in Robotics—State of the Art and Perspectives". In: *International Journal of Robotics Research* (2020). Submitted.

[Fab19] Alexander Fabisch. "Empirical Evaluation of Contextual Policy Search with a Comparison-based Surrogate Model and Active Covariance Matrix Adaptation". In: *Genetic and Evolutionary Computation Conference Companion*. Ed. by Manuel López-Ibáñez. GECCO '19. ACM, 2019, pp. 251–252. ISBN: 978-1-4503-6748-6. DOI: 10.1145/3319619.3321935.

[FM14] Alexander Fabisch and Jan Hendrik Metzen. "Active Contextual Policy Search". In: *Journal of Machine Learning Research* 15 (2014), pp. 3371–3399. URL: http://jmlr.org/papers/v15/fabisch14a.html.

The overview of behavior learning problems is based on Fabisch et al. [Fab+20]. While the complete publication is a joint work with the co-authors, the analysis that is included in this thesis (Chapters 1, 2, and 8) is my contribution. Exceptions are marked accordingly: Figure 2.4 and Appendix A.

The description of the algorithm C-REPS has been extracted from Fabisch and Metzen [FM14] and the description of C-CMA-ES from Fabisch [Fab19].

Part II.

Enhanced Methods for Robot Behavior Learning

Chapter 3.

Imitation with Automatic Embodiment Mapping

In this chapter, we will discuss imitation of human motion with a focus on transferring state space trajectories to robotic systems. Demonstrations from humans are recorded with a motion capture system, hence, we cannot directly map them to robot commands because of the correspondence problem (see Section 2.2.1.2). Therefore, an automated approach to define the embodiment mapping is required. We propose a procedure that consists of the following steps:

- global trajectory optimization: black-box optimization to determine synchronization frames for the whole trajectory,
- local pose optimization: approximation of inverse kinematics,
- spatial and temporal scaling to take the kinematic and dynamic capabilities of the target system into account,
- and refinement with policy search.

The first three steps alone will only guarantee that the actions are executable on the target system. They are agnostic to the task, which means the actions do not necessarily produce the same effects that can be observed in the demonstration or might not solve the task. The last step is task-specific and will ensure that the task will be solved. Not all of these methods are required for all categories of tasks. We will discuss and evaluate these methods in this chapter.

3.1. Task-Agnostic Embodiment Mapping

Parts of this section were published originally as [Gut+18; Gut+19; Fab20] and have been revised.

The most important task-agnostic parts of the automatic embodiment mapping are a trajectory optimization and a pose optimization. We will discuss and evaluate these.

3.1.1. Global Trajectory Optimization

We observe sequences of end effector poses (trajectories) from a human teacher with a motion capture system. Thus, an embodiment mapping has to be obtained that maps

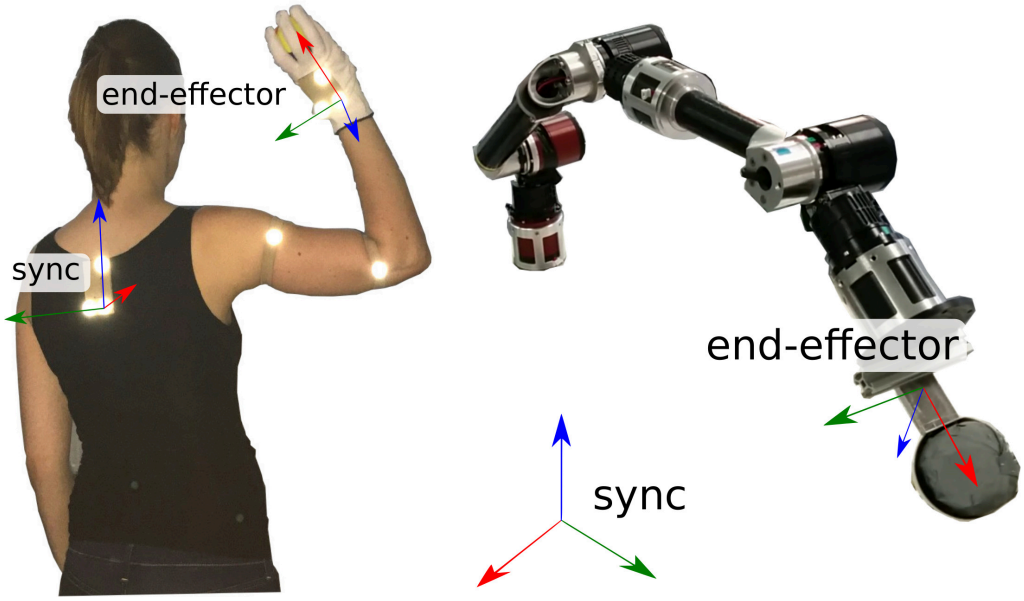


Figure 3.1.: Synchronization frames on the human teacher and on the robot.

these trajectories to the workspace of the robot so that they are reachable and there are no discontinuities in joint space.

As a first step, we define so-called synchronization frames (see Bongardt [Bon15] for a thorough mathematical introduction) that exist in the teacher's workspace and in the robot's workspace. A synchronization frame is a reference frame, in which a demonstration is recorded (teacher) or reproduced (robot) so that we can directly transfer demonstrations to the corresponding synchronization frame in the robot's workspace. See Figure 3.1 for an illustration of a ball throwing example.

In some situations it is obvious how the corresponding synchronization frame of the robot should be selected to transfer demonstrated trajectories in Cartesian space. For example, if we define the teacher's reference frame to be the target of a grasping movement, we will not change this for the robot. In other cases it is not obvious. Consider a ball-throwing movement where the reference frame of the recorded movement is the human teacher's back (see Figure 3.1). When we want to transfer the observed trajectory to a robotic arm, it is not obvious where we would put the synchronization frame.

We can choose it arbitrarily and use this freedom to account already for some of the problems caused by the correspondence problem without any information about the task. We use optimization to do this. For simplicity, we assume that both synchronization frames are constant over time, so that we effectively shift the whole trajectory globally to compensate for some differences in the structure of the human teacher and the robotic learner.

More precisely, we optimize a parameterized linear mapping of the form

$$g^E(x_t, y_t, z_t, \alpha_t, \beta_t, \gamma_t) = \begin{pmatrix} \boldsymbol{R}_{\alpha_s,\beta_s,\gamma_s} \left((1-s) \begin{pmatrix} x_0 \\ y_0 \\ z_0 \end{pmatrix} + s \begin{pmatrix} x_t \\ y_t \\ z_t \end{pmatrix} \right) + \begin{pmatrix} x_s \\ y_s \\ z_s \end{pmatrix} \\ \alpha_t + \alpha_s \\ \beta_t + \beta_s \\ \gamma_t + \gamma_s \end{pmatrix},$$

to transfer a pose $(x_t, y_t, z_t, \alpha_t, \beta_t, \gamma_t)$ given by a position and Euler angles (intrinsic rotations around x-, y'- and z"-axis) in a sequence of poses $t \in \{0, \dots, T\}$ from the synchronization frame of a human demonstration to the origin of the robotic target system. We also use a spatial scaling factor $s \in (0, 1]$ to more easily fit the trajectory into the robot's workspace, $\boldsymbol{\theta} = \alpha_s, \beta_s, \gamma_s$ are Euler angles that define a rotation, and $\boldsymbol{b}_s = (x_s, y_s, z_s)$ is a translation. $s, \boldsymbol{\theta}, \boldsymbol{b}$ will be selected to maximize the objective

$$\begin{aligned} f(s, \boldsymbol{\theta}, \boldsymbol{b}) = & \exp\left(\frac{10}{T+1} \sum_t r(g^E(\boldsymbol{p}_t)) \right) && \text{(reachability)} \\ & - w_{vel} \sum_t \dot{\boldsymbol{q}}(g^E(\boldsymbol{p}_t)) && \text{(velocities)} \\ & - w_{acc} \sum_t \ddot{\boldsymbol{q}}(g^E(\boldsymbol{p}_t)) && \text{(accelerations)} \\ & - w_{jrk} \sum_t \dddot{\boldsymbol{q}}(g^E(\boldsymbol{p}_t)) && \text{(jerks)} \\ & - w_{coll} \sum_t c(\boldsymbol{q}_t) && \text{(self-collisions)} \\ & - w_{dist} ||\boldsymbol{p}_T|| && \text{(displacement)} \\ & + w_{height} \sum_t \boldsymbol{p}_{3,t} && \text{(height)} \\ & + w_{size} \sum_{d=1}^{3} \max_t \boldsymbol{p}_{d,t} - \min_t \boldsymbol{p}_{d,t}, && \text{(spatial extent)} \end{aligned}$$

where $t \in \{0, \dots, T\}$ is the time step, $r(\boldsymbol{p})$ is 1 if $\boldsymbol{p}$ is a reachable end-effector pose and 0 otherwise, $c(\boldsymbol{q})$ is 1 if the configuration results in self-collision and 0 otherwise, $\boldsymbol{p}_t$ is an end effector pose and $\boldsymbol{q}_t$ are corresponding joint angles at step t. The objective maximizes reachability, while minimizing the risk of getting too close to singularities, avoiding self-collisions, and maximizing exploitation of the robot's workspace. To maximize f any black-box optimizer can be used. We decided to use CMA-ES [HO01] for the following evaluation. We could also use gradient-based optimizers such as L-BFGS [Noc80] with finite differences. The weights of the objective have to be configured appropriately.

3.1.2. Local Pose Optimization with Approximate Inverse Kinematics

We will use an approximation to the numerical solution of inverse kinematics if it is difficult to find a mapping that fits the trajectory into the workspace of the robot without changing its global structure.

3.1.2.1. Related Work: Inverse Kinematics

Let $\boldsymbol{q}_t$ be the joint angles of a kinematic chain at time t. The forward kinematics of the chain is given by $f(\boldsymbol{q}_t) = \boldsymbol{p}_t$, where $\boldsymbol{p}_t$ denotes the end effector's pose. An exact solution to the inverse kinematics (IK) problem would be $f^{-1}(\boldsymbol{p}_t) = \boldsymbol{q}_t$; however, f^{-1} is often not a function, as more than one joint configuration can result in the same end-effector pose.

Numerical solutions to inverse kinematics efficiently handle kinematic chains that have many solutions or no analytical solution. Widely used approaches are based on the Jacobian pseudo-inverse or Jacobian transpose [Nil09], although sequential quadratic programming has been shown to outperform these with regard to joint limit handling and computation time [BA15]. Indirect formulations of the form

$$\arg\min_{\boldsymbol{q}_t \in \mathbb{R}^n} (\boldsymbol{q}_{t-1} - \boldsymbol{q}_t)^T (\boldsymbol{q}_{t-1} - \boldsymbol{q}_t), \quad \text{s.t. } g_i(\boldsymbol{q}_t) \leq b_i, \quad i = 1, \ldots, m$$

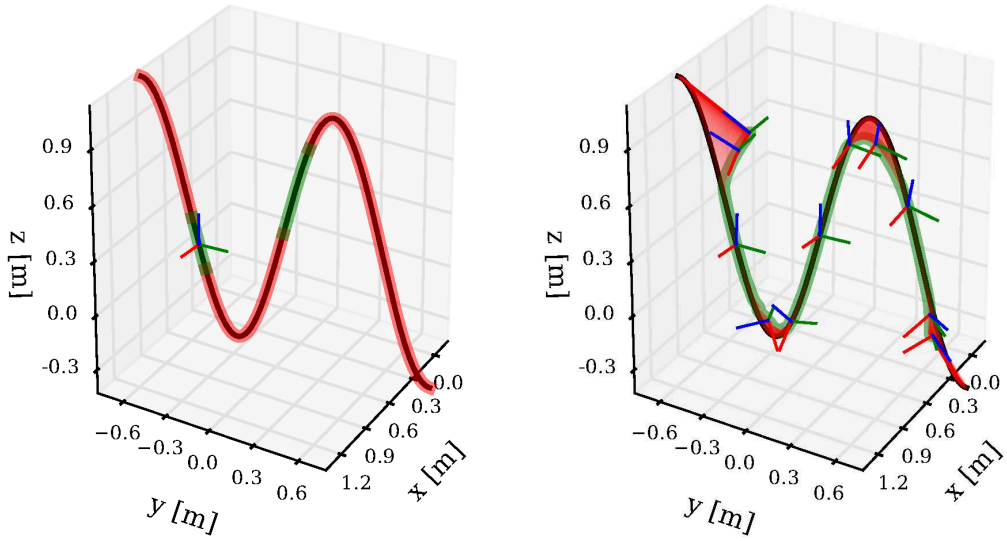
where the constraints defined by g_i, b_i include the Euclidean distance error, the angular distance error, and joint limits (for example, Kumar et al. [KSB10] and Fallon et al. [Fal+15]), have a lower success rate than direct formulations of the form

$$\arg\min_{\boldsymbol{q}_t \in \mathbb{R}^n} \quad d(f(\boldsymbol{q}_t), \boldsymbol{p}_t^{\text{des}})) \tag{3.1}$$

$$s.t. \quad g_i(\boldsymbol{q}_t) \leq b_i, \quad i = 1, \ldots, m \tag{3.2}$$

where d is a pose distance metric, $\boldsymbol{p}_t^{\text{des}}$ is the desired end-effector pose, and the inequality constraints only consist of the joint limits [BA15].

Few works investigate approximate IK solutions; however, when the desired end-effector pose cannot be reached exactly, we should use at least the closest possible solution. Unreachable poses might be a problem of robots that have a low number of joints [Hen14] or at the borders of workspaces as this can be seen in visualizations of capability maps [ZBH07]. Traditional methods based on the Jacobian pseudoinverse tend to be unstable and run into local minima [Nil09; Hen14] here. Rakita et al. [RMG18] develop RelaxedIK, which approximates solutions to the inverse kinematics problem but also generates smooth joint space trajectories, avoids self-collisions, and avoids singularities. Those are typical features of an offline planner and are now solved online with a sequential quadratic programming algorithm. RelaxedIK can be regarded as the link between conventional inverse kinematics solvers and whole-body control [SK06], which optimizes multiple arbitrary objectives for a complex robotic system online. RelaxedIK is similar to our solution, which has been developed independently in 2016 before RelaxedIK has been published.



(a) Exact inverse kinematics. The end effector of a Kuka iiwa should follow a sine curve with a fixed rotation (indicated by small coordinate frame). Most parts of the trajectory are not exactly reachable and marked by a red line. Reachable poses are indicated by a green line.

(b) Approximation of inverse kinematics. The desired trajectory is marked by the black line and its approximation with numerical inverse kinematics by the green line with small coordinate frames indicating orientations. We use the weight 0.001 for the orientation and 1 for the position. Deviations from the desired positions are marked by red areas.

Figure 3.2.: Comparison of exact inverse kinematics and an approximation. The desired trajectory is close to the border of the robot's workspace.

3.1.2.2. Configurable Approximate Inverse Kinematics

A disadvantage of conventional inverse kinematics solvers is that they do not allow to configure the approximation of a pose. In behavior learning it is often not required to reach each pose in a trajectory exactly. Consider the problem of learning to grasp an object. Orientation of the end effector is not relevant at the beginning of a reaching movement. It only becomes important at the end. Therefore, we develop a configurable IK solver based on the work of Beeson and Ames [BA15].

The approach is illustrated in Figure 3.2. In Equation 3.1 we use the distance metric

$$\begin{aligned} d(\boldsymbol{p}_1, \boldsymbol{p}_2) &= w_{\text{pos}} ||\boldsymbol{p}_{1,1:3} - \boldsymbol{p}_{2,1:3}||_2^2 \\ &\quad + w_{\text{rot}} \left[\min(||\ln(\boldsymbol{p}_{1,3:7} * \overline{\boldsymbol{p}}_{2,3:7})||_2, 2\pi - ||\ln(\boldsymbol{p}_{1,3:7} * \overline{\boldsymbol{p}}_{2,3:7})||_2)\right]^2, \end{aligned}$$

where $\boldsymbol{p}_{1:3}$ represents the position of the end effector and $\boldsymbol{p}_{3:7}$ is a quaternion that represents the orientation. ln is the logarithmic map. The metric consists of a position

distance and a rotation distance. We could also use other orientation distances [Huy09]. This allows us to set different weights on position and rotation because it is often much more important to reach a desired position than the desired orientation. We could extend this pose metric so that we can weight each of the six degrees of freedom individually.

3.1.3. Evaluation of Task-Agnostic Embodiment Mapping

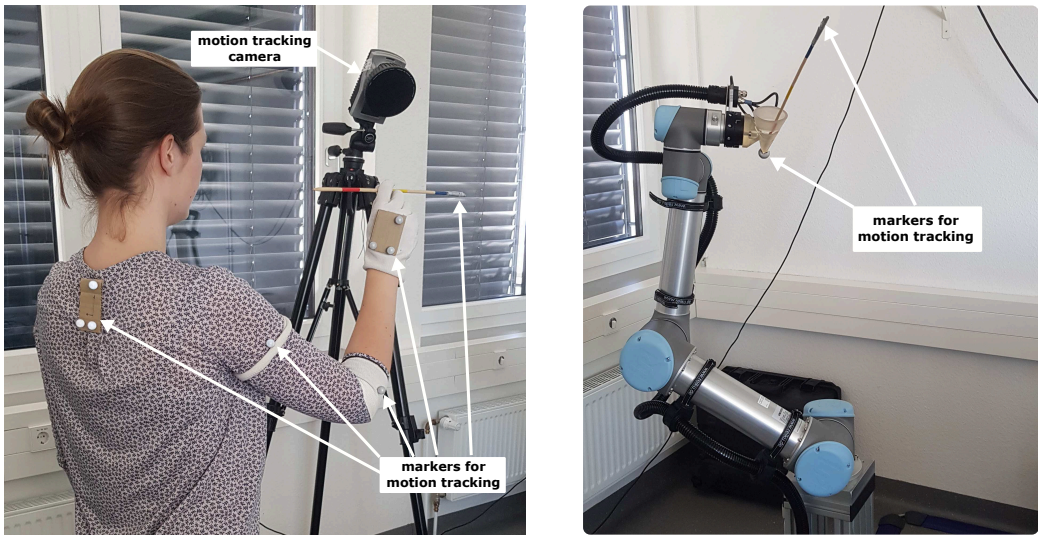
In this section, we evaluate the proposed task-agnostic part of the embodiment mapping with Touhu, which is a throwing game that is traditionally played in East Asia. The goal is to throw a stick from a given distance into a pot. We use this scenario to evaluate the transferability of throwing movements to different robotic systems using their kinematic models. We use a large dataset of demonstrated throws to find out how many throws are transferable, if they are distorted a lot, and what the differences between various robotic systems are. Additionally, the quality of the transferred movements is evaluated in a second experiment, where the trajectory and goal position of the stick thrown by humans is compared with the one thrown by a real UR5 robot. Finally, we want to answer the question how far we get with a task-agnostic embodiment mapping and whether task-specific refinement is required.

3.1.3.1. Experimental Setup

Throwing motions that are demonstrated by human subjects are recorded with a marker-based motion capture system [Qua20a]. Markers are attached to the hand, elbow, shoulder, and back of each subject to track these positions. We use three markers to determine the orientation of the back and the hand. The complete marker setup can be seen in Figure 3.3.

In a second experiment, in addition to the human movement, we track the position of the stick. The trajectories of the thrown stick from the demonstration can be compared with the resulting trajectories after imitation of the throwing motion on the real system. Thus, a marker is placed on one end of the stick. Additionally, the movement of the robotic arm UR5 is captured by placing a marker at the end effector (see Figure 3.3).

We extract the relevant throwing movements automatically from the recorded motions of the human (see Chapter 5 for more details). To transfer the recorded demonstrations to the four robotic systems, the embodiment mapping has been optimized with the following weights in the objective function: $w_{coll} = 100, w_{vel} = 100, w_{acc} = 10, w_{jrk} = 1, w_{dist} = 0.1, w_{height} = 50, w_{size} = 100$. These weights have been determined empirically. The optimization was limited to the Cartesian translation of the trajectory within the workspace of the robot. Orientation and scaling remained unchanged. In addition, the resulting trajectory is smoothed with a mean filter in Cartesian space and in joint space (positions and accelerations) to avoid high accelerations.



(a) Setup to record the human arm motion with markers attached to the back, arm, and hand.

(b) A marker is attached to the stick to record its position. UR5 is tracked with a marker at the end effector.

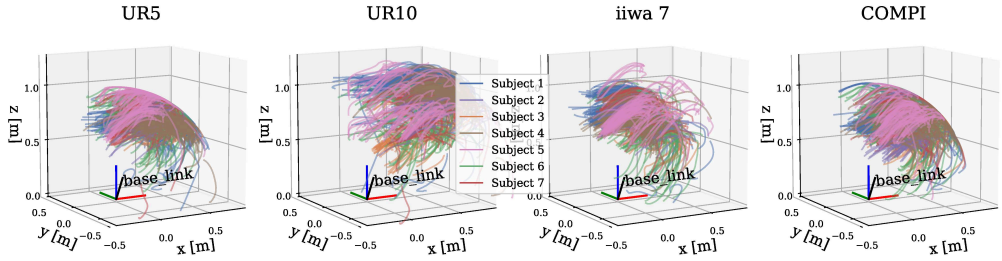
Figure 3.3.: Motion capture setup. (Illustrated by Lisa Gutzeit [Gut+19].)

3.1.3.2. Transfer in Simulation

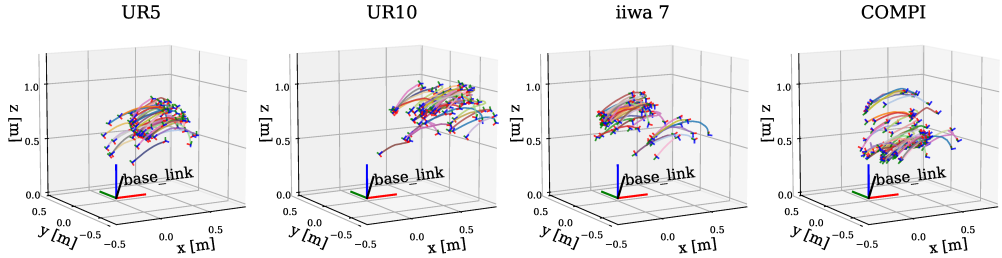
Throwing motions of seven different subjects, each performing between 41 and 246 throws, were recorded to evaluate the generality of our approach. In total, 697 Touhu demonstrations were recorded. With such a large dataset, we are able to evaluate the generality of the movement segmentation as well as the transfer to different robotic systems, namely Universal Robots' UR5 and UR10, Kuka iiwa 7 and DFKI's COMPI (see Appendix F for details).

The throwing trajectories mapped into the workspace of the robotic system UR5, UR10, Kuka iiwa and COMPI, are shown in Figures 3.4a and 3.4b. 682 out of 697 trajectories were transferred to the workspaces of all target systems. We can see that most trajectories easily fit in the workspace of UR10 (arm radius: 1300 mm) and KUKA iiwa (arm radius: 1266 mm), while many trajectories have to be distorted or are close to the borders of the workspace of UR5 (arm radius: 850 mm) and COMPI (arm radius: 940 mm). Throwing movements often tend to be close to the borders of the human's workspace. Hence, the skill that we selected is quite challenging for smaller robots. Figure 3.4b shows the demonstration of subject 5 transferred to each target system.

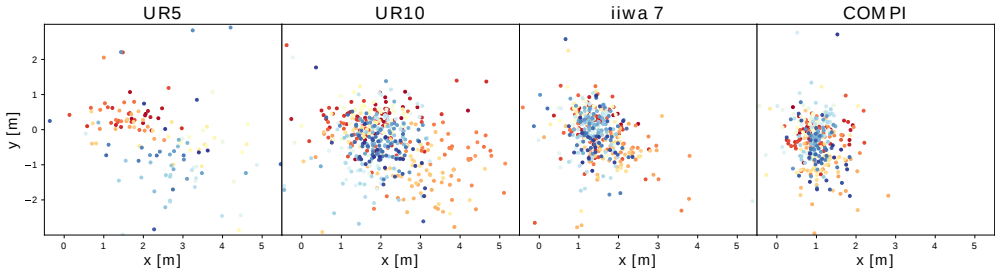
3.4c shows the ground contact points of sticks for the presented throwing trajectories from simulation. We see that the UR10 has the widest distribution as it has the largest workspace. We will quantify how well the throwing trajectories can be transferred to the real UR5 robot with the next experiment, as it is one of the robotic systems with a more restricted workspace.



(a) All recognized throwing movements transferred to the workspace of the four robots. Trajectories from the same subject are shown in the same color.



(b) All throwing movements of subject 5. Colors indicate the indices of the throws. The frames of the robots' base links are shown.

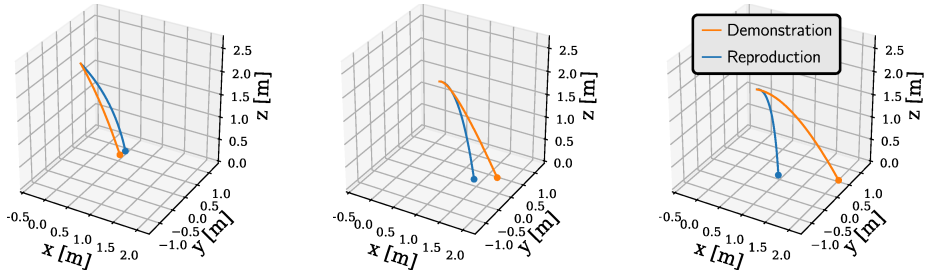


(c) Throwing results in simulation. We display the distribution of ground contact points of the sticks. Colors of the points indicate the index of the transferred throwing trajectory. Thus, similar colors most likely correspond to the same subject.

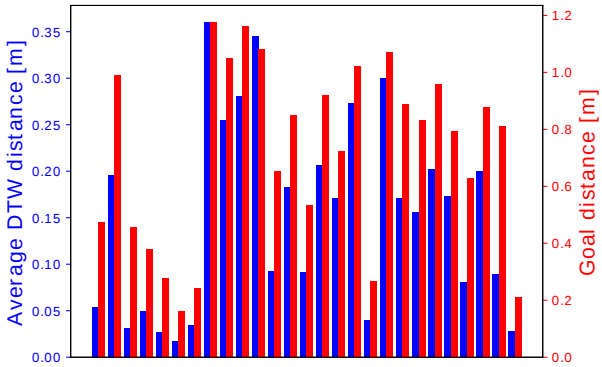
Figure 3.4.: End-effector trajectories of throwing movements in robots' workspaces and corresponding ground contact points of the sticks.

3.1.3.3. Transfer to a Real System

We additionally recorded 34 throwing movements of three subjects, in which also the position of the stick was tracked. Three subjects performed 10, 11 and 13 throws respectively. These demonstrations were transferred to the real UR5 robotic arm. We analyzed the number of successful throws, the stick position during the throw and its goal position. A demonstrated throw on the UR5 robot is evaluated as transferable if robot and stick do not collide with anything, the stick does not fall out of the stick holder while the robot approaches the starting pose, and the stick leaves the holder during



(a) 3D plot of stick trajectories of the best (left), a good (middle), and the worst (right) result. The orange trajectory indicates how the stick was thrown in the demonstration and the blue trajectory is the reproduction by the UR5.



(b) Average dynamic time warping distances and distances of goal positions.

Figure 3.5.: Analysis of the execution of throws on the real UR5.

the throwing motion. To evaluate the quality of the transferred throws, we compare the stick trajectories of the demonstrated throws and the recordings of the throws transferred to the UR5. Since the motion capture system sometimes returned noisy stick position measurements, the trajectories had to be interpolated. A quadratic model was used for interpolation. The same model has been used to extrapolate both the demonstrated and the reproduced stick trajectory until the stick hit the ground. Before this, we aligned the demonstrated trajectory with the start position of the transferred one. Thus, the distance between ground contact points as well as the similarity of the stick trajectories can be determined. We use the average dynamic time warping (DTW) [SC78] distance, that is, the DTW distance divided by the maximum number of steps of the two time series, to compare the trajectories.

27 out of 33 throws could be transferred to the real UR5. A comparison of the stick trajectories can be seen in 3.5a, in which the best, a good, and the worst result are visualized. The mean average DTW distance is 0.15 m (standard deviation: 0.1 m), and the mean goal distance is 0.72 m (standard deviation: 0.31 m). The full error distribution is shown in 3.5. The results show, that it is possible to automatically imitate demonstrated throws and that most of these throws are executable on the real system. Furthermore,

the executed movements show useful throws. Despite everything, the goal positions of the demonstrated throws are not reached by the system.

3.1.4. Discussion

Throwing trajectories are automatically extracted from human demonstrations, and transferred to four robotic target systems. We show that the embodiment mapping, which is needed to map human movement trajectories into the robot's workspace, can be automized for a dataset of 697 throws. Throwing is a challenging skill for these robots because it has high accelerations and velocities and is close to the border of the workspace of humans. Nonetheless, most of the demonstrated throws could be transferred to the systems using our approach. Furthermore, we evaluate the difference of stick trajectories and ground contact points between demonstrated throws and reproductions of those on a real UR5. Although the demonstrated throwing motions could be successfully executed on different systems, there is still a considerable gap between the outcome of demonstrated throws and their reproductions.

Furthermore, this approach does not work well for arbitrary types of robot skills. To transform demonstrated trajectories to the robot's workspace, the trajectories sometimes have to be modified. Long trajectories may have to be scaled down to completely fit into the workspace. Fast trajectories may have to be slowed down to be executable by the system. Often trajectories have to be translated with respect to the synchronization frame in the workspace of the robot. This conflicts with trajectories that require that certain poses, such as via points and goals, are reached. For this kind of task it is required that these particular poses are reachable by the end effector, relative to given reference frames. These constraints are not integrated in the objective of our embodiment mapping and will be completely ignored. Thus, even though the human may have demonstrated a successful reaching behavior, the robot might not be able to solve the same task because the trajectory has been shifted by the embodiment mapping. The goal of our embodiment mapping is to generate a good initial motion that is executable and can later be refined for a specific task and target system.

Similar work on automating the embodiment mapping has been presented by Maeda et al. [Mae+16]. Our approach to this problem has been developed before this work has been published. It is is more restricted, as we do not integrate task-specific constraints in the optimization process (for example, collision penalties for external objects or via points), but this also is the benefit of our approach as it is more modular, which allows us to use any standard black-box optimization method to optimize the embodiment and do task-specific adaptation with standard reinforcement learning algorithms. Furthermore, no model of the environment is needed at this stage. The only prior knowledge that is needed is a kinematic model of the robot. In this section we examined if this restricted approach already generates useful trajectories in the workspace of the robot. The main focus lies on the automation of the embodiment mapping to map human movement recordings to a trajectory which is executable on the system.

3.2. Task-Specific Policy Refinement

This section was published originally as [Fab20] and has been revised.

Until now we only used knowledge about our target system. To ensure that the imitated skill has the same effects as the demonstration, we must integrate knowledge about the task. This will be done in the policy refinement step.

Here we account for kinematic and dynamic differences that cannot be resolved easily. For instance, a human teacher might have a hand structure that is different from the target system. An example is displayed in Figure 3.1: the target system does not even have an actively controllable hand. It has a scoop mounted on the tip of the arm. In other cases, the robot might have a gripper that does not have all of the capabilities of a human hand. Another problem in the ball-throwing domain is the dynamic and kinematic difference between the human demonstrator and the target system. It might be possible for the robot to execute the throwing movement after temporal scaling, but this step can reduce the velocity in Cartesian space so drastically that the ball does not even leave the scoop.

3.2.1. Policy Search for Policy Refinement

We propose to use policy search methods for task-specific refinement. Therefore, we must define a reward function that can be used by policy search to complete the embodiment mapping.

We will use DMPs as policies at this stage because they are well known and stable trajectory representations that can be used for imitation and policy search. Many policy search algorithms for DMPs are based on similar principles: we learn a Gaussian distribution from which we sample policy parameters. We will use CMA-ES [HO01] or REPS [PMA10] to refine policies. CMA-ES has few critical hyperparameters and is still reliable, when those are not set perfectly. REPS is a bit more difficult to tune but can be similarly sample-efficient.

We are particularly interested in the question whether it is better to do policy search directly in task space because demonstration and main objectives of the learning problem are given in task space, for example, end-effector poses in Cartesian space. We will investigate this empirically in the following experiments. When policy search methods are used to learn end-effector trajectories in task space, it often occurs that a trajectory is not completely in the robot's workspace. The resulting return landscapes are difficult to optimize. We will again address this problem with the approximation of IK.

3.2.2. Related Work: Behavior Learning in Cartesian Space

Examples for skills that have been demonstrated in task space and transferred to robots are pancake flipping (learned from kinesthetic teaching; [KCC10b]), peg in hole (learned from teleoperation; [Krü+14]), and serving water (kinesthetic teaching with a similar arm; [Pas+09]). There are several reasons for learning trajectories in task space. Learning in Cartesian space is often easier when the main objectives are defined in Cartesian space. Learning to grasp an object in joint space might result in complicated Cartesian

trajectories. When we want to transfer a skill from one robot to another it is easier over end-effector poses instead of joint angles because of different kinematic structures [Pas+09], since part of the correspondence problem is solved by forward and inverse kinematics.

3.2.3. Experiments: Refinement in Cartesian Space and Joint Space

We want to answer the questions (1) whether it is easier to refine trajectories in Cartesian space or in joint space and (2) whether it is better to use an approximation of inverse kinematics than only exact solutions for refinement in Cartesian space. The following paragraphs will summarize the results obtained for several problems that have different return surfaces.[1]

3.2.3.1. Methods

The artificial RL problems that we will solve vary in difficulty, that is, indirection and smoothness of the return. We will only learn the weights of the DMPs. Metaparameters are constant. A DMP based on quaternions [Ude+14] will be used to generate end-effector poses.

We will use CMA-ES for policy search. Its most important hyperparameter is the initial step size σ. It determines the width of the initial search distribution. If it is not large enough, the algorithm will first have to increase the step size for several updates before it converges. If it is too large, convergence will take longer. For a comparison it is important to select the correct step size ratio between joint space and Cartesian space so that the results will not be distorted by a wrong choice of this parameter. In these experiments, we use the Kuka iiwa 14 R820 robot arm with 7 DOF (see Appendix F.2), for which we determined empirically that the initial step size in joint space must be 2 to 3 times higher than in Cartesian space to achieve similar end-effector motions.

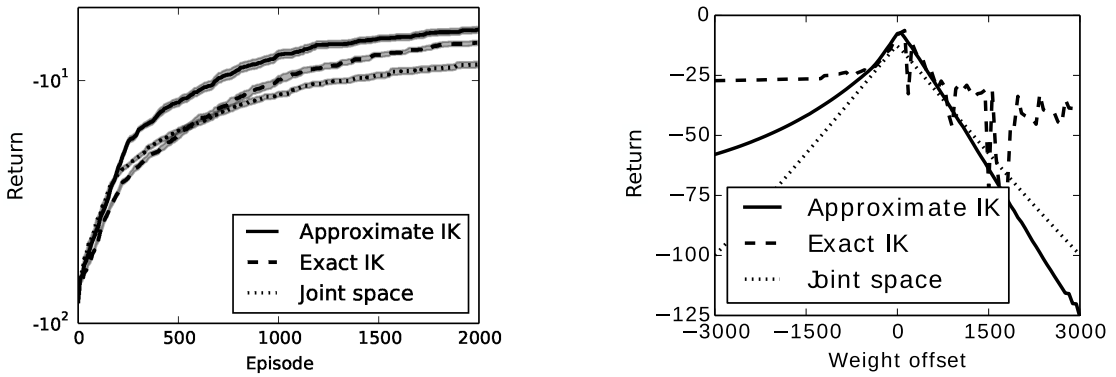
In the first two experiments, the optimum solutions are close to the border of the workspace so that not every orientation can be reached. In all experiments we compare (1) learning weights of DMPs in joint space, (2) in Cartesian space with the proposed approximate IK, and (3) in Cartesian space with an *exact* IK. The term *exact* throughout this section means a numerical IK solver based on the pseudoinverse of the Jacobian that does not move the end effector if no valid solution has been found.

We will execute 30 runs per configuration with different random seeds for CMA-ES. We use the results to plot learning curves with the mean and standard error of the maximum return obtained so far. Each DMP in our setup has 50 weights per joint space or task space dimension.

3.2.3.2. Via Point

In the via-point problem, the end effector should pass through intermediate positions (via points). Simpler via-point problems have been used before to compare various policy search methods [Kob+10; PS06]. In this version we want the end effector to pass through

[1] Implementation is available at `https://github.com/rock-learning/approxik`.



(a) Learning curves for the via point problem. The y-axis is logarithmic.

(b) Projection of the return surface of the via point problem on one axis.

Figure 3.6.: Results of the via-point problem.

five Cartesian points (see Figure 3.7) while minimizing joint velocities and accelerations. The return is

$$R = -10\left(\sum_{(t_\nu,\nu)} ||f(\boldsymbol{q}_{t_\nu}) - \nu||_2\right) - 10^{-3}\left(\sum_{t,j} |\dot{\boldsymbol{q}}_{t,j}|\right) - 10^{-5}\left(\sum_{t,j} |\ddot{\boldsymbol{q}}_{t,j}|\right),$$

where $t \in \{1, \ldots, 101\}$ represents the step, j are joint indices and (t_ν, ν) represents a position ν that has to be reached at time t_ν.

The results are shown in Figure 3.6a. Learning in Cartesian space is more sample-efficient because the primary objective is defined in Cartesian space. The exact IK performs worse because the approximate IK generates more smooth trajectories. Via-point problems belong to the most simple class of problems for policy search methods because

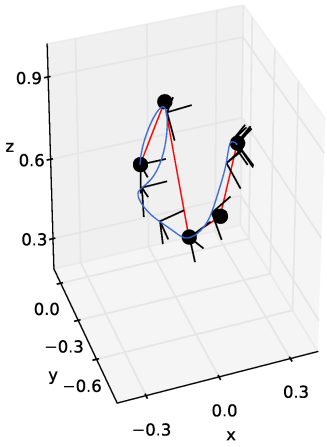
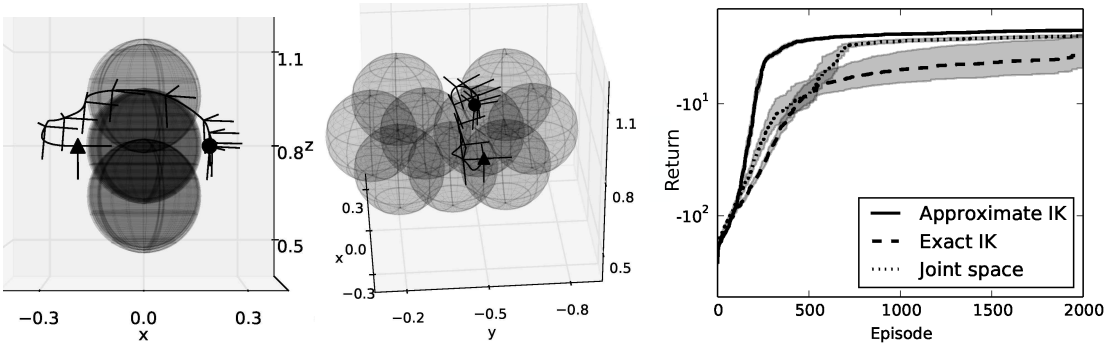


Figure 3.7.: Via point problem: 5 via points (circles) have to be reached. The red line indicates the order. The blue line is a trajectory generated by a DMP with small coordinate frames indicating orientations.



(a) Illustrations of obstacle avoidance problem: ball-shaped obstacles must be avoided between start (triangle) and goal (circle). An example of an optimized trajectory is displayed. The line indicates an optimized trajectory (orientations are indicated by small coordinate frames).

(b) Learning curves for the obstacle avoidance problem. The y-axis is logarithmic.

Figure 3.8.: Obstacle avoidance problem.

there are no flat regions or abrupt changes in the return surface so that it is easy to determine the direction of improvement. We can get a rough impression of the return surface of the problem from Figure 3.6b. To generate a projection of the return surface on one DMP weight dimension, we learn a DMP for 1000 episodes, keep the best policy, modify the 50th weight by adding an offset, and measure the corresponding return. We can see that for both the joint space DMP and the Cartesian DMP with an approximate IK the return surface is smooth with only one maximum. With the exact IK, the return surface is rough and abruptly changing in some regions, which makes learning more difficult for CMA-ES. Learning in joint space, however, is worse than learning in Cartesian space because of the nonlinear mapping from weights to positions in Cartesian space through the forward kinematics of the robot arm. This results in more complex dependencies between parameters.

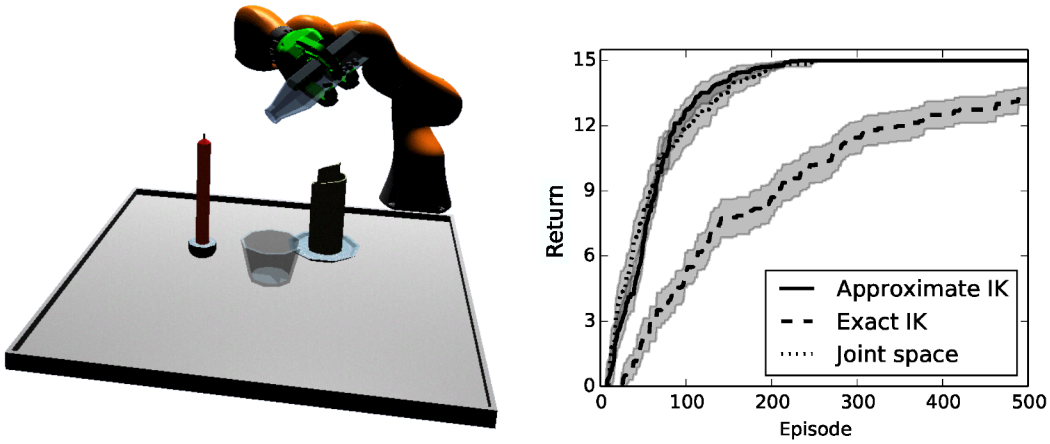
3.2.3.3. Obstacle Avoidance

In the obstacle avoidance problem, the end effector has to avoid several obstacles represented by spheres. It is an artificial problem because we only consider end-effector collisions and not the arm to which it is attached. The environment is displayed in Figure 3.8a. The return is

$$R = -10\left(\sum_{\boldsymbol{\rho}} p\left(\min_{\boldsymbol{q}_t} ||f(\boldsymbol{q}_t) - \boldsymbol{\rho}||_2\right)\right) - 100||f(\boldsymbol{q}_T) - \boldsymbol{g}||_2 - 10^{-2}\left(\sum_{t,j} |\dot{\boldsymbol{q}}_{t,j}|\right) - 10^{-5}\left(\sum_{t,j} |\ddot{\boldsymbol{q}}_{t,j}|\right),$$

where $t \in \{1, \ldots, T\}$ with $T = 101$ represents the step in time, j are the joint indices, $\boldsymbol{g}$ is the desired goal position, and $p(d) = \max(0, 1 - \frac{d}{0.17})$ is greater than zero when the end effector is in the vicinity of one of the obstacles.

Figure 3.8b shows the results. We see that learning in Cartesian space is again more sample-efficient because the primary objective is defined in Cartesian space. The exact



(a) Illustration of the pouring problem. (b) Learning curves for pouring problem.

Figure 3.9.: Pouring problem.

IK performs worse because the approximate IK and learning in joint space results in more smooth trajectories. The difficulty of this task and its return surface are similar to those of the via-point task. The difference is that it is less smooth because the negative penalty for being in the vicinity of obstacles abruptly vanishes when the end effector is outside of the radius of the spheres. In this case only the velocity and acceleration penalties guide the search to a better solution.

3.2.3.4. Pouring a Glass

In the pouring task, the robot fills a glass with 15 marbles from a bottle while avoiding nearby obstacles. The setup is displayed in Figure 3.9a. The reward function and, hence, the return, is complex. For every marble that is inside the glass we add 1. For every marble that is outside of the glass but on the table the negative squared distance to the center of the glass is given. For every marble that is still in the bottle we give a reward of -1. For collisions with obstacles we give a reward of -100 and abort the episode. Marbles that fall down the table also finish the episode and a reward of -1000 is given.

Results are displayed in Figure 3.9b. Learning in joint space and learning in Cartesian space results in nearly identical learning curves. Using an exact IK is again worse than using an approximate solution. The mapping from weights to return in this problem is complex, nonlinear, and not smooth. There are abrupt changes in the reward function when a small change of one weight results in a marble falling down the table or the arm touching an obstacle. There are flat regions, for example, when all marbles stay in the bottle because the arm does not turn the bottle upside down, or all marbles miss the table. Therefore, the structure of the return surface is complex independent of the space in which we describe DMPs, hence, learning in joint space and learning in Cartesian space work similarly well.

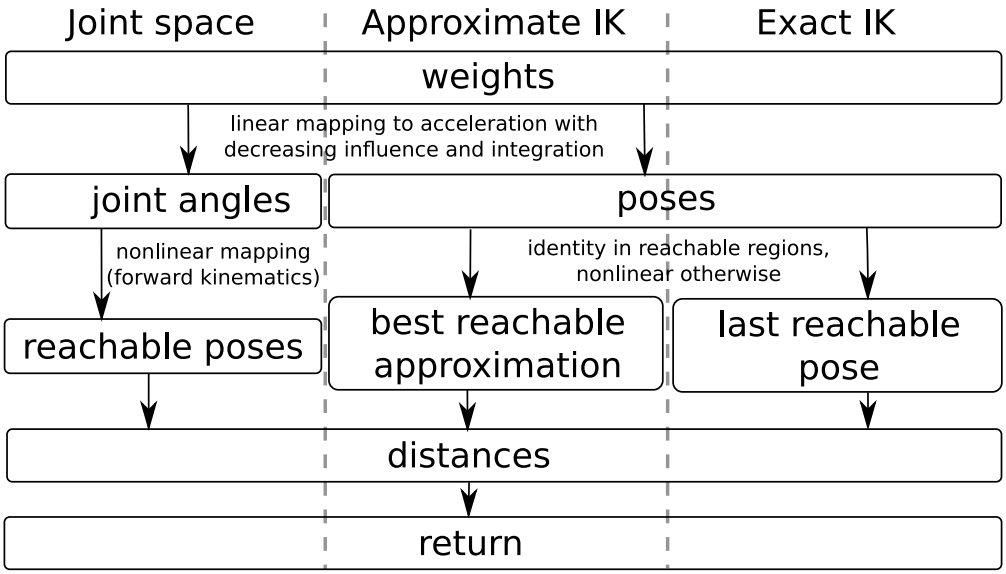


Figure 3.10.: Mapping from weights to corresponding return in the via-point problem. In each case the return of a weight vector involves different mappings.

3.2.4. Discussion

Weights in a DMP are bound to a specific radial basis function. They only influence one specific joint or pose dimension and only affect accelerations of the arm locally in time. They will affect the positions in all following time steps but with a decreasing influence because the weight of the learnable forcing term converges to zero. So it is possible to design a return so that the optimization problem becomes partially separable and, hence, easy to solve.

In the via-point and obstacle avoidance problems there is a direct relation between weights of the Cartesian DMP and obtained return (see Figure 3.10). With the approximate IK we use the weights to generate a pose trajectory. An acceleration is computed based on the linear forcing term that includes the weights and decays exponentially over time. The acceleration is integrated to obtain a trajectory of end-effector poses. The trajectory might not be perfectly executable so that in unreachable regions the pose is mapped to the closest reachable pose. For each via point we compute the distance to the corresponding poses from the trajectory. Hence, the mapping from specific weights to the return in the workspace of the robot is straightforward and the return is partially separable with respect to the weights of a DMP.

Joint space DMPs result in a nonlinear mapping with coupling between dimensions of the weight space because of the forward kinematics of the robot. This results in a difficult optimization problem that is not partially separable anymore and is even multi-modal.

The more complex the relation between weights and return becomes, the smaller is the difference between learning in joint space and learning in Cartesian space. An example is the pouring problem. It has an almost flat return surface where the arm collides with

an obstacle or a marble falls down. Small weight changes can make a difference between a marble staying within the glass and falling down, hence, the return will abruptly change in the corresponding regions of the weight space. The relation between the DMP weights and a successful behavior is complex, highly nonlinear, and non-separable. This eradicates the advantage of learning in Cartesian space.

Hence, learning in Cartesian space can be beneficial but it is not always the best option. It will be advantageous if the main objective has to be solved in Cartesian space and the return is almost separable. It is not always obvious when this is the case though. The advantage of learning in Cartesian space vanishes for more complex, nonlinear, non-separable returns. Policy search works best in the space in which the primary objectives are defined directly. Generating smooth trajectories is simpler in joint space. This is, for example, required for throwing behaviors. When we learn in Cartesian space and a pose is not reachable we should use the best possible approximation of a solution to the IK problem to generate a smooth return landscape.

3.3. Summary

We propose a procedure to automatically generate the embodiment mapping to transfer demonstrated end-effector trajectories from a human to a robotic system. It consists of three optimization phases that transform trajectories so that they are first executable on the target system and then solve the desired task, while the former only requires information about the target system the latter also requires information about the task in form of a reward function that describes how a good solution to the problem has to look like.

The contributions of this chapter to the state of the art are the following: the development of the previously mentioned procedure, a large-scale evaluation of the task-agnostic part of the automated embodiment mapping with various subjects and robotic target systems, the development of an approximation to inverse kinematics, and a comparison of trajectory refinement in Cartesian and joint space.

While inverse kinematics has not been reinvented with the approximation of inverse kinematics, it is a trick that works remarkably well in comparison to using strict inverse kinematics solvers. It is highly beneficial in the task-agnostic embodiment mapping, can be beneficial for policy refinement, and, as we will see in Chapter 4, can be used to generalize behaviors.

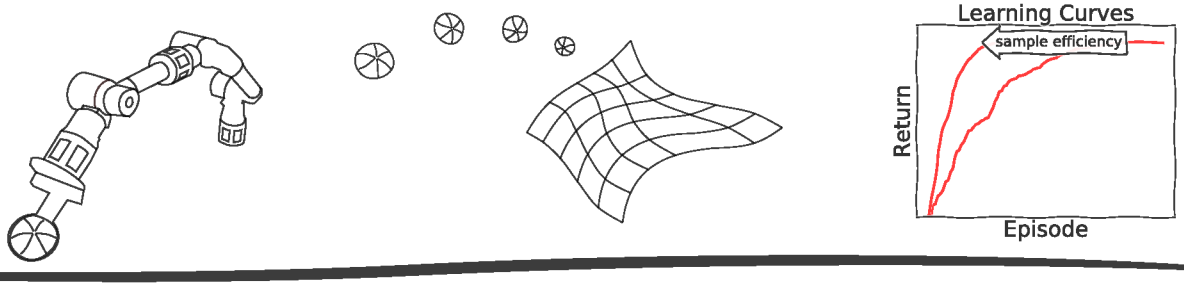
Related Publications

[Fab20] Alexander Fabisch. “A Comparison of Policy Search in Joint Space and Cartesian Space for Refinement of Skills”. In: *Advances in Service and Industrial Robotics*. Ed. by Karsten Berns and Daniel Görges. Springer, 2020, pp. 301–309. ISBN: 978-3-030-19648-6. DOI: 10.1007/978-3-030-19648-6_35.

[Gut+18] Lisa Gutzeit, Alexander Fabisch, Marc Otto, Jan Hendrik Metzen, Jonas Hansen, Frank Kirchner, and Elsa Andrea Kirchner. "The BesMan Learning Platform for Automated Robot Skill Learning". In: *Frontiers in Robotics and AI* 5 (2018), p. 43. ISSN: 2296-9144. DOI: 10.3389/frobt.2018.00043.

[Gut+19] Lisa Gutzeit, Alexander Fabisch, Christoph Petzoldt, Hendrik Wiese, and Frank Kirchner. "Automated Robot Skill Learning from Demonstration for Various Robot Systems". In: *KI: Advances in Artificial Intelligence*. Ed. by Christoph Benzmüller and Heiner Stuckenschmidt. Springer International Publishing, 2019, pp. 168–181. ISBN: 978-3-030-30179-8. DOI: 10.1007/978-3-030-30179-8_14.

The automatic embodiment mapping has been published first in Gutzeit et al. [Gut+18] and the large-scale evaluation has been done in Gutzeit et al. [Gut+19]. It is embedded in a larger framework as we will see in Chapter 5. I contributed the concept of the automatic embodiment mapping, implemented it for several application scenarios that we will discuss in Chapter 5, and did the analysis of transferred trajectories for simulated and real robots in this chapter. In Gutzeit et al. [Gut+19], Christoph Petzoldt contributed to the implementation and application of the global trajectory optimization that has been developed earlier in Gutzeit et al. [Gut+18] as well as the integration of software components. Hendrik Wiese was responsible for recording throwing trajectories from the real UR5. Lisa Gutzeit's contributions to both publications are data recording, preprocessing, segmentation, and classification. These aspects are not discussed in detail in this chapter, but we will summarize them briefly in Chapter 5. The comparison of policy search in Cartesian space and in joint space has been presented in Fabisch [Fab20], which also introduces the approximation to inverse kinematics that we use.



Chapter 4.

Sample-Efficient Contextual Policy Search

Our goal for this chapter is to increase the sample efficiency of algorithms for contextual policy search so that we can learn directly on real robots and generalize over several task parameters. In reinforcement learning we say that we increase sample efficiency, when we reduce the number of required samples. Sample efficiency is sometimes quantified by measuring the number of steps in the environment, but we will count episodes since we often only get a meaningful reward at the end of an episode and episodes often have fixed lengths for the scope of this thesis. We will often use ball throwing with a specified target area (the context space) to measure the performance of new algorithms because ball throwing is a problem that is easy to understand and evaluate but not too easy to solve and it requires learning if we do not have a perfect model of the ball and the robot.

4.1. Active Context Selection

This section was published originally as [FM14] and has been revised.

During contextual policy search we often assume that the agent cannot control the distribution of contexts $p(\boldsymbol{s})$, that is, the environment imposes tasks on the agent. Often an agent would be able to select tasks in which it would like to increase its performance. For instance, when learning target-oriented throwing, the agent can select targets that it cannot hit reliably to improve its performance. In addition, there are problem domains, in which some configurations might be harder to learn than others. An example would be grasping a cup in environments with obstacles. The task is more difficult when the cup is surrounded by other cups or when the handle is on the far side. One way to approach such problems is to start with easy tasks and progress to more complex tasks, where knowledge acquired in prior tasks is transferred and reused. Thus, by autonomously selecting learning tasks (contexts), for example, based on intrinsic motivation [BSC04], an agent can increase its competence in a self-controlled manner.

In this section, we investigate how an agent can actively select contexts to make the best progress in learning the upper-level policy $\pi_\omega(\boldsymbol{\theta}|\boldsymbol{s})$. We propose handling the active task-selection problem as a non-stationary bandit problem and using the algorithm Discounted Upper Confidence Bound (D-UCB) [KS06] as task-selection heuristic. Furthermore, we examine several intrinsic motivation heuristics for context selection during the learning process, which are considered to increase the learning progress, that is, to reduce the number of episodes that are required to master a given contextual problem.

4.1.1. Related Work: Active Learning and Artificial Curiosity

In machine learning it is desirable to require as few training examples as possible because it is often costly to acquire them. For instance, in supervised learning it can be expensive to label data. Similarly, in reinforcement learning domains such as robotics, performing an episode is costly. Hence, we would like to perform those episodes that maximize the learning progress. A research field that deals with selecting data or tasks from which we can learn the most is *active learning*. The goal of active learning is to label only data or examine only tasks that promise the greatest learning progress, that is: the most informative instances. Different query strategies to find these most informative instances are discussed by Settles [Set10].

The idea to actively select tasks that accelerate the learning progress is related to *curriculum learning*: Bengio et al. [Ben+09] and Gulcehre and Bengio [GB13] assume that learning simple concepts first helps to learn more complex concepts that build upon those previously learned simple ones in the context of supervised learning. We also use this hypothesis in our empirical evaluation.

Another related work has been published by Ruvolo and Eaton [RE13] in the context of lifelong multi-task learning, which compares several heuristics for actively selecting the task that will be learned. Among these heuristics are the following: select the task that maximizes the expected information gain (information maximization heuristic) and select the task on which the current model performs worst (diversity heuristic). Once a task is selected, all data with labels of this task are revealed to the agent.

It is not straightforward to transfer this approach to the reinforcement learning setting. A problem that occurs in reinforcement learning is that we do not get the correct solution (the optimal policy) of a queried task directly. Instead, we receive only rewards and need to optimize the solution by trial and error in order to maximize the long-term reward, which requires to select the same training tasks multiple times consecutively until a close-to-optimal policy is learned. This approach is followed by da Silva et al. [dKB14], who extend the parameterized skill introduced by Silva et al. [SKB12] to an active learning setting. For this, the authors propose a novel criterion for skill selection. In this criterion, the skill performance is modeled through Gaussian process regression with a spatiotemporal kernel which addresses the inherent non-stationarity of tracking the skill performance. Based on this estimate of the skill performance, the next task is chosen such that the maximum expected improvement in skill performance would be obtained if the outcome of learning this task is assumed to be an optimistic upper bound. A drawback of this approach is that it may not be the most sample-efficient task-selection strategy to stick for many episodes to the same task until a close-to-optimal policy for this task is learned. We consider task selection in a setting in which a novel task is chosen after each episode, which means that most likely no close-to-optimal policy has been learned in the previous task.

A field related to active learning with applications in reinforcement learning is artificial curiosity [OK04; Got+13]. In particular, Baranes and Oudeyer [BO13] derive self-adaptive goal generation—robust intelligent adaptive curiosity (SAGG-RIAC) from the ideas of artificial curiosity and apply it to learning in contextual problems. SAGG-

RIAC divides the context space into rectangular regions. These are split once the number of contexts that have been explored in these regions exceeds a threshold. For each region $\mathcal{R}_i$, we can compute an interest based on the derivative of competence in that region

$$\text{interest}_i = \frac{1}{\zeta}\left|\left(\sum_{j=|\mathcal{R}_i|-\zeta}^{|\mathcal{R}_i|-\frac{\zeta}{2}} R(\boldsymbol{s}_j, \boldsymbol{\theta}_j)\right) - \left(\sum_{j=|\mathcal{R}_i|-\frac{\zeta}{2}}^{|\mathcal{R}_i|} R(\boldsymbol{s}_j, \boldsymbol{\theta}_j)\right)\right|,$$

where $|\mathcal{R}_i|$ is the number of contexts that have been explored in region $\mathcal{R}_i$, ζ is the size of a sliding window, and $R(\boldsymbol{s}_j, \boldsymbol{\theta}_j)$ is the return of a low-level policy with parameters $\boldsymbol{\theta}_j$ in context $\boldsymbol{s}_j$ (not to be confused with the expected return $\mathbb{E}[R(\boldsymbol{s}, \boldsymbol{\theta})]$). Essentially, this means regions with greater differences between recent and previous returns are considered to be more interesting. Hence, on the one hand SAGG-RIAC focuses on regions where the return increases considerably over time. On the other hand, it also favors regions where the return decreases. Such a decrease might be caused by change in the environment and, hence, it would make sense to explore this region more. SAGG-RIAC selects goals either randomly from the whole context space or from a random region, where the probability of each region corresponds to its interest. For a selected region, the goal is either sampled from a uniform random distribution or from the vicinity of the context with the lowest previous return. All three cases are selected randomly with fixed probabilities.

SAGG-RIAC naturally handles continuous context spaces, which is an advantage. It will most of the time concentrate on regions where it expects a high learning progress but does not converge to a single region of the context space. As an alternative to SAGG-RIAC, we present an approach that is based on estimates of the learning progress and multi-armed bandit algorithms, which more naturally trade off exploration and exploitation of the noisy estimate of the learning progress.

4.1.2. Proposed Method for Active Context Selection

In contextual policy search we seek to maximize $J(\boldsymbol{\omega}) = \int_{\mathbb{S}} p(\boldsymbol{s}) \int_{\mathbb{R}^n} \pi_{\boldsymbol{\omega}}(\boldsymbol{\theta}|\boldsymbol{s}) R(\boldsymbol{s}, \boldsymbol{\theta})\, d\boldsymbol{\theta}\, d\boldsymbol{s}$ by changing $\boldsymbol{\omega}$ based on collected experience from episodes, in which we use control policies with parameters $\boldsymbol{\theta}$ in contexts $\boldsymbol{s}$. In contrast to prior work, we consider the context distribution during learning not to be fixed but to be under the agent's control. Although this assumption might not apply to all contextual policy search problems, there are sufficiently many settings to make studying this setting worthwhile. For instance, ball throwing with self-selection of targets. Note that the context distribution $p(\boldsymbol{s})$ within the objective $J(\boldsymbol{\omega})$ remains unaffected.

4.1.2.1. Overview

Formally, we introduce a context selection policy π_β, which selects the context in which the next episode will be performed. π_β aims at optimizing the expected *learning progress* of a contextual policy search method such as C-REPS and, hence, minimizing the required number of episodes to reach a desired level of performance. In contrast, the control policy

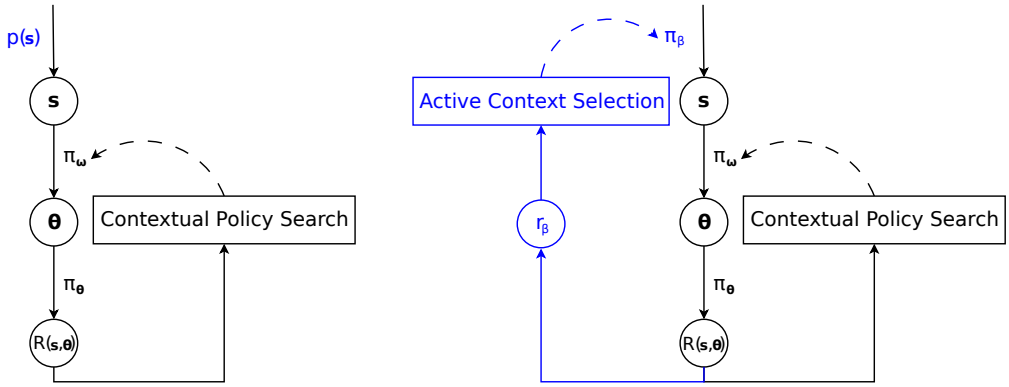


Figure 4.1.: Active versus passive context selection. Contextual policy search (left): A context vector $\boldsymbol{s}$ is given by the environment according to some fixed distribution $p(\boldsymbol{s})$, the upper-level policy $\pi_{\omega}(\boldsymbol{\theta}|\boldsymbol{s})$ generates the parameters of the control policy for $\boldsymbol{s}$, the control policy is executed in the environment and a return $R(\boldsymbol{s},\boldsymbol{\theta})$ is obtained. A contextual policy search component updates the upper-level policy based on the return with the goal to maximize $J(\boldsymbol{\omega})$. Active contextual policy search (right): The context vector $\boldsymbol{s}$ is selected by the context selection policy π_{β} which will be adjusted by an active context selection component based on the intrinsic reward r_{β}, an estimate of the learning progress based on $R(\boldsymbol{s},\boldsymbol{\theta})$. Differing parts are marked in blue.

$\pi_{\boldsymbol{\theta}}$ and the upper-level policy π_{ω} are learned to maximize $J(\boldsymbol{\omega})$ directly. An illustration of the components of contextual policy search is given in Figure 4.1.

We define the learning progress at time t when performing an episode in context $\boldsymbol{s}$, with the control policy parameters $\boldsymbol{\theta}$ selected according to π_{ω}, as $\Delta_{\boldsymbol{s}}(t) = J(\boldsymbol{\omega}_{t+k}) - J(\boldsymbol{\omega}_t)$, where $\boldsymbol{\omega}_t$ are the parameters learned by contextual policy search at time t. Note that contextual policy search methods such as C-REPS do not update $\boldsymbol{\omega}$ after every episode but only after a certain number of episodes k. Hence, it is more appropriate to define the learning progress over the window k than between successive values of $J(\boldsymbol{\omega}_t)$.

The learning task on the task-selection level can now be framed as finding π_{β} such that π_{β} selects contexts that maximize the expected learning progress $\mathbb{E}_{\pi_{\omega}}[\Delta_{\boldsymbol{s}}(t)]$. The learning progress $\Delta_{\boldsymbol{s}}(t)$ is stochastic because the environment and the upper-level policy π_{ω} are stochastic, hence the expectation. Furthermore, the learning progress is also non-stationary since it depends on the quality of $\boldsymbol{\omega}_t$: if $J(\boldsymbol{\omega}_t)$ is already close to the optimum, the expected learning progress is smaller than the learning progress at the beginning of learning when $\boldsymbol{\omega}_t$ is expected to be far from optimal.

We restrict π_{β} to select among a finite number K of predetermined contexts $\{\boldsymbol{s}_1, \ldots, \boldsymbol{s}_K\}$ during learning. These K predetermined contexts are, however, elements of a continuous context space $\mathcal{S}$ over which π_{ω} should generalize and where $J(\boldsymbol{\omega})$ is evaluated. The restriction to a discrete set of training contexts allows application of well-established multi-armed bandit algorithms. Choosing the set of training contexts can be based

Algorithm 4 D-UCB [KS06]

Require: K: number of tasks; γ: discounting factor; $\xi > 0$: parameter that controls the strength of padding; t: number of episodes; $i_1, \ldots, i_t$: previously selected tasks; $r_1, \ldots, r_t$: previous rewards

if $t < K$ **then**
  return t ▷ Sample each task at least once
else
  for $i \in \{1, \ldots, K\}$ **do**
    $n_i \leftarrow \sum_{t_j=1}^{t} \gamma^{t-t_j} \mathbb{1}_{\{i_{t_j}=i\}}$ ▷ Discounted number of episodes in task i
  end for
  $n \leftarrow \sum_{i=1}^{K} n_i$ ▷ Discounted total number of episodes
  for $i \in \{1, \ldots, K\}$ **do**
    $\overline{r}_i \leftarrow \frac{1}{n_i} \sum_{t_j=1}^{t} \gamma^{t-t_j} r_{t_j} \mathbb{1}_{\{i_{t_j}=i\}}$ ▷ Discounted mean reward in task i
    $c_i \leftarrow 2B\sqrt{\frac{\xi \ln n}{n_i}}$ ▷ Padding function for task i
  end for
  return $\arg\max_{i \in \{1,\ldots,K\}} \overline{r}_i + c_i$ ▷ Select task in which discounted UCB is maximal
end if

either on domain knowledge or on simple heuristics. In low-dimensional context spaces, an equally spaced grid over the context space is sufficient. In high-dimensional context spaces this is infeasible because of the course of dimensionality, and sampling the set of training contexts from a uniform random distribution over the continuous context space is more viable.

4.1.2.2. Non-Stationary Multi-Armed Bandit Problems

A Multi-armed bandit problem (MABP) [Rob52; BC12] can be regarded as a one-step or one-state Markov decision process, in which an agent has to select one of K actions and then obtains a reward that is assumed to be drawn iid for each action. The agent tries to minimize regret, which is defined as the expected difference between its accumulated reward and the reward that would have been accumulated with the optimal but unknown stationary policy. UCB [Agr95] is a popular algorithm in this setting. It uses a deterministic policy that selects the action with the maximum upper bound on the confidence interval of the reward. This upper bound is constructed from the past rewards for the action and is based on their empirical mean and a padding function. The padding function summarizes the uncertainty in the estimate of the expected reward. Since UCB always selects the action with the maximum upper bound, this results in choosing actions where the reward is either highly uncertain (large padding) or expected to be high (large empirical mean). By this, UCB trades off exploration and exploitation.

One crucial assumption of most MABP algorithms is that the reward distributions do not change over time. This assumption will not be satisfied in our settings. There exist

bandit algorithms that are designed to deal with non-stationary MABPs, in which the reward distributions might change over time, either abruptly, leading to sliding window UCB [GM11], or slowly but continuously, leading to discounted UCB [KS06].

The problem of learning π_β can be framed as a MABP, where the contexts correspond to the *arms* and r_β to the bandit's reward, which is the learning progress in the corresponding context. Due to the non-stationarity of r_β, we propose to use D-UCB (Algorithm 4) for active context selection since it explicitly addresses changing reward distributions. For this, D-UCB estimates the instantaneous expected reward by a weighted average of past rewards where higher weight is given to recent rewards. More specifically, a discount factor $\gamma \leq 1$ is introduced and the reward that has been obtained at time step t_j is weighted with the factor γ^{t-t_j} at time t. The central idea of D-UCB is that the discounting can compensate for continuously but slowly changing reward distributions.

The upper confidence bound of the D-UCB algorithm for arm i takes the form $\overline{r}_i + c_i$ where $\overline{r}_i = \frac{1}{n_i} \Sigma_{t_j=1}^{t} \gamma^{t-t_j} r_{t_j} \mathbb{1}_{\{i_{t_j}=i\}}$ is the discounted mean and $c_i = 2B\sqrt{\xi \ln(n) \,/\, n_i}$ is the padding function which controls the width of the confidence intervals. In these formulas, n_i corresponds to the discounted number of samples of arm i and n to the total discounted number of samples (see Algorithm 4). Furthermore, r_{t_j} is the reward obtained in the t_j-th episode and i_{t_j} is the arm played in this episode. $\mathbb{1}_{\{i_{t_j}=i\}}$ is the Kronecker delta, which is one if the equality holds and zero otherwise. B and ξ are parameters of the algorithm which control the width of the confidence intervals, where B is an upper bound on the rewards and ξ needs to be chosen appropriately.

4.1.2.3. Intrinsic Reward Functions for Context Selection

Since $\mathbb{E}_{\pi_\omega}[\Delta_s(t)]$ is unknown to the agent and difficult to estimate because of its non-stationarity, we propose several heuristic but easily computable reward functions r_β that reward the agent for certain proxy criteria. These proxy reward functions can be considered as means for intrinsic motivation of an agent [BSC04], which drives it to engage in activity that increases its competence in task solving on a larger time scale. In empirical experiments, we evaluate which r_β is a good proxy for the actual expected learning progress. Note that the heuristics that we propose are not exactly comparable to query strategies from supervised learning, since the learning process in reinforcement learning in a specific task is iterative with distribution shifts unlike most supervised settings.

In a generalized form, we can write the proposed proxies of the learning progress as

$$r_\beta = f(R(\boldsymbol{s}_t, \boldsymbol{\theta}_t) - \hat{b}_{\boldsymbol{s}_t}),$$

where $R(\boldsymbol{s}_t, \boldsymbol{\theta}_t)$ is the return obtained in episode t with policy $\pi_{\boldsymbol{\theta}_t}$ in context $\boldsymbol{s}_t$, $\hat{b}_{\boldsymbol{s}_t}$ is a baseline term, and f is an operator. Note that the bandit algorithm handles stochasticity in $R(\boldsymbol{s}_t, \boldsymbol{\theta}_t)$ so that the heuristics do not have to account for it.

Best-Reward Heuristic: This heuristic directly uses the reward obtained by the control policy $\pi_{\boldsymbol{\theta}}$ in task $\boldsymbol{s}$ and episode t as the proxy for the learning progress, that is, $r_\beta = R(\boldsymbol{s}_t, \boldsymbol{\theta}_t)$. Thus $\hat{b}_{\boldsymbol{s}_t} = 0$ and the operator f is the identity. This heuristic corresponds

to assuming that the agent makes the most progress when it focuses on improving its performance in tasks in which it is already performing well. The idea behind this heuristic is that we should first learn easy tasks because this can help us to learn similar but more difficult tasks later on.

A potential problem of this heuristic are problems in which high reward does not correspond to easy tasks, for instance, when the maximum achievable reward differs in contexts. Additionally, the best-reward heuristic requires that knowledge can be transferred well between contexts. Otherwise it converges to context with the highest rewards.

Diversity Heuristic: As the opposite of the best-reward heuristic, this heuristic encourages to select tasks in which the agent receives the worst reward. For this, the negative actual reward $r_\beta = -R(\boldsymbol{s}_t, \boldsymbol{\theta}_t)$ is used. Thus $\hat{b}_{\boldsymbol{s}_t} = 0$ and the operator f is simply $f(x) = -x$. The intuition for this heuristic is that we focus on the hardest tasks in which the current performance is worst since in these tasks the potential for large improvements is high. This heuristic bears similarities to the heuristic proposed by Ruvolo and Eaton [RE13] for supervised learning and is hence called *diversity heuristic*.

Similar to the best-reward heuristic, this heuristic might have problems in settings in which the obtained rewards might not be comparable between different contexts. Another disadvantage appears in problems with unlearnable contexts, as the diversity heuristic would then focus on these.

1-step Progress Heuristic: This heuristic uses the difference of the last two rewards obtained in the same context as proxy for the actual learning progress, that is, $r_\beta = R(\boldsymbol{s}_t, \boldsymbol{\theta}_t) - R(\boldsymbol{s}_{\bar{t}}, \boldsymbol{\theta}_{\bar{t}})$, where $\bar{t}$ is the index of the previous episode in context $\boldsymbol{s}$. Thus, the baseline is the last obtained return $\hat{b}_{\boldsymbol{s}_t} = R(\boldsymbol{s}_{\bar{t}}, \boldsymbol{\theta}_{\bar{t}})$ and the operator f is the identity. The heuristic is a direct proxy for the learning progress, as it replaces effectively the integration over $\boldsymbol{s}$ and $\boldsymbol{\theta}$ in $J(\omega)$ by sampling. As it is based solely on differences of rewards rather than absolute values, it should cope better than the best-reward and diversity heuristic with situations, in which the maximum reward differs across contexts.

A drawback of the 1-step progress heuristic is that reward signals r_β have high variance. Variance in r_β is inevitable because of the stochasticity of both the environment and the upper-level policy π_ω, however, the baseline has also a high variance.

Monotonic Progress Heuristic: Although bandit algorithms account for the stochasticity in the 1-step progress heuristic, it might be useful to reduce the variance of the intrinsic reward. We can use more stable baselines such as the maximum reward of all previous episodes in the context $\boldsymbol{s}$, that is: $\hat{b}_{\boldsymbol{s}_t} = \max_{t'} R(\boldsymbol{s}_{t'}, \boldsymbol{\theta}_{t'})$. Furthermore, since the learning progress should be monotonically increasing, using the operator $f(x) = \max(0, x)$ to avoid negative r_β appears to be reasonable. The resulting heuristic is denoted as *monotonic progress heuristic* and has the form $r_\beta = \max(0, R(\boldsymbol{s}, \boldsymbol{\theta}_t) - \max_{t'} R(\boldsymbol{s}_{t'}, \boldsymbol{\theta}_{t'}))$.

The heuristic considers the learning progress to be monotonic and positive. In comparison to the 1-step progress heuristic, the intrinsic reward r_β will be more often zero and its baseline has less variance since unsuccessful explorative episodes have less influence.

4.1.3. Experiments: Generalizing Throwing Movements

We want to answer the following questions: (1) Which intrinsic reward is the best for active contextual policy search? (2) How much more sample efficient is active context selection in comparison to standard contextual policy search and is the final return better? (3) Is context selection plausible?

Preliminary experiments that compare context selection methods are described in Appendix E for the sake of brevity. These also contain comparisons of our approach to SAGG-RIAC. The most interesting experiments are presented here.

Figure 4.2.: Visualization of the simulated Mitsubishi PA-10 throwing a ball.

We consider the problem of learning to throw a ball at a given target. A similar setting has been investigated by Wirkus et al. [WdK12], where the objective was to learn throwing an object at a specified target based on a forward model of the system. In our experiments, the target can be located at different positions in a predetermined area on the ground and we do not provide any model of the system, making it a contextual model-free policy search problem with the target position being the context. We consider two different reward functions for this experiment. One reward function is continuous (grid problem) while the other has discontinuities and flat regions without clear direction of improvement (dartboard problem), resulting in a setting with easy and more difficult tasks. We provide an empirical evaluation on the grid problem and apply the gained insight on the dartboard problem. In our experiments, we use a simulated Mitsubishi PA-10 robot arm with seven joints for throwing (see Figure 4.2 and Appendix F.4 for details).

4.1.3.1. Methods

A throwing behavior in this experiment consists of a sequence of two DMPs, where the first corresponds to the strike out and the second one to the actual throwing movement. Both primitives have a duration of $\tau_1 = \tau_2 = 0.5\,\mathrm{s}$. The movement primitives define joint trajectories directly for the 7 joints of the Mitsubishi PA-10. The weights of the forcing terms and the metaparameters of the DMPs (DMP formulation of Mülling et al. [MKP11; Mül+13]) have been initialized such that the resulting throw hits the ground position $(-3.55m, -3.55m)$, where $(0m, 0m)$ is the position of the PA-10. Some of the metaparameters of the initial policy are to be adapted later on such that other ground positions are hit. These include the final state of the first movement primitive $\boldsymbol{g}_1$ and the velocities at the end of the two movement primitives $\dot{\boldsymbol{g}}_1, \dot{\boldsymbol{g}}_2$ so that our low-level policy is

described by the parameters $\boldsymbol{\theta} = (\boldsymbol{g}_1, \dot{\boldsymbol{g}}_1, \dot{\boldsymbol{g}}_2)$. The weights $\boldsymbol{w}$ of the forcing terms of the two primitives remain fixed during this adaptation. Thus, the contextual learning task consists of finding a mapping for $3 \cdot 7 = 21$ parameters such that different target positions on the ground are hit.

We use C-REPS for contextual policy search with the context $\boldsymbol{s}$, which contains the Cartesian coordinates of the target position for the throw. The mapping $\boldsymbol{\phi}(\boldsymbol{s})$ projects the context to polar coordinates and generates all quadratic terms of the polar coordinates. A policy update is performed after every 50 episodes and a memory of at most 300 episodes is used for the update. This memory consists of the best $300/K$ episodes for each of the K tasks. We restricted the maximum Kullback-Leibler divergence of the old and new policy distributions to $\epsilon = 0.5$, the initial weight matrix was set to $\boldsymbol{W} = \boldsymbol{0}$, the initial covariance was set to $\boldsymbol{\Sigma} = \sigma_0^2 \boldsymbol{I}$ with $\sigma_0^2 = 0.02$, and the regularization weight to $\lambda = 10^{-4}$. For D-UCB, we use $\gamma = 0.99$, $B = 10,000$, and $\xi = 10^{-9}$ in all cases.

4.1.3.2. Grid Problem

We define two contextual learning problems in this setting whose main difference is the structure of the reward function. The first reward function provides the squared distance of the position hit by the throw to the target position as reward so that we can directly compute the return as $R(\boldsymbol{s}, \boldsymbol{\theta}) = -||\boldsymbol{s} - \boldsymbol{b}_{\boldsymbol{\theta}}||_2$, where $\boldsymbol{s}$ is the target and $\boldsymbol{b}_{\boldsymbol{\theta}}$ are the Cartesian coordinates of the ball when it hits the ground after executing policy $\boldsymbol{\pi}_{\boldsymbol{\theta}}$. In this problem, we generate an equidistant grid of 25 targets for training over the area $[-3m, -5m] \times [-3m, -5m]$. Another set of 16 targets is used to test the generalization. These targets form an equidistant grid in the area $[-3.25m, -4.75m] \times [-3.25m, -4.75m]$. While different contexts share the same reward function, learning them might differ in complexity. Some of the targets are harder to hit because of the kinematic structure of the arm. In addition, the distance to the initial policy makes some contexts easier to solve by exploration than others.

We compared our methods with three baselines: continuous random sampling of contexts from $[-3m, -5m] \times [-3m, -5m]$ (*Random (cont.)*), selection in a fixed order (*Round Robin*) and the best policy that we have found in all experiments (*Best Policy*). On the left side of Figure 4.3 the learning curves of several intrinsic reward heuristics are shown and on the right side the best intrinsic reward heuristic is compared with the baselines.

The *diversity heuristic*, which prefers selecting hard tasks in which a low reward is obtained, performs worse than round robin or random selection. Thus, selecting the more difficult tasks first is not beneficial to speed up learning in this setting. This effect might be even more pronounced, when the most difficult tasks are unsolvable. The *Best-Reward heuristic* performs worst here. We observed that it quickly converged to nearly always selecting the same task and hence failed to learn a generalizable upper-level policy. The reason for this behavior is that it encourages D-UCB to focus on the simplest task first. Because it improves quickly in this task, the reward will be considerably greater than the reward of the other tasks. Even though it periodically samples other tasks, the reward in these tasks will be smaller and thus, D-UCB sticks to the same task. Selecting the tasks in which we can make the greatest estimated learning progress gives

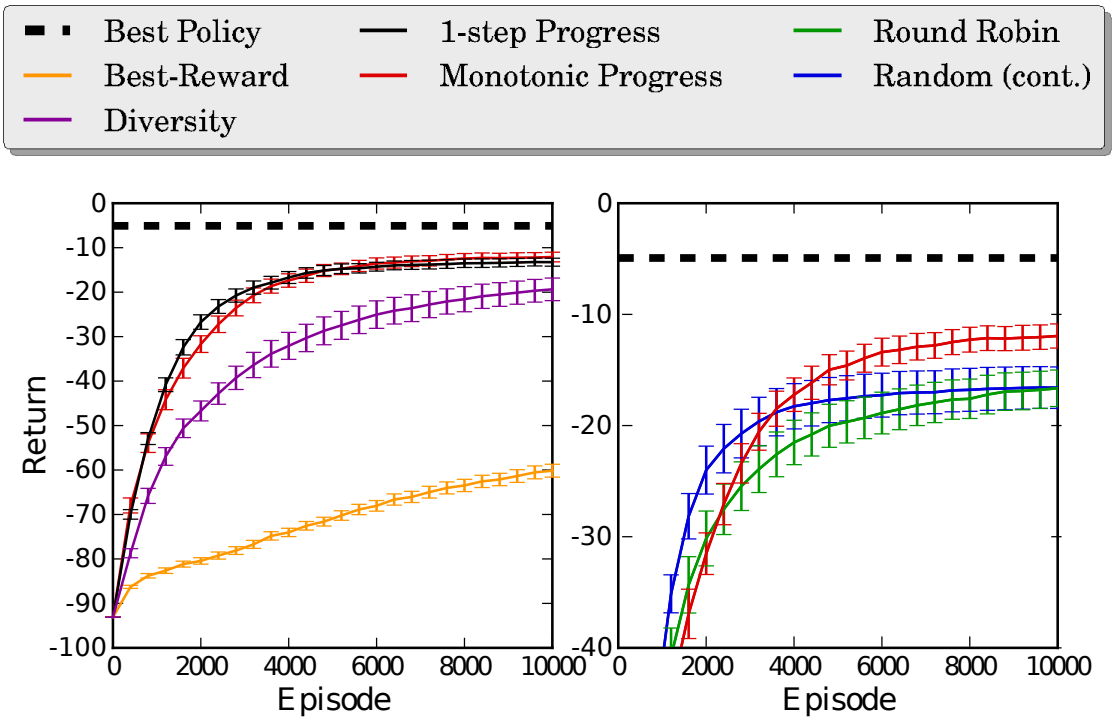


Figure 4.3.: Left: Learning curves of several intrinsic reward heuristics for D-UCB in the grid problem. The return is the average negative distance to test targets. The curves and the error bars show the mean and standard error of the mean over 20 runs. *Best Policy* indicates the performance of the best policy that we found in all experiments. Right: Comparison of D-UCB with intrinsic reward based on the monotonic progress with the baselines.

a considerable advantage in this problem. In contrast to the results in Section E.2.2, the *1-step Progress heuristic* is on a par with the *Monotonic Progress* heuristic. A possible reason for this is that the inherent context complexities in this task do not vary as strongly as in the problem in Section E.2.2. In comparison to the baselines, D-UCB under the monotonic progress intrinsic reward is on a par with round robin selection and slightly worse than continuous random selection. Continuous random sampling learns more quickly in the beginning because the comparison of different methods is done on a set of test contexts that are maximally dissimilar from the discrete training context set. Methods that use only the discrete set of training contexts during the training phase have thus a disadvantage compared with continuous random sampling, which often samples closer to the test contexts.

In the long-term, however, D-UCB performs better than both baselines and its average performance gets closer to the best policy's performance. This shows that active task selection can accelerate learning considerably and improve the reliability of the result of

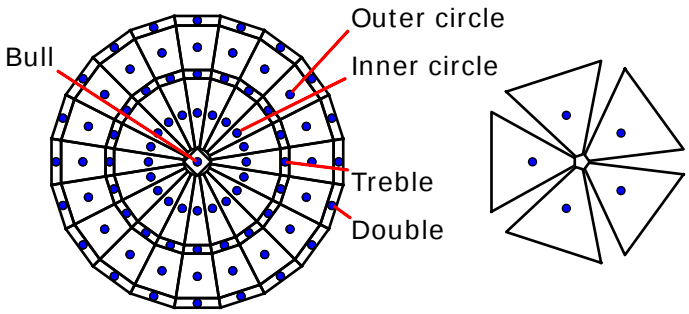


Figure 4.4.: Left: The dartboard is composed of 81 quadrangular fields. Bull and bull's eye are regarded as one field, the rest belongs to one of the four circles (inner circle, outer circle, doubles, and trebles). The initial policy given by $\boldsymbol{\theta}_0$ would throw at the center of the dartboard, that is the bull. Right: The target regions of the bull, inner circle, and outer circle are greater than the corresponding fields on the dartboard and overlap neighboring fields. Shown here are 5 of the 20 target regions of the inner circle. These target regions are only used to compute the return. Larger target regions can be assumed to make the corresponding learning tasks easier.

contextual policy search. A potential reason why continuous random sampling performs worse in the long term is that it cannot use the strategy otherwise employed in C-REPS, namely to keep a separate history of samples per context of identical size ($300/K$ episodes per context), because it does not encounter any context twice.

4.1.3.3. Dartboard Problem

As second scenario with the PA-10, we consider a problem which poses tasks of more varying complexity onto the agent. The targets are fields on a dartboard. A virtual dartboard is placed on the ground in front of the PA-10. Placing the dartboard on the ground instead of a wall was done to simplify implementation. The diameter is set to 1.4 m (real dartboards have a diameter of 0.451 m). Each field is approximated by quadrangles (see Figure 4.4) and the center of this field is the context of the task corresponding to the field. For each target, we assign a corresponding target area, which is usually the quadrangular field. For some tasks, we enlarge each side of the quadrangular field by the factor 3.5 to build the corresponding target region (see Figure 4.4 for details). This will make these tasks considerably easier than others. The reward function gives a constant negative reward outside of this target area and only provides a reward gradient inside the target area. The return is defined as

$$R(\boldsymbol{s}, \boldsymbol{\theta}) = \begin{cases} -\frac{10{,}000}{d_{\boldsymbol{s}}} ||\boldsymbol{s} - \boldsymbol{b}_{\boldsymbol{\theta}}||_2 & \text{if } \boldsymbol{b}_{\boldsymbol{\theta}} \text{ is within the target area of } \boldsymbol{s} \\ -10,000 & \text{otherwise} \end{cases},$$

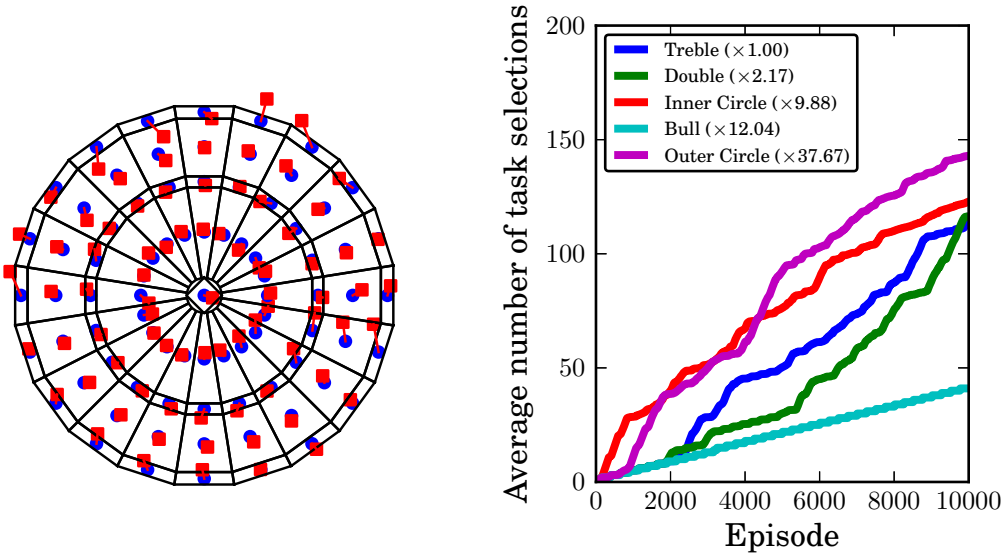


Figure 4.5.: Left: Final upper-level policy. The blue circles mark the center of each field, the red squares that are connected to the corresponding centers with a red line show the position at which the robot arm has actually thrown the ball. Right: Accumulated average number of selections for different task categories. Larger target areas and target areas that are closer to the initial policy are selected more often at the beginning. The relative size in comparison to the smallest target region is given in brackets in the legend.

where $d_{\boldsymbol{s}}$ is the maximum distance to the center within the target area of context $\boldsymbol{s}$. A constant reward outside of the target area complicates the problem compared with the grid problem, as no reward gradient helps the agent if it does not hit the target area, within which a reward gradient guides the agent to the center of the field. Thus, the tasks corresponding to larger fields can be considered to be easier since it is more likely that the agent finds a reward gradient through exploration. The agent should thus focus first on these tasks and select more difficult tasks not before it has improved its upper-level policy so much that it is able to hit the corresponding target areas.

We train with all 81 targets for $50,000$ episodes using the monotonic progress intrinsic reward. Results are displayed in Figure 4.5: the final policy hits the bull, 18 of 20 fields in the inner circle, 15 of 20 trebles, 20 of 20 fields in the outer circle, and 9 of 20 doubles. The mean of the error is 4.62 cm, the median 3.77 cm, and the maximum 15.96 cm.

Doubles are the most difficult tasks to learn for the agent because they are far away from the initial policy, the target region is small, and we cannot transfer much knowledge from neighboring tasks because they are on the edge of the dartboard. For this reason, some doubles have not been learned well. In contrast, the trebles can be learned easily, because the solution can be obtained approximately by transferring the solutions of neighboring fields from the inner circle and the outer circle.

In Figure 4.5 we show which kind of tasks have been selected in the initial learning phase. We can see that the inner circle and the outer circle, which have the greatest target regions, are selected most often in the beginning. The targets from the inner circle are selected even more often than the targets from the outer circle during the first 1,000 episodes even though they are smaller. This is because they are more likely to be reached when exploring from the initial policy, which throws at the center of the dartboard. For the same reason trebles are selected more often than doubles at the beginning. After this initial phase, fields of the outer circle are selected more often because they are now much easier to learn. The bull is so rarely selected because the initial policy will already generate a good result for this context, hence, improvement is hardly possible.

4.1.4. Discussion

We investigated the hypothesis that active selection of contexts during training time can improve sample efficiency. We proposed to use the non-stationary bandit algorithm D-UCB to select tasks that have a large estimated learning progress. The learning progress is estimated by heuristics that can be considered as proxies for the actual learning progress. The underlying model of the learning process assumes that each task has a different intrinsic expected learning progress which might change abruptly in the context space and that knowledge can be transferred between similar contexts.

Our empirical results show that active task selection can make a difference for the learning speed of contextual policy search, the performance of the final policy, and the stability of the learning process. We found that a task-selection method should explore in the beginning, then focus on several easy tasks to acquire some initial procedural knowledge, thereupon transfer knowledge to similar but more difficult tasks, and concentrate in the end on those tasks that have not been learned yet. Some of the proposed intrinsic reward heuristics provide an advantage over round robin or uniform random selection.

A limitation is that we restricted ourselves to a discrete set of context vectors for training. It is interesting to note that we are thus able to keep a history of the best samples for each context and use this for the update of the upper-level policy instead of just the last N samples. This is, however, not easily possible for continuous contexts.

Moreover, it would also be desirable to stop sampling of tasks for which the policy has reached an acceptable level of performance, that is, tasks that are solved. Since the expected learning progress becomes small in such tasks, an active context selection mechanism should recognize this. As D-UCB assumes non-stationary rewards, it assumes that the expected reward could increase again and will continue to sample such tasks. Active context selection algorithms, which take the property of the learning progress to converge to zero into account, could thus be better than D-UCB.

Among the kind of tasks for which we consider active contextual policy search to be promising are goal-directed reaching, hitting and throwing problems such as ball throwing, darts, and hockey. Moreover, scenarios in which an agent can select from a small set of predefined contexts, for example, grasping one object from a set of objects with varying size and shape, are promising.

Our objective for active context selection was to reduce the amount of episodes that we have to perform to successfully learn a certain behavior. We set a few hundred episodes as the objectives for this thesis and we can clearly see in Figure 4.3 that a well-performing throwing behavior is only reached after a couple of thousand episodes. Even worse, continuous random sampling seems to be slightly better in the region that we are interested in although the overall performance is so bad after a few hundred episodes that this does not matter. Hence, adding active context selection alone is not an appropriate means to achieve this objective. It is, however, a method that can easily be applied to a variety of contextual policy search algorithms to improve their performance, stability, and even their sample efficiency slightly.

4.2. Active Training Set Selection

This section was published originally as [Fab+15] and has been revised.

A set of samples $(\boldsymbol{s}, \boldsymbol{\theta}, R(\boldsymbol{s}, \boldsymbol{\theta}))$ is required for contextual policy search to update the upper-level policy. Therefore, in each episode we test parameters $\boldsymbol{\theta}$ in context $\boldsymbol{s}$ and observe the corresponding return $R(\boldsymbol{s}, \boldsymbol{\theta})$. As we have seen in the previous section it is also sometimes useful to keep a buffer of the best previous samples for the update. Determining the best samples for a discrete set of contexts is possible for each context individually, but it is not possible for continuous contexts because returns are often not comparable between contexts. The maximum return value in one context could be far from optimal in another context. We have to normalize returns with respect to their corresponding context before we select the best samples. In this section we will introduce a method for reward normalization based on the context.

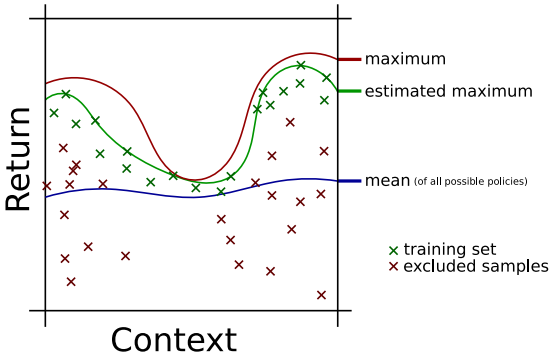


Figure 4.6.: Active training set selection.

4.2.1. Proposed Method for Training Set Selection

In this section, we propose a novel approach for addressing task incommensurability. This approach allows to estimate an upper boundary of the achievable return and a typical return range (as illustrated in Figure 4.6) and thus, allows to make returns comparable by normalization. We can then use these normalized returns to keep the best samples for the next update.

4.2.1.1. Positive Upper Boundary Support Vector Estimation

Given observations $\mathcal{D} = \{(\boldsymbol{s}_i, \boldsymbol{\theta}_i, R_i)\}_{i=1}^n$, we assume that for all i, R_i is an estimator of $\mathbb{E}[R(\boldsymbol{s}_i, \boldsymbol{\theta}_i)] = V^{\pi_{\boldsymbol{\theta}_i}}(\boldsymbol{s}_i) \leq V^*(\boldsymbol{s}_i)$. In contrast to standard regression problems, we do not want to approximate the mean function $V^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s})$ of some policy $\pi_{\boldsymbol{\theta}}$ but we want to

approximate the upper boundary $V^*(\boldsymbol{s})$, which is the context value function of the best upper-level policy.

One constraint on the approximation $\hat{V} \approx V^*$ is that $\hat{V}(\boldsymbol{s}_i) \geq R_i$, that is, $\hat{V}$ should be an *upper boundary* on returns from $\mathcal{D}$. Since this constraint does not uniquely determine the approximation, we add two additional objectives: (1) $\hat{V}$ should be smooth, that is, similar inputs $\boldsymbol{s}_1 \approx \boldsymbol{s}_2$ should have similar values $\hat{V}(\boldsymbol{s}_1) \approx \hat{V}(\boldsymbol{s}_2)$. (2) $\hat{V}$ should be less or equal to V^* in the absence of data, that is, it should be a pessimistic estimate with a bias towards values that are too small. Since V^* corresponds to the value of the optimal policy, the current policy will almost always obtain returns smaller or equal to V^* and a pessimistic estimate $\hat{V}$ will necessarily be closer to the level of performance of the current policy than a too optimistic estimate with the same overall error $|V^* - \hat{V}|$.

Since a linear model of $\hat{V}$ is often too restrictive, we use a non-parametric, kernelized model for $\hat{V}$, that is, $\hat{V}(\boldsymbol{s}) = b + \sum_{i=1}^{n} \alpha_i k(\boldsymbol{s}_i, \boldsymbol{s})$ with offset b and RBF kernel $k(\boldsymbol{s}_i, \boldsymbol{s}_j) = \exp(-\gamma ||\boldsymbol{s}_i - \boldsymbol{s}_j||^2)$ for bandwidth γ. This model allows for both a smooth (constant) upper boundary by setting $\alpha_i = 0$ and $b \geq \max_{i'} y_{i'}$ and a pessimistic upper boundary for large γ and small b. We propose a new method, the positive upper boundary support vector estimation (PUBSVE), for learning such a model of the upper boundary. While the concept has been developed and the algorithm has been implemented for this thesis, the learning algorithm has been derived in collaboration with Krell [Kre15, pages 199–201].

Without loss of generality, we assume that all R_i are positive (because we subtract $\min_{i'} R_{i'}$ for normalization). This implies $b \geq 0$. The PUBSVE has the objective

$$\begin{aligned} &\min_{\boldsymbol{\alpha}, b} \frac{1}{2} \sum_{i,j} \boldsymbol{\alpha_i} \boldsymbol{\alpha_j} k(\boldsymbol{s}_i, \boldsymbol{s}_j) + \frac{H}{2} b^2 \\ \text{subject to} \quad &b + \sum_{j} \boldsymbol{\alpha_j} k(\boldsymbol{s}_i, \boldsymbol{s}_j) \geq R_i \text{ for all } i. \end{aligned}$$

H is a hyperparameter that allows to balance between a pessimistic upper boundary ($b \to 0$ when $H \to \infty$) and a constant upper boundary ($b \to \max_{i'} R_{i'}$ and $\boldsymbol{\alpha} \to 0$ when $H \to 0$). $H = 100 \gg 0$ gives good results empirically and is used in our experiments. The RBF's bandwidth γ controls the generalization between similar samples: the greater γ is the less similarity between similar inputs is assumed and the more local will the generalization be. The model has similarities to support vector machines and allows to use related implementation techniques [MM98; SHS09]. Further details are provided by Krell [Kre15].

4.2.1.2. Incremental Return Normalization

In this section, we describe how PUBSVE allows to normalize the obtained return: for a given context $\boldsymbol{s}$, we map the PUBSVE prediction $\hat{V}(\boldsymbol{s})$ to 1 and the context's typical return level $\tilde{R}(\boldsymbol{s})$ to 0. This can be achieved via

$$\overline{R_i} = \frac{R_i - \tilde{R}(\boldsymbol{s}_i)}{\hat{V}(\boldsymbol{s}_i) - \tilde{R}(\boldsymbol{s}_i)}.$$

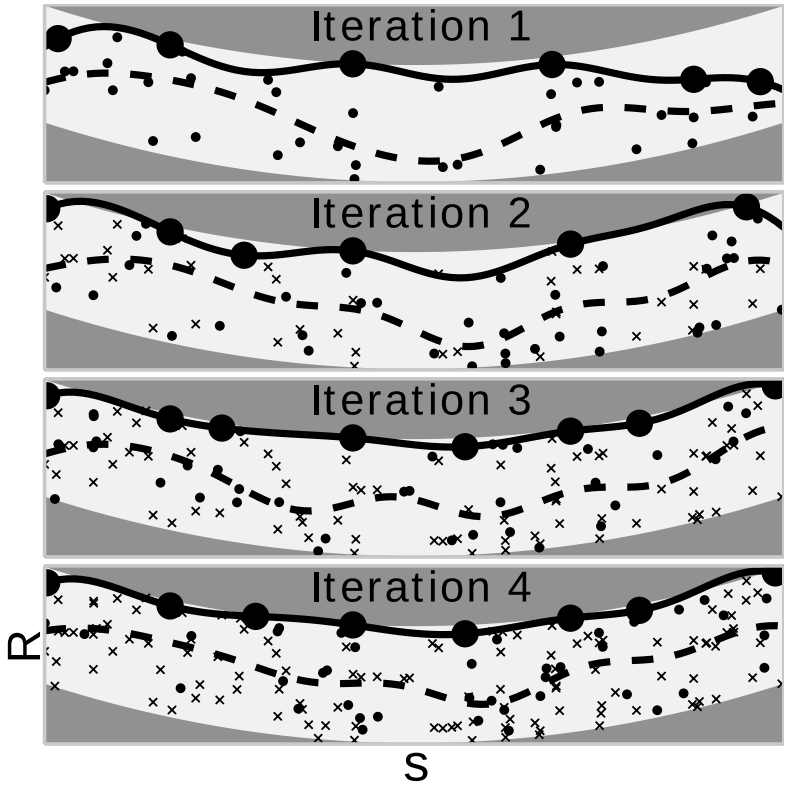


Figure 4.7.: Four iterations of incremental learning of the upper boundary $\hat{V}$ (solid line) and medium return level $\tilde{R}$ (dashed line). For each update of the PUBSVE, new samples (black dots) and support vectors of PUBSVE from the previous iteration (large circles) are used as training set. All previous samples that are not used for the incremental training are displayed as small crosses. $\tilde{R}$ is learned using an SVR on the whole data. New samples are drawn uniform randomly from the white background area.

Approximating a lower boundary on the returns cannot be performed analogously to estimating an upper boundary since contextual policy search tries to avoid sampling regions of low return and in some problems, no lower bound exists. Therefore, we normalize returns based on their estimated typical return level $\tilde{R}(\boldsymbol{s})$ obtained from a standard SVR [Dru+97] model trained on $\mathcal{D}$.

Since the PUBSVE needs to be updated when new samples arrive, an incremental training procedure is desirable. For this, instead of training the PUBSVE on the whole set $\mathcal{D} = \{(\boldsymbol{s}_i, R_i)\}_{i=1}^{n}$, we can update it incrementally [SLS99], where we forget every old example $(\boldsymbol{s}_i, R_i)$ except the support vectors $\boldsymbol{s}_i$ with $\alpha_i > 0$, collect new samples, and use the new samples and the retained support vectors to update $\hat{V}$. Note that this heuristic does not guarantee that the same results are obtained as in the non-iterative PUBSVE; however, it provides good results in practice. For the SVR model, we don't perform an

incremental update but keep the computational cost limited by subsampling the set $\mathcal{D}$ to a size of 5000. An illustration of this approach is given in Figure 4.7.

4.2.1.3. Training Set Selection for Contextual Policy Search

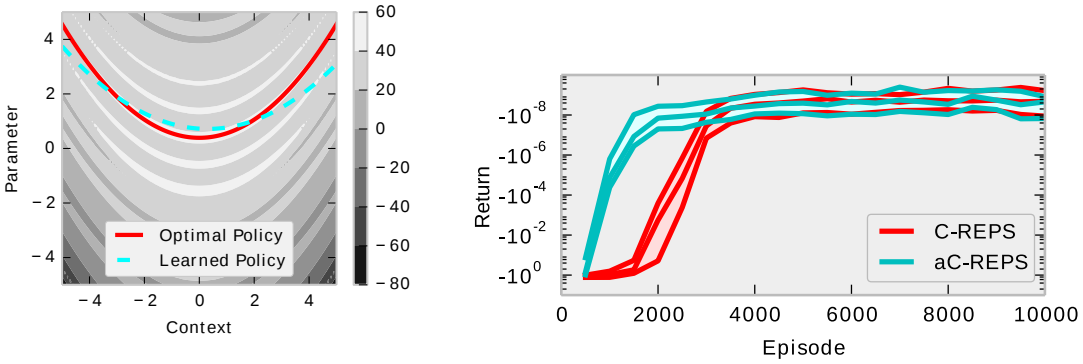
Contextual policy search methods such as C-REPS perform a search through the space of policies where updates of the policy are done such that we move in the direction of increasing expected return while, at the same time, bounding the loss of information between the observed data distribution and the data distribution generated by the new policy [PMA10]. Bounding the information loss results in small steps in policy space and avoids that a large step is taken into an unknown area of the policy space, which might contain policies whose execution is dangerous for a robot.

It remains to define the observed data distribution. Possible choices are the samples drawn from the last policy or from the last K policies [DNP13]. The former choice has the disadvantage that all prior experience not generated by the last policy is effectively forgotten and the latter choice slows down learning initially, as the new policy is enforced for K iterations to stay close to the data generated by the initial policies, which are typically behaving badly and strongly exploratory. In principle, it would be appealing if the old data distribution would consist of the *best* samples from the entire history, as this enforces the new policy to stay close to what has worked well in the past. Note that one potential risk of this strategy is premature convergence to local optima; it is thus important to maintain a sufficient level of exploration.

As discussed above, the main challenge for determining the best samples is the incommensurability of returns in different contexts. In the previous section we investigated a solution in a simplified setup, in which only a discrete set of K contexts exists and we can store the best N/K obtained returns for each context and use these for the update of the upper-level policy, where N is the number of examples for each update. As the method proposed in Section 4.2.1.1 allows to make the returns from different contexts comparable, we can now extend this training data selection to continuous context spaces by selecting the training examples with the highest normalized return for training. To avoid premature convergence and to be more robust to errors in the estimate $\hat{V}$, we suggest to use softmax for training set selection instead of the maximum, which means that each of the samples $(\boldsymbol{s}_i, \boldsymbol{\theta}_i, R_i)$ that we observed will be selected with probability

$$p(\boldsymbol{s}_i, \boldsymbol{\theta}_i, R_i) = \frac{\exp(\tau \overline{R_i})}{\sum_j \exp(\tau \overline{R_j})},$$

where $\tau = 10$ in our experiments and $\overline{R_i}$ is the normalized return (see Section 4.2.1.2). We call this active training set selection C-REPS (aC-REPS). Since we reuse the best samples from previous policies to estimate the reward baseline in C-REPS, we can say that aC-REPS is an off-policy version of C-REPS. We compare standard C-REPS and aC-REPS in the following sections.



(a) Contextual Rastrigin function. Function value are indicated by the background color.

(b) Learning curves for best configurations of standard C-REPS (update after 200 episodes, $\epsilon = 1$) and aC-REPS (update after 100 episodes, $\epsilon = 2$). Lines correspond to quartiles. The y-axis is logarithmic.

Figure 4.8.: Rastrigin benchmark function.

4.2.2. Experiments

With the following experiments we want to answer the question whether our algorithm aC-REPS is more sample-efficient than C-REPS. We will discuss three problems with increasing difficulty.

4.2.2.1. Benchmark Function

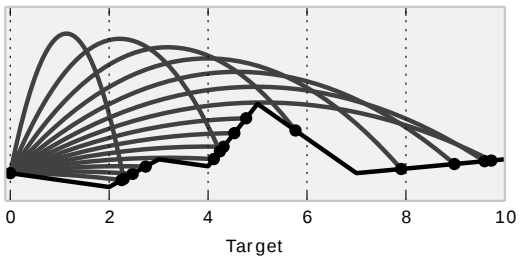
We evaluate both standard C-REPS and aC-REPS on a benchmark function. The solution π of the benchmark is a quadratic function and the upper return bound V^* is quadratic as well.

The contextual optimization objective is defined based on a function f from the BBOB testbed [Han+10]:

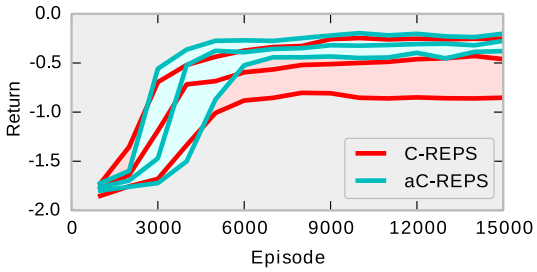
$$R(\boldsymbol{\theta}, \boldsymbol{s}) = -f(\boldsymbol{\theta}) + \frac{1}{10M}\boldsymbol{s}^T \boldsymbol{V}^* \boldsymbol{s}, \quad \text{where for all } i : -5 \leq \boldsymbol{\theta}_i \leq 5.$$

The parameters $\boldsymbol{W}_1, \ldots, \boldsymbol{W}_N \in \mathbb{R}^{M \times M}, \boldsymbol{V} \in \mathbb{R}^{M \times M}$, and $b_1 \ldots, b_N \in \mathbb{R}$ are generated randomly and scaled so that all context-dependent optima are within $[-5, 5]$. The optimum of the benchmark function depends on the context via $\boldsymbol{\theta}_i^*(\boldsymbol{s}) = \frac{1}{100M}\boldsymbol{s}^T \boldsymbol{W}_i \boldsymbol{s} + b_i$. In the following evaluation, we use the Rastrigin function f_3 from the BBOB testbed as f. The contextual objective function as well as the optimal π and a non-optimal solution are displayed in Figure 4.8a.

We use 200 samples for each policy update and set the minimum allowed temperature $\eta_{\text{min}} = 10^{-8}$; moreover, we use quadratic context transformation for the linear upper-level policy, bound the output of the upper-level policy to the interval $[-5, 5]$, and initialize the upper-level policy such that it generates zeros for all inputs. The parameter γ for the kernels of the boundary estimations is set to $10\overline{d}$, where $\overline{d}$ is the average distance of



(a) Illustration of the catapult problem: the objective is to shoot an object from a catapult situated at position 0 such that it hits a prespecified target position s. Several example trajectories are shown.



(b) Learning curves for standard C-REPS (update after 200 episodes, $\epsilon = 0.5$) and aC-REPS (update after 200 episodes, $\epsilon = 0.2$). The 3 lines correspond to the quartiles.

ϵ	0.1	0.2	0.5	1	2
Update after 50 samples	+	+	+	+	o
Update after 100 samples	+	+	+	+	o
Update after 200 samples	+	+	o	o	o

(c) Comparison of aC-REPS and C-REPS for several configurations after 7,500 episodes. We test statistical significance with a Wilcoxon signed-rank test. "+" means aC-REPS is significantly better and "o" means no significance.

Figure 4.9.: Catapult experiments.

training contexts. This choice is often not critical as long as it does not make the model too smooth. We are looking for a configuration that is stable on the one hand and fast on the other hand. To guarantee both, we select critical parameters by a grid search so that the performance after 5,000 episodes is optimized. There are two critical parameters of (a)C-REPS that have to be selected: the number of episodes between policy updates (options: 50, 100, 200) and the upper limit ϵ on the allowed Kullback-Leibler divergence between successive search distribution (options: 0.1, 0.2, 0.5, 1, 2).

We perform 50 runs with 10,000 episodes. The learning curves are displayed in Figure 4.8b. We can see that both methods reach a similar optimum, but the active training set selection (aC-REPS) is much faster in the beginning. There are configurations of C-REPS that reach the optimum similarly fast but these are unstable so that the algorithm diverges after finding a good solution.

4.2.2.2. Catapult Domain

Figure 4.9a shows the catapult problem introduced by da Silva et al. [dKB14]. The goal is to learn an upper-level policy that generates appropriate actions $\boldsymbol{\theta}_i$, consisting of the angle $\phi_i \in [0, \pi/2]$ and velocity $v_i \in [5, 10]$ of the catapult's shot, such that a specific

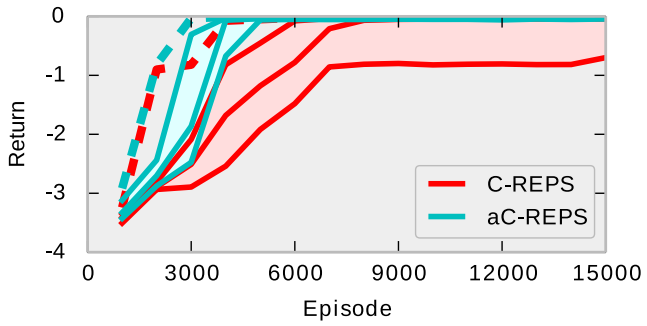


Figure 4.10.: Throwing experiments. Learning curves for best configurations of standard C-REPS (update after 200 episodes, $\epsilon = 2$) and aC-REPS (update after 50 episodes, $\epsilon = 2$). Sold lines correspond to quartiles and dashed lines correspond to the best values over all runs.

target position, the context $\boldsymbol{s}_i \in [2, 10]$, is hit. The target surface is unknown to the agent, which requires the agent to learn by trial and error. The return of an episode is computed as $R_i = -|\boldsymbol{s}_i - \boldsymbol{s}_h| - 0.5v_i$, where $\boldsymbol{s}_h$ is the position that was actually hit. Returns obtained in different contexts are not directly comparable, because certain target positions such as those behind the top of a hill are more challenging than others.

We use 200 samples for each policy update and set the minimum allowed temperature η_{min} to 10^{-8}; moreover, we employ a Nyström approximation [WS01] of an RBF kernel with $\gamma = 10^{-5}$ and 10 components as context features for the linear upper-level policy and initialize the upper-level policy such that it generates velocity 7.5 and angle $\pi/4$ for each context. The optimum parameter configuration is determined after 7,500 episodes similar to Section 4.2.2.1.

We perform 50 runs with 15,000 episodes. We found that aC-REPS is more robust with respect to parameter configuration which we demonstrate with a Wilcoxon signed-rank test in Table 4.9c. This property is desirable for learning, for instance, with robots because we would otherwise require a lot of episodes to find the optimum configuration. The learning curves in Figure 4.9b show that both variants of C-REPS improve initially; however, the best configuration of standard C-REPS is not as reliable as aC-REPS because there are some runs that do not improve beyond a moderate level of performance. aC-REPS does not exhibit this problem.

4.2.2.3. Ball Throwing

We use the simulated robot arm COMPI (see Appendix F.1 for details) with 6 degrees of freedom to generalize a throwing movement over a target area on the ground. The return is the negative distance to the target location. Targets are sampled from $\boldsymbol{s} \in [-3, 3] \times [3, 6]$. We use a DMP as defined by Ijspeert et al. [Ijs+13] with an execution time of 200 ms, a control frequency of 250 Hz, and 10 basis functions per dimension to generate the throwing motion in joint space. The ball is attached to the end effector and

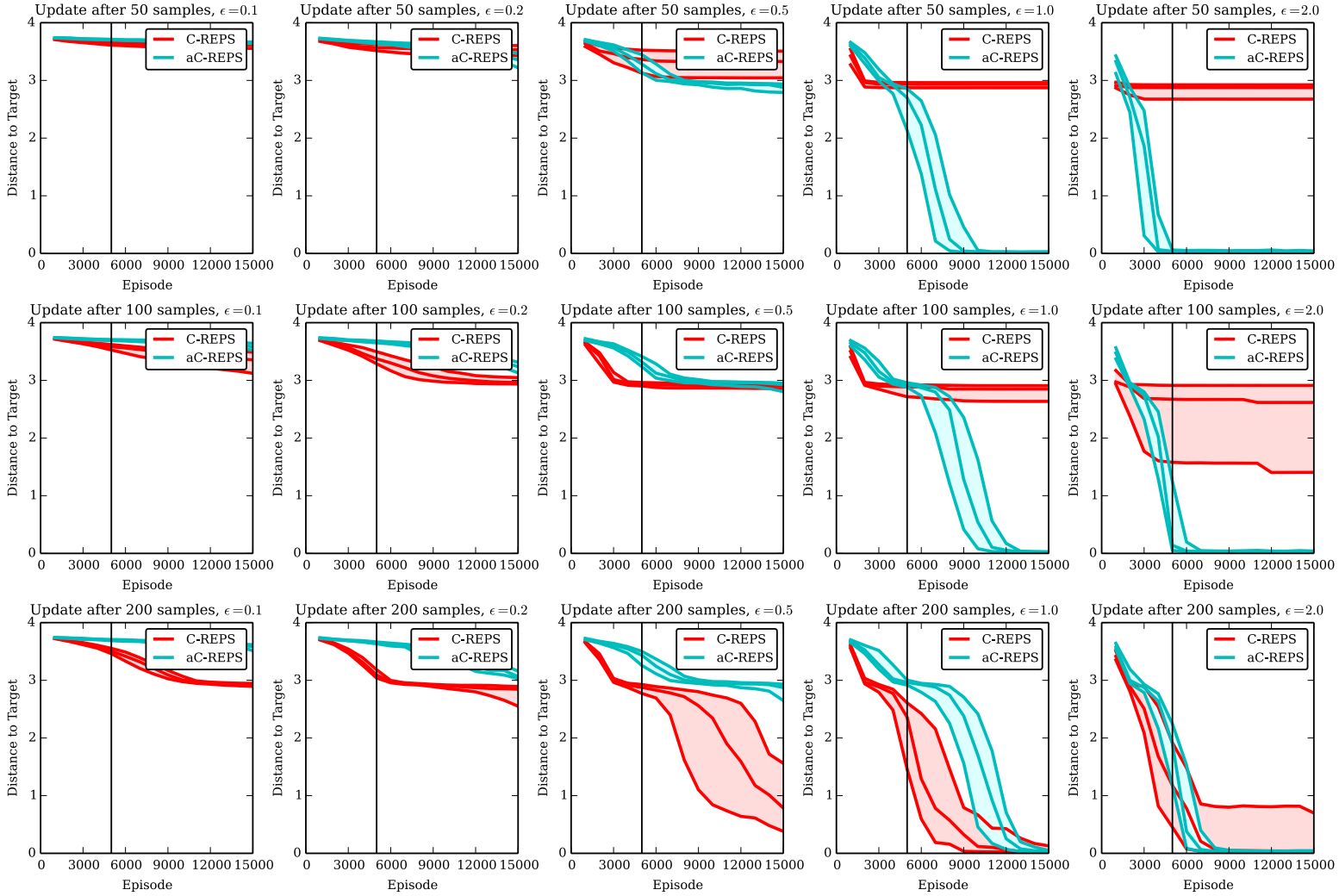


Figure 4.11.: Learning curves for ball throwing with varied hyperparameters. We plot the target distance on the y-axis.

is released after 140 ms. An initial throwing motion, which throws into the middle of the target area, has been learned with standard REPS [PMA10]. Only the 60 weights of the DMP are to be adapted. Each policy update is based on 200 samples and the minimum value allowed for $\eta_{\min}$ is 10^{-8}. We use quadratic features of the context.

In all experiments, we perform 30 independent runs and 15,000 episodes per run. We evaluate learning progress in an equally spaced grid of 100 test contexts

$$\boldsymbol{s} \in \{-2.7m, -2.1m, -1.5m, \ldots, 2.7m\} \times \{3.15m, 3.45m, 3.75m, \ldots, 5.85m\}.$$

Figure 4.11 shows the effect of hyperparameters. We varied ϵ and the number of samples before we update the upper-level policy. We can see that C-REPS is better for small ϵ and large number of samples before the update, which stabilizes training at the cost of slowing down the learning progress. $\epsilon > 2$ often leads to numerical problems in any version of the algorithm so we do not recommend it. Hence, aC-REPS seems to handle the largest possible value of ϵ well and stabilizes training over all independent runs. It is also able to update the upper-level policy more frequently, which accelerates the learning progress. Figure 4.10 compares the best hyperparameter configurations (measured after 5000 episodes). aC-REPS learns good policies fast and reliably. Standard C-REPS is slower and less reliable, which is consistent with the results of Sections 4.2.2.1 and 4.2.2.2.

4.2.3. Discussion

We propose the novel approach PUBSVE for addressing the incommensurability of returns in contextual policy search with contexts of different difficulties. This approach can be employed in contextual policy search and achieves more robust and faster learning by selecting high quality data for the policy update.

The PUBSVE by itself is an interesting approach since most approaches in regression are about estimating mean values or standard deviations, but the PUBSVE can be used to estimate the lower or upper bound of samples. This could not just be useful for contextual policy search in the future but also for other reinforcement learning algorithms. Similar loss functions could also be used to train other models such as neural networks.

We have seen that aC-REPS can not only improve the sample efficiency but also make the training process more reliable and consistent. Although it is impressive that aC-REPS is able to approximately halve the number of episodes required to reach a good median performance in ball throwing in comparison to C-REPS, this number is still in the range of a few thousands, which is too much to reach our objective of learning within a few hundred episodes.

4.3. Extensions from Black-box Optimization

This section was published originally as [Fab19a] and has been revised.

C-REPS and C-CMA-ES (see Section 2.3.5) are similar, although C-REPS has been developed in the field of reinforcement learning and C-CMA-ES is an extension of the black-box optimizer CMA-ES. There is more potential to transfer algorithms from black-box optimization to the reinforcement learning domain and to contextual policy search

in particular. In this section, we will investigate two common extensions of CMA-ES and apply them to C-CMA-ES. Note that also active training set selection could be used in C-CMA-ES, and elitist selection, which C-CMA-ES performs in each generation based on advantage values, is similar.

4.3.1. Proposed Extensions of C-CMA-ES

4.3.1.1. Active C-CMA-ES

Active CMA-ES (aCMA-ES) [JA06] modifies the covariance update of CMA-ES to take into account the worst samples of a generation and discourage exploration in their direction. Similar to the rank-μ update of the covariance with the best samples of a generation, we compute a matrix

$$\boldsymbol{S}^{-} \leftarrow \frac{1}{{\sigma^t}^2} \sum_{i=1}^{N} \left(\boldsymbol{\theta}_j - \boldsymbol{W}^t \phi(\boldsymbol{s}_j)\right) \cdot \boldsymbol{D}_{ii} \cdot \left(\boldsymbol{\theta}_j - \boldsymbol{W}^t \phi(\boldsymbol{s}_j)\right)^T ,$$

where $j = 1 + N - i$. $\boldsymbol{S}^{-}$ will be subtracted from the covariance. Line 29 of Algorithm 3 is replaced by

$$\boldsymbol{\Sigma}^{t+1} \leftarrow (1 - c_{1a} - c_\mu - \frac{1}{2}c_{\mu^-})\boldsymbol{\Sigma}^t + c_1 \boldsymbol{p}_c^{t+1} {\boldsymbol{p}_c^{t+1}}^T + \left(c_\mu + \frac{1}{2}c_{\mu^-}\right) \boldsymbol{S} - c_{\mu^-} \boldsymbol{S}^{-},$$

with $c_{\mu^-} = \frac{(1-c_\mu)\mu_{eff}}{4\left((n+n_s+2)^{1.5}+2\mu_{eff}\right)}$.

4.3.1.2. A Surrogate Model for C-CMA-ES

Another extension to CMA-ES is CMA-ES with a ranking SVM as surrogate model (ACM-ES) [LSS10]. The ranking support vector machine (ranking SVM) [Joa02] orders samples, hence, CMA is ordered alphabetically in the acronym of this extension. A ranking surrogate model fits well because CMA-ES only takes into account the ranking of samples in a generation. It does not consider actual returns. Assuming all samples are ordered by their rank, a ranking SVM finds

$$\begin{aligned} \min_{\boldsymbol{w}} \quad & \frac{1}{2}||w||_2^2 + \sum_{i=1}^{N} C_i \xi_i \\ \text{subject to} \quad & \boldsymbol{w}^T \left(\phi(\boldsymbol{x}_i) - \phi(\boldsymbol{x}_{i+1})\right) \geq 1 - \xi_i \wedge \xi_i \geq 0, \\ & \text{for all } i = 1, \ldots, N-1, \end{aligned}$$

for a nonlinear projection ϕ of the feature vectors $\boldsymbol{x}_i$ with corresponding ranking penalty terms ξ_i. $\boldsymbol{w}$ is a weight vector. This can be solved by a form of sequential minimal optimization [Pla98] and we can use the kernel trick to generate a nonlinear ranking function without explicitly specifying ϕ. We will use an RBF kernel

$$k(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\frac{(\boldsymbol{x} - \boldsymbol{x}')^2}{2\sigma^2}\right),$$

with σ set to the average distance between training samples. The cost of an error depends on the ranks of corresponding samples and is $C_i = 10^6(N-i)^{c_{pow}}$, where $c_{pow} = 2$.

Instead of ordering samples by their returns, we will order them by their advantage values (returns with subtracted context-dependent baseline, see Algorithm 3, line 21). We found this to be crucial in preliminary experiments.

Loshchilov et al. [LSS10] use the surrogate only conservatively to pre-screen a larger set of samples from which more promising samples are selected with a higher probability for evaluation. This is more difficult in a contextual setting because we have no control over the contexts in which we can evaluate samples. Once we know in which context we can evaluate the next sample, we could sample several parameter vectors $\boldsymbol{\theta}_i$ and select the most promising samples with higher probability for evaluation. In experiments that we conducted, this often caused preliminary convergence. Instead, we will exploit the surrogate model directly, that is, we will not use it for pre-screening but we will use predicted ranking values directly in the update step.

Another key idea of ACM-ES is to learn the surrogate model with normalized samples

$$\boldsymbol{\theta}' \leftarrow \boldsymbol{\Sigma}^{-\frac{1}{2}} (\boldsymbol{\theta} - \boldsymbol{\mu})$$

with respect to the covariance[1] and the mean of the search distribution. We adopt the idea of using a ranking SVM as a surrogate model with normalized samples for Contextual ACM-ES (C-ACM-ES). The surrogate model will also take into account the context $\boldsymbol{s}$ in addition to the parameters $\boldsymbol{\theta}$. The normalization is more complicated:

$$\boldsymbol{\theta}' \leftarrow \boldsymbol{\Sigma}^{-\frac{1}{2}} \left(\boldsymbol{\theta} - \boldsymbol{W}^T \phi(\boldsymbol{s})\right),$$

where the contexts of the training set will be normalized to have a mean of zero and a standard deviation of one. We assume that there is no correlation between context variables, which is not correct in general.

The search distribution is updated after N samples from the objective function. We store the last $40 + \lfloor 4d^{1.7} \rfloor$ samples to train the local surrogate model, where d is the number of parameters to be optimized. For each update of the search distribution, in addition to the N samples that we evaluated on the real objective function, we will draw $N' - N$ samples from the previous search distribution for random contexts that we observed in the training set and predict their ranking values to compute the update of the search distribution with these N' samples. Additional hyperparameters will be described and evaluated in Subsection 4.3.2.2.

Using a surrogate model does not decrease the computational complexity in comparison to C-CMA-ES. Our expectation is that it increases sample efficiency and, hence, the suitability for expensive objective functions.

[1] The inverse square root $\boldsymbol{\Sigma}^{-\frac{1}{2}} = \boldsymbol{B}\boldsymbol{D}^{-1}\boldsymbol{B}^T$ of the symmetric covariance matrix can be computed from the eigendecomposition $\boldsymbol{\Sigma} = \boldsymbol{B}(\boldsymbol{D}\boldsymbol{D})\boldsymbol{B}^T$.

4.3.2. Experiments

The questions that we want to answer with our experiments are: (1) Which hyperparameters should we use for our extensions? (2) Are our extensions more sample-efficient than existing contextual policy search algorithms? (3) Are they sample-efficient enough to learn on real robots?

4.3.2.1. Objective Functions for Contextual Black-box Optimization

Another idea that can be transferred from black-box optimization to the contextual setting is a set of standard benchmark functions. The analysis in this section is similar to the one of Abdolmaleki et al. [Abd+17a]. We use some additional objective functions. We take standard objective functions and make them contextual by defining $f_{\boldsymbol{s}}(\boldsymbol{\theta}) = f(\boldsymbol{\theta} + \boldsymbol{G}\phi(\boldsymbol{s}))$, where $\boldsymbol{G}$ is a matrix with components sampled iid from a standard normal distribution. If not stated otherwise, $n_s = 1$, $\phi(\boldsymbol{s}) = \boldsymbol{s}$, and the components of $\boldsymbol{s}$ are sampled from the interval $[1, 2)$.

To make results comparable to the results of Abdolmaleki et al. [Abd+17a], we use the same definition of $f_{Sphere}(\boldsymbol{\theta}) = \sum_{i=1}^{d} \theta_i^2$ and $f_{Rosenbrock}(\boldsymbol{\theta}) = \sum_{i=1}^{d-1} 100(\theta_i^2 - \theta_{i+1})^2 + (\theta_i - 1)^2$. In addition, we use the functions *ellipsoidal*, *discus*, *different powers* from the COCO benchmark platform for black-box optimization [Han+16], and

$$f_{Ackley}(\boldsymbol{\theta}) = -20 \exp\left(-0.2\sqrt{\frac{1}{d}\sum_{i=1}^{d}\theta_i^2}\right) + 20 - \exp\left(\frac{1}{d}\sum_{i=1}^{d}\cos(2\pi\theta_i)\right) + \exp(1).$$

The sphere objective evaluates the optimal convergence rate of an algorithm, the ellipsoidal function has a high conditioning that requires the algorithm to estimate the individual learning rates per dimension but is symmetric and separable, the Rosenbrock function checks whether the optimizer is able to change its direction multiple times, the discus function also has a high conditioning, the different powers function has no self-similarity, and the Ackley function is a multimodal function with many local minima. We do not use any other multi-modal function because contextual black-box optimization algorithms do not work well for this kind of problems.

Initial parameters are sampled from $\mathcal{N}(\boldsymbol{0}, \sigma_0^2 \boldsymbol{I})$ with $\sigma_0 = 1$ in most cases. Initial parameters of the Ackley function are sampled with $\sigma_0 = 14.5$ and the function has bounds at $[-32.5, 32.5]$. The number of dimensions of the parameter vector $\boldsymbol{\theta}$ is 20. N is set close to the smallest possible value that generates a stable learning progress. Unless otherwise stated, we use $N = 50$ samples of the objective function before we make an update of the search distribution. The number of updates corresponds to the number of iterations or generations in the following analysis. Instead of minimizing the $f_{\boldsymbol{s}}$, we will maximize $-f_{\boldsymbol{s}}$. Listed function values are averaged over one generation, that is, multiple contexts will be considered but not always the same contexts.

Table 4.1.: Comparison of hyperparameters. Performance is averaged over 20 runs.

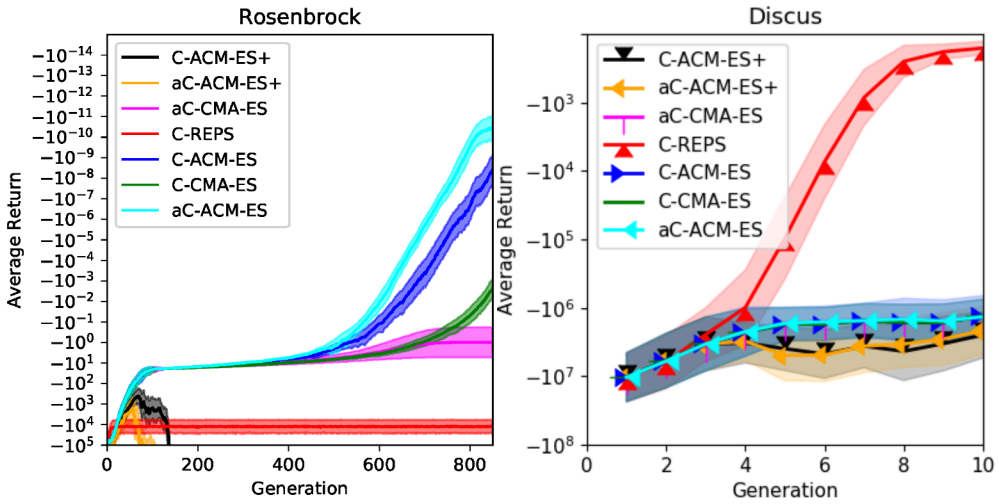
Hyperparameter	$-\frac{1}{\lvert S\rvert}\sum_{s\in S} f_s(x)$
Rosenbrock ($n_s = 1$), after generation 850	
$N' = 2N$	$\mathbf{-7.817 \cdot 10^{-10}}$
$N' = 3N$	$-1.485 \cdot 10^{-9}$
$N' = 5N$	$-4.089 \cdot 10^{-3}$
$N' = 10N$	$-1.445 \cdot 10^{15}$
$n_{iter} = 300$	$-1.679 \cdot 10^{-3}$
$n_{iter} = 1000$	$-1.485 \cdot 10^{-9}$
$n_{iter} = 300$	$-4.607 \cdot 10^{-13}$
$n_{iter} = 10000$	$\mathbf{-2.396 \cdot 10^{-14}}$
$c_{pow} = 1$	$\mathbf{-3.656 \cdot 10^{-9}}$
$c_{pow} = 2$	$-1.977 \cdot 10^{41}$
Ackley ($n_s = 1$), after generation 1100	
$n_{start} = 100$	$-1.411 \cdot 10^{1}$
$n_{start} = 300$	$-1.085 \cdot 10^{1}$
$n_{start} = 1000$	$-1.086 \cdot 10^{0}$
$n_{start} = 3000$	$\mathbf{-3.995 \cdot 10^{-9}}$
$n_{start} = 10000$	$-1.155 \cdot 10^{-8}$

4.3.2.2. Hyperparameters of C-ACM-ES

First, we look for a good configuration of C-ACM-ES. There are several hyperparameters: we have to define after how many samples the surrogate model is accurate enough to be used (n_{start}), the number of samples N' evaluated by the surrogate model, c_{pow} of the ranking SVM objective, and the number of iterations n_{iter} that will be used to optimize the ranking SVM per training sample. While one parameter has been investigated, the others were kept to the values $N' = 3N$, $n_{start} = 100$, $c_{pow} = 1$, $n_{iter} = 1000$. We found that n_{start} is particularly important for optimizing f_{Ackley} and c_{pow}, n_{iter}, and N' are critical for $f_{Rosenbrock}$.

The most important results of the performed experiments are shown in Table 4.1. Setting $N' = 2N$ gives the best result for the Rosenbrock function, but $N' = 3N$ is a better compromise between exploitation of the model and a conservative handling. There are some objectives, where we can much better exploit the model. In the following experiments, we will use the configurations $N' = 3N$ and $N' = 10N$.

Larger values for n_{iter} improve the result, which is not surprising. This works particularly well on a complex function such as the Rosenbrock function. As a compromise between computational overhead and sample efficiency, we select $n_{iter} = 1000$ which



(a) Rosenbrock function.

(b) Discus function, first ten generations.

Figure 4.12.: Learning curves of several contextual policy search methods. Mean and standard deviation of 20 experiments are displayed on logarithmic y-axes.

seems to work well. $n_{iter} = 3000$ also does not seem to be a bad choice, as it considerably improves the result on the Rosenbrock function and only increases computational cost by a factor of three. On the Ackley function, it is important to bring the search distribution in a good state in which we can learn an accurate surrogate model before we start exploiting the surrogate. $n_{start} = 3000$ is the best parameter here. In addition, we will use an aggressive version in the following experiments and set $n_{start} = 100$. The effect of c_{pow} is quite interesting. Although in the original implementation for ACM-ES [LSS10] the default value is 2, this seems to have a catastrophic effect on the Rosenbrock function. For all other functions, differences are negligible. During all conducted experiments, we found the estimate of the context-dependent baseline and the surrogate model to be critical for C-ACM-ES.

4.3.2.3. Comparison of C-CMA-ES Extensions

We did an extensive evaluation of several combinations of extensions of C-CMA-ES. Results are displayed in Table 4.2. NaN indicates divergence. We use C-REPS with the hyperparameter $\epsilon = 1$ and C-CMA-ES as baselines. Note that it is better to set ϵ for C-REPS as large as possible to learn as fast as possible, however, the algorithm becomes numerically unstable if ϵ is too large. $\epsilon = 1$ is a good compromise. In more real-world scenarios, setting ϵ to such a large value (or setting the initial step size of CMA-ES to a large value) could result in dangerous exploration. The term aC-CMA-ES refers to active C-CMA-ES, C-ACM-ES uses the surrogate model, and aC-ACM-ES is its active counterpart. "+" indicates that the surrogate model is exploited aggressively, that is, we

Table 4.2.: Average performance of algorithms (20 runs, best option is underlined).

Method	$-\frac{1}{\lvert S\rvert}\sum_{\boldsymbol{s}\in S} f_{\boldsymbol{s}}(x)$	Method	$-\frac{1}{\lvert S\rvert}\sum_{\boldsymbol{s}\in S} f_{\boldsymbol{s}}(x)$
Sphere ($n_s = 2$), after generation 200		Ellipsoidal ($n_s = 1$), after generation 800	
C-REPS	$-4.509 \cdot 10^{+01}$	C-REPS	$-2.944 \cdot 10^{+05}$
C-CMA-ES	$-1.815 \cdot 10^{-05}$	C-CMA-ES	$-2.337 \cdot 10^{+02}$
aC-CMA-ES	$-1.348 \cdot 10^{-05}$	aC-CMA-ES	$-1.524 \cdot 10^{+02}$
C-ACM-ES+	$-1.294 \cdot 10^{-08}$	C-ACM-ES+	$-1.300 \cdot 10^{+16}$
aC-ACM-ES+	$-1.506 \cdot 10^{-01}$	aC-ACM-ES+	$-2.407 \cdot 10^{+18}$
C-ACM-ES	$-6.257 \cdot 10^{-04}$	C-ACM-ES	$-1.039 \cdot 10^{-10}$
aC-ACM-ES	$-2.309 \cdot 10^{-04}$	aC-ACM-ES	$-2.388 \cdot 10^{-11}$
Rosenbrock ($n_s = 1$), after generation 850		Diff. Powers ($n_s = 1$), after generation 600	
C-REPS	$-1.255 \cdot 10^{+04}$	C-REPS	$-9.088 \cdot 10^{+02}$
C-CMA-ES	$-2.328 \cdot 10^{-03}$	C-CMA-ES	$-1.562 \cdot 10^{-07}$
aC-CMA-ES	$-9.736 \cdot 10^{-01}$	aC-CMA-ES	$-3.038 \cdot 10^{-07}$
C-ACM-ES+	$-1.445 \cdot 10^{+15}$	C-ACM-ES+	$-7.111 \cdot 10^{+74}$
aC-ACM-ES+	$-3.227 \cdot 10^{+19}$	aC-ACM-ES+	$-8.717 \cdot 10^{+82}$
C-ACM-ES	$-3.656 \cdot 10^{-09}$	C-ACM-ES	$-2.464 \cdot 10^{-14}$
aC-ACM-ES	$-3.899 \cdot 10^{-11}$	aC-ACM-ES	$-1.284 \cdot 10^{-14}$
Ackley ($n_s = 1$), after generation 1100		Discus ($n_s = 1$), after generation 850	
C-REPS	$-1.947 \cdot 10^{+01}$	C-REPS	$-1.288 \cdot 10^{+02}$
C-CMA-ES	$-8.762 \cdot 10^{-07}$	C-CMA-ES	$-2.995 \cdot 10^{-10}$
aC-CMA-ES	$-8.773 \cdot 10^{-07}$	aC-CMA-ES	$-3.838 \cdot 10^{-10}$
C-ACM-ES+	NaN	C-ACM-ES+	$-8.297 \cdot 10^{+27}$
aC-ACM-ES+	NaN	aC-ACM-ES+	$-1.250 \cdot 10^{+24}$
C-ACM-ES	$-3.995 \cdot 10^{-09}$	C-ACM-ES	$-8.877 \cdot 10^{-12}$
aC-ACM-ES	$-1.813 \cdot 10^{-08}$	aC-ACM-ES	$-1.684 \cdot 10^{-11}$

set $N' = 10N$ and $n_{start} = 100$, otherwise $N' = 3N$ and $n_{start} = 3000$. Exemplary learning curves are displayed in Figure 4.12a for the Rosenbrock function.

Variants of C-ACM-ES mostly outperform vanilla C-CMA-ES. Although the surrogate model focuses on ordering the samples with the highest rank more correctly and aC-CMA-ES is often not better than C-CMA-ES, aC-ACM-ES performs best in most cases. If this is not the case, C-ACM-ES performs best. The sphere function with two context variables seems to be different. Here, it is important to exploit the surrogate model as early and aggressively as possible to have a chance against C-CMA-ES.

An interesting result, however, is that C-REPS is often much faster in the early phase. See, for example, Figure 4.12b: in the first 10 generations, which amounts to 500 episodes, C-REPS outperforms all variants of C-CMA-ES by orders of magnitude. Unfortunately this is the phase of the optimization that is interesting for learning in the real world. We

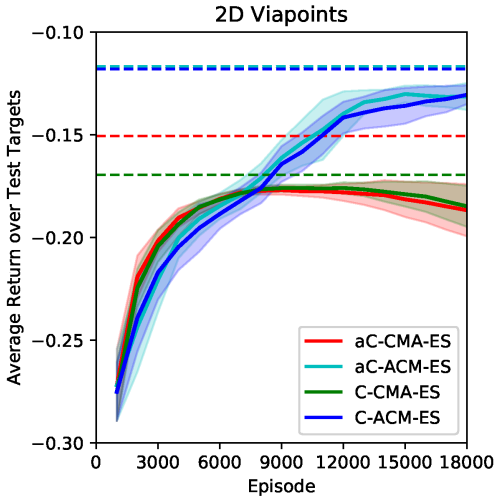


Figure 4.13.: Learning curves for the via-point problem averaged over 20 experiments. Dashed lines indicate the maximum return over all experiments.

can also see that C-REPS converges already and does not learn anything for the next 840 generations. Variants of C-CMA-ES will continue making progress until the last episode. Because the surrogate model is only useful when we have a good estimate of the covariance matrix and a substantial amount of samples from the objective function, we can only use it in later stages of the optimization to improve the learning progress.

4.3.2.4. 2D Via-Point Problem

We will evaluate C-CMA-ES variants on a 2D via-point problem. A DMP with $\boldsymbol{x}_0 = (0,0)^T$, $\boldsymbol{g} = (1,1)^T$, $\tau = 1.0$, and ten weights per dimension will be used as trajectory representation. The reward is defined for each step $t = 0, \ldots, T-1$ as $r_t = -0.001||\boldsymbol{v}||_t$, and for the last step as the distance to each via point $r_T = -\sum_{\boldsymbol{p}_{via,t}} ||\boldsymbol{p}_{via,t} - \boldsymbol{p}_t||$. Via points are defined as a tuple of time and positions $\{\left(0.2, (0.2, 0.5)^T\right), (0.5, \boldsymbol{s})\}$, where $\boldsymbol{s} \in [0.3, 0.7] \times [0.3, 0.7]$.

We did not use C-REPS as a baseline because it is not robust against the choice of its hyperparameter ϵ. In the experiments, we use $N = 100$, $n_{start} = 1000$, and a quadratic baseline. Learning curves are displayed in Figure 4.13. The performance is evaluated on a grid of 25 test contexts, where $\boldsymbol{s}_1, \boldsymbol{s}_2 \in (0.3, 0.4, 0.5, 0.6, 0.7)$. Active C-CMA-ES does not have an advantage. The convergence behavior of (a)C-ACM-ES is much better, however, the advantage only occurs after about 8,000 episodes, which is too late for such a simple problem from the robotics perspective.

4.3.3. Discussion

We demonstrate that the extensions active C-CMA-ES and C-ACM-ES can be combined and yield impressive results on contextual function optimization problems in comparison

to C-CMA-ES. We show, however, that these results are actually not directly transferable to the domain of robotics. We would like to learn successful upper-level policies in a few hundred episodes at maximum. The presented extensions, however, start to be better than standard C-CMA-ES just after the range of interest. They exhibit much better convergence behavior though.

In this section, we investigate a surrogate model for the first time in this thesis. We see that black-box optimization in combination with a good surrogate model can be more sample-efficient, however, it would be desirable to have a surrogate model that can be exploited after only a few episodes.

4.4. Bayesian Optimization for Contextual Policy Search

Parts of this section were published originally as [MFH15; Gut+18] and have been revised.

Contextual policy search is often based on local search approaches. From the field of black-box optimization, it is well-known that local search approaches are well suited for problems with a moderate dimensionality and no gradient-information. For the special case of relatively low-dimensional search spaces combined with an expensive cost function, which limits the number of evaluations of the cost functions, global search approaches with a surrogate model such as Bayesian optimization [BCd10] are usually superior. Policy search with pre-trained movement primitives can also fall into this category, as evaluating the cost function requires an execution of the behavior on the robot while only a small set of metaparameters might have to be adapted. In this section, we will apply Bayesian optimization to contextual policy search.

Bayesian optimization has been used for non-contextual policy search on locomotion tasks [Liz+07; Cal+16] and robot grasping [Kro+10] before. Calandra et al. [Cal+16] optimize eight parameters of a finite state machine that controls bipedal walking of a small robot. About 20 to 80 episodes are required for this task, which suggests that Bayesian optimization can be very sample-efficient.

4.4.1. Extension of Bayesian Optimization to Contextual Policy Search

> Note that the algorithm BO-CPS, which is presented in detail in this section, is not my contribution. It was developed by Jan Hendrik Metzen and published as Metzen et al. [MFH15]. I did the experimental evaluation of the approach in simulation, which will be presented in the next section.

BO-CPS [MFH15] internally learns a model of the return $\boldsymbol{R}$ of a parameter vector $\boldsymbol{\theta}$ in a context $\boldsymbol{s}$ with Gaussian process regression (GPR) [RW05]. BO-CPS is summarized in Algorithm 5. The GPR is trained from sample returns $\boldsymbol{R}$ obtained in episodes at query points consisting of a context $\boldsymbol{s}$ determined by the environment and a parameter vector $\boldsymbol{\theta}$ selected by BO-CPS. By learning a joint model over the context-parameter space, experience collected in one context is naturally generalized to similar contexts.

The GPR provides both an estimate of the expected return $\mu_{GP}[R(\boldsymbol{s}, \boldsymbol{\theta})]$ and the uncertainty $\sigma_{GP}[R(\boldsymbol{s}, \boldsymbol{\theta})]$ of this estimate. The uncertainty contains a homoscedastic

Algorithm 5 BO-CPS

Require: kernel for Gaussian process regression $k(\boldsymbol{x}, \boldsymbol{y})$, dataset $\mathcal{D} = \emptyset$, trade-off parameter for exploration and exploitation κ, number of random samples n_{random} before surrogate model is used

$t \leftarrow 1$
while not converged **do**
  Observe $\boldsymbol{s}$
  if surrogate model is initialized **then**
    $\boldsymbol{\theta} \leftarrow \arg\max_{\boldsymbol{\theta}} \mu_{GP}[R(\boldsymbol{s}, \boldsymbol{\theta})] + \kappa \sigma_{GP}[R(\boldsymbol{s}, \boldsymbol{\theta})]$ ▷ Optimize GP-UCB
  else
    Sample $\boldsymbol{\theta}$ randomly
  end if
  Obtain $R(\boldsymbol{s}, \boldsymbol{\theta})$ ▷ Return of policy $\pi_{\boldsymbol{\theta}}$ in environment with context $\boldsymbol{s}$
  $\mathcal{D} \leftarrow \mathcal{D} \cup (\boldsymbol{s}, \boldsymbol{\theta}, R(\boldsymbol{s}, \boldsymbol{\theta}))$
  if $t \geq n_{\text{random}}$ **then**
    Train surrogate model on $\mathcal{D}$ to predict return from context and parameters
  end if
  $t \leftarrow t + 1$
end while

aleatoric and an epistemic component. Based on this information, the parameter vector for the given context is selected by maximizing a so-called acquisition function. These acquisition functions allow controlling the trade-off between exploitation (selecting parameters with maximum estimated return) and exploration (selecting parameters with high uncertainty). Common acquisition functions in Bayesian optimization are the probability of improvement (PI) and the expected improvement (EI) [BCd10]. PI and EI, however, both require the definition of an incumbent, which denotes the parameter vector with the maximum observed return. This notion of an incumbent is not extended easily to contextual policy search, where the incumbent is different for every context, returns are noisy, and each context will only be sampled once because of the continuous context space. Thus, we use the acquisition function Gaussian process Upper Confidence Bound (GP-UCB) [Sri+10] instead, which does not require the definition of the incumbent. We define GP-UCB of a parameter vector in a context as

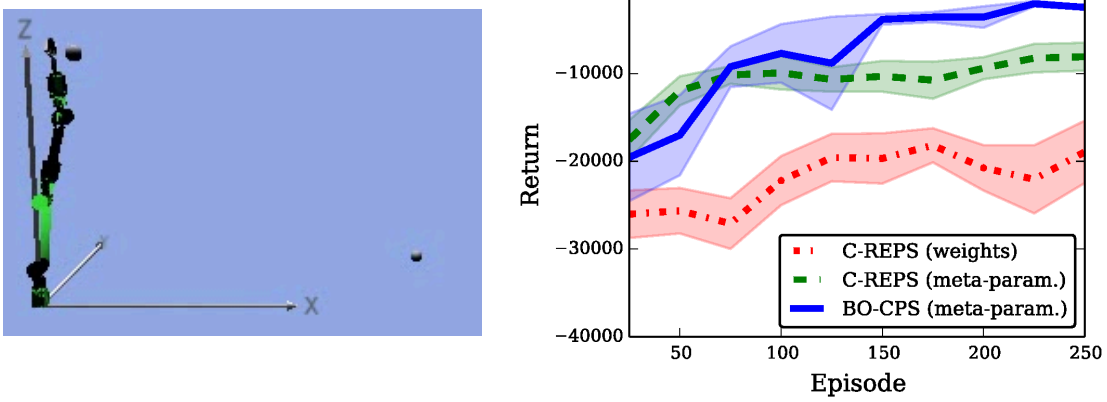
$$\text{GP-UCB}(\boldsymbol{s}, \boldsymbol{\theta}) = \mu_{GP}[R(\boldsymbol{s}, \boldsymbol{\theta})] + \kappa \sigma_{GP}[R(\boldsymbol{s}, \boldsymbol{\theta})],$$

where κ controls the exploration-exploitation trade-off.

BO-CPS selects parameters for a given context $\boldsymbol{s}$ by choosing

$$\boldsymbol{\theta} \leftarrow \arg\max_{\boldsymbol{\theta}} \text{GP-UCB}(\boldsymbol{s}, \boldsymbol{\theta}).$$

This maximization is performed by using the global optimizer DIRECT [JPS93] to find the approximate global maximum, followed by L-BFGS-B [Byr+95a] to refine it. Note that this maximization is performed only over the parameter space for fixed context $\boldsymbol{s}$.



(a) Simulated robot arm COMPI.

(b) Learning curves: mean and standard error for test contexts $\{-1m, -0.33m, 0.33m, 1m\} \times \{1m, 1.5m, 2m, 2.5m\}$ after intervals of 25 episodes.

Figure 4.14.: Simulated ball-throwing experiments with BO-CPS.

4.4.2. Experiments

In the following experiments we will investigate the sample efficiency of BO-CPS and its suitability for learning on a real robot.

4.4.2.1. Simulated Ball Throwing

We present results in a simulated robotic control task, in which the robot arm COMPI (see Figure 4.14a and Appendix F.1 for details) is used to throw a ball at a target on the ground encoded in a context vector. The target area is $[1m, 2.5m] \times [-1m, 1m]$ as shown in Figure 4.15. We compare three learning approaches, which all start from a manually initialized DMP where the start and goal position are defined in joint space and the weights are set to zero: (1) C-REPS learning all weights of the DMP, resulting in a high-dimensional search space with 300 parameters, (2) C-REPS learning only two metaparameters, and (3) BO-CPS learning the same two metaparameters. These two metaparameters are the execution time τ of the DMP, which determines how far the ball is thrown, and the final angle g_0 of the first joint, which determines the rotation of the arm around the z-axis, that is, $\boldsymbol{\theta} = (g_0, \tau)^T$.

For (1), we use a Gaussian upper-level policy to map contexts to DMP weights, with the mean being a quadratic function of the context and initial variance 100. For (2), we also use a quadratic Gaussian upper-level policy to map contexts to metaparameters with an initial $\boldsymbol{\theta}_0 = (0, 1)^T$, initial variance $\left((0.1\pi)^2, 1^2\right)$, and parameter space $g_0 \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$ and $\tau \in [0.2, 5]$. In (1) and (2), the upper level policy is updated after 25 episodes based on the last 100 samples. For (3), an anisotropic RBF kernel (a length scale for each dimension) is used for the GPR and GP-UCB's exploration parameter is set to $\kappa = 1.0$.

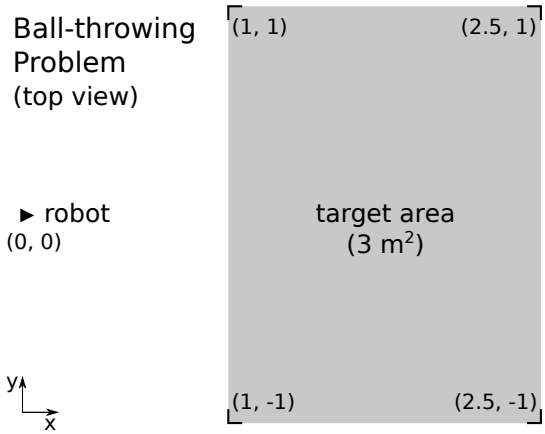


Figure 4.15.: Sketch of the ball throwing problem.

The return $R(\boldsymbol{s}, \boldsymbol{\theta}) = -||\boldsymbol{s} - \boldsymbol{b}_{\pi_{\boldsymbol{\theta}}}||_2^2 - 0.01 \sum_t \dot{\boldsymbol{q}}_t^2$ includes the squared distance between the goal $\boldsymbol{s}$ and the position $\boldsymbol{b}_{\pi_{\boldsymbol{\theta}}}$ hit by the ball with policy $\pi_{\boldsymbol{\theta}}$ (both in centimeters) as well as the squared sum of joint velocities during DMP execution.

Figure 4.14b shows the corresponding learning curves: although C-REPS performs initially better than BO-CPS, BO-CPS obtains a considerably higher return than C-REPS after 150 episodes. Learning only metaparameters is considerably faster than learning all weights.

4.4.2.2. Ball Throwing on a Real Robot

Jonas Hansen did this experimental evaluation of BO-CPS on a real robot in his master's thesis [Han15], which I supervised. This work is published with Gutzeit et al. [Gut+18]. I will present his work here as the results are essential for the following section and to achieve our goal of sample-efficient contextual policy search.

In the experiment presented by Gutzeit et al. [Gut+18] ball throwing is learned directly in reality without any simulation. Similar but not directly comparable works for ball-throwing have been published by da Silva et al. [da +14] and Ude et al. [Ude+10].

Similarly to the previous experiments in simulation, from an initial motion plan we generalize to a behavior template that can hit any target $\boldsymbol{s}$ (context) in the area $[1\ m, 2.5\ m] \times [-1\ m, 1\ m]$, where COMPI (see Figure 4.16) is located at $(0\ m, 0\ m)$. The performance is evaluated on a grid of 16 test contexts

$$\{-0.7m, -0.2m, 0.2m, 0.7m\} \times \{1.3m, 1.6m, 1.9m, 2.2m\}$$

that are not be explored during training.

The maximum reachable accuracy of COMPI was determined previously for four manually designed throws. The standard deviation of the touchdown position in 20 experiments per throw was measured. The mean positions were 1.3 m to 2.3 m away from

Figure 4.16.: The robotic arm COMPI with a ball.

COMPI. The standard deviations were about 4.5 cm to 7 cm. To measure the position where the ball hits the ground we use a motion capture system that recognizes the ball.

We optimize execution time and goal position of the first joint. Note that there is no direct linear relation between the goal position of the first joint and the throwing angle because of a complex interaction between the deformable ball and the scoop. It depends on the execution time as well. We conduct a thorough evaluation that compares BO-CPS with the GP-UCB acquisition function and entropy search [Met15], which an active context selection approach.

Figure 4.17 shows a learning curve. The return (negative squared distance to the target) and distance to the target over the test contexts are displayed. After about 80 episodes the performance will not increase much. Therefore, we measure the final performance after 80 episodes in five experiments per acquisition function. The average measured distance to the test targets are 13.46 cm, 15.24 cm, 22.62 cm, 17.40 cm, and 29.99 cm (mean 19.74 cm, median 17.4 cm) for GP-UCB and 6.54 cm, 14.74 cm, 6.68 cm, 8.96 cm, and 8.15 cm (mean 9.01 cm, median 8.15 cm) for entropy search. This evaluation clearly shows that it is possible to generalize throwing movements to a large area in only 80 episodes so that the average deviation from the target is almost at the maximum achievable precision. In addition, the computationally more complex entropy search shows consistently better results than upper confidence bound.

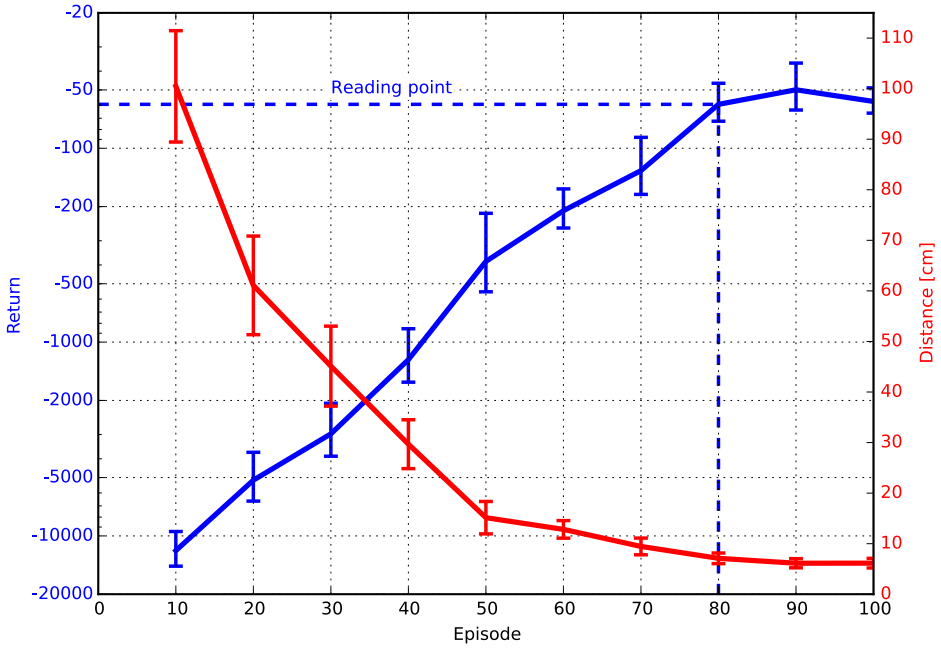


Figure 4.17.: Learning curve of contextual policy search (BO-CPS with entropy search) for real ball-throwing. The logarithmic y scale on the left represents the mean return and the linear y scale on the right represents the mean distance between the touchdown position and the context. Mean and standard error are measured over 16 test contexts. The reading point indicates where we measure the final performance. (Created by Jonas Hansen for [Han15].)

4.4.3. Discussion

4.4.3.1. Summary

Bayesian optimization is a sample-efficient black-box optimizer that can be used to learn difficult tasks such as throwing a ball and generalize it to a given context space with fewer than one hundred episodes. The main drawback of BO-CPS is its computational complexity. It is required to reduce the number of policy parameters and episodes to a small amount, which we did manually for the experiments in this section. It would be better to extract relevant policy parameters automatically. Note that reducing the number of parameters does not reduce the asymptotic computational complexity though. It merely further improves the sample efficiency of BO-CPS so that we do not need as many samples as are required to slow down the update step of the algorithm to a level that is not useful for contextual policy search anymore. Let us take a closer look at the asymptotic computational complexity of BO-CPS.

4.4.3.2. Computational Complexity

Let n_{opt} be the number of optimization steps on the acquisition function, t be the number of episodes, and n_{hyper} be the number of hyperparameter optimization steps. Then the asymptotic time complexity of querying a new parameter vector is $\mathcal{O}(n_{\text{opt}}t^2)$ for the GP-UCB acquisition function. The complexity of updating the model is $\mathcal{O}(n_{\text{hyper}}t^3)$.

Updating the Gaussian process regression model requires hyperparameter optimization. In each of the n_{hyper} optimization steps we recompute the symmetric Gram matrix of size t^2. Computing this matrix is of complexity $\mathcal{O}(dt^2)$, where d is the dimensionality of the data, in our case the sum of the numbers of context and parameter dimensions. The Cholesky decomposition of this matrix will be computed in $\mathcal{O}(t^3)$, which dominates the complexity of other steps in this procedure. In our application we set $n_{\text{hyper}} > t$ so that we can effectively say the complexity is $\mathcal{O}(t^4)$.

Querying a new parameter vector involves optimizing the GP-UCB acquisition function. This requires computing in each of the n_{opt} steps the standard deviation of the Gaussian process, which requires the computation of $\boldsymbol{k}^T\boldsymbol{K}^{-1}\boldsymbol{k}$ with $\boldsymbol{k} \in \mathbb{R}^t$ and $\boldsymbol{K} \in \mathbb{R}^{t\times t}$ of complexity $\mathcal{O}(t^2)$. Entropy search [MFH15] is even more expensive than GP-UCB.

4.5. Variational Trajectory Autoencoder

This section was submitted originally as [FK20] and has been revised.

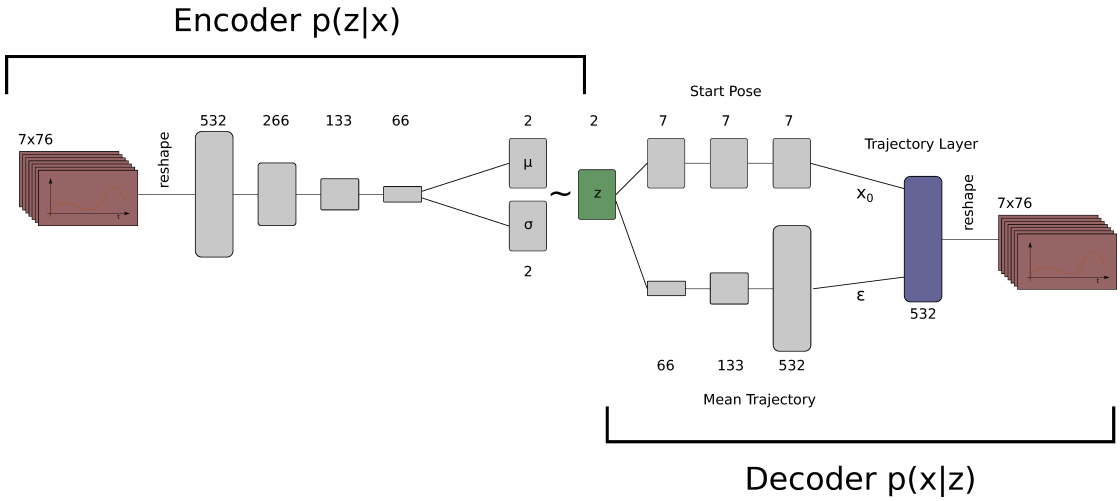


Figure 4.18.: Architecture of the VTAE. The encoder reduces the dimensionality of trajectories. The decoder generates trajectories from a low-dimensional representation. The decoder has a special structure that splits the initial state from the shape of the trajectory. Both are combined in the trajectory layer.

The space of possible trajectories is huge. Let us assume that we want to create a trajectory of 100 steps for a robot with three joints. We would have to generate 300 continuous values. This results in a large search space that contains far more possible trajectories than required to solve a specific task. We want to reduce the search space

for policies to a low number of dimensions and make useless trajectories less likely. This can be achieved by manifold learning from demonstrated trajectories. The manifold learning algorithm should be able to figure out which latent variables are best suited to cover the distribution of the demonstrated motions. We can then apply BO-CPS to the low-dimensional search space.

Problems such as playing table tennis [Kob+10] or darts [Kob+12], have often been handled with domain-specific policy representations such as DMPs [Ijs+13] in combination with policy search. These methods mostly use linear or non-parametric models. We are also interested in building the bridge between these approaches and deep learning, which learns nonlinear, parametric models. While neural networks are good at robot perception, there is a lack of approaches that generate complex movement patterns. Without a tight, reactive coupling of sensor measurements and actions it is difficult to generate complex trajectories, which is a problem that we will address in this section.

We will see how we can learn a trajectory generator that outputs dense high-dimensional trajectories from a low-dimensional latent space. The generator is a decoder extracted from a variational autoencoder (VAE) trained on a specific kind of trajectories that are demonstrated by an expert. We will discuss a new building block for neural networks: a trajectory layer that enforces smoothness on generated trajectories. The trajectory generator can be used to perform sample-efficient policy search with Bayesian optimization. We show this by applying the approach to robotic grasping on a real system.

4.5.1. Related Work: Manifold Learning for Behavior Learning

The manifold assumption is a fundamental idea in machine learning: "(high-dimensional) data lie (roughly) on a low-dimensional manifold." [CSZ06, page 6] Manifold learning tries to identify this low-dimensional manifold to circumvent the curse of dimensionality that would require us to collect a number of samples that exponentially increases with the number of dimensions.

Manifold learning for action generation has a biological motivation. A concept that has been widely adopted to explain how animals control complex motor systems are muscle synergies [SFS98; TSB99; dSB03]. Muscle synergies are defined as coherent activations of a group of muscles in space or time [dSB03]. They are building blocks of motor behaviors and a small number of these can be combined to form complex motor behaviors. Hence, low-dimensional representation of complex high-dimensional movement patterns is also present in animals that have to deal with a high number of individual motor units. This motivated research in the direction of manifold learning for robot control [Rue+15].

Several previous works combine manifold learning and dimensionality reduction with imitation or reinforcement learning. A survey of dimensionality reduction in learning for manipulation and control has been written by Ficuciello et al. [FFC18]. We will give a short introduction here.

The work presented in this section is inspired by Matsubara et al. [MHM10] who introduce one of the earliest methods that apply dimensionality reduction to state space trajectories in imitation learning: SDMPs. SDMPs reduce the dimensionality of DMPs learned from multiple demonstrations via singular value decomposition (SVD). Mat-

subara et al. [MHM10] use SDMPs to identify and represent multiple styles of similar motions. The shortcomings of SDMPs are that they are deeply coupled with DMPs and they exclude metaparameters from the dimensionality reduction. Furthermore, an SVD only allows linear transformations. A similar approach based on principal component analysis (PCA) [Pea01] has been suggested by Tosatto et al. [TSP20] for parameters of ProMPs and Rueckert et al. [Rue+15] present an extension of ProMPs with hierarchical priors to learn a low number of control parameters for trajectories from multiple demonstrations. They were also able to represent multi-modal distributions.

As previously stated, black-box optimization and policy search are closely connected. Bayesian optimization can be used for policy search, but it is difficult to scale it to many parameters. Parameter reduction has been used for this reason in high-dimensional problems by Wang et al. [Wan+16b], who use random embeddings to solve high-dimensional optimization problems with low intrinsic dimensionality. The idea of using fixed low-dimensional representations of high-dimensional policies has also been employed successfully in neuroevolution [Kou+13; Fab+13].

Instead of applying manifold learning to policy parameters, we can also reduce the dimensionality of the states or of the observations. Ha and Schmidhuber [HS18] use a VAE [KW14] to compress images from video games. They train a recurrent neural network to predict future states and employ a linear controller that uses the compressed observation and hidden state of the recurrent network. The controller is optimized with CMA-ES. An autoencoder can be used for dimensionality reduction (encoder) but it can also be used to generate data from the latent space (decoder). A VAE specifically can be used to learn such a generative model.

The approach that we discuss in this section is similar to the work of Matsubara et al. [MHM10] and Rueckert et al. [Rue+15] in the sense that we learn a low-dimensional representation of trajectories from expert demonstrations, but we do not need an indirect trajectory representation. We will furthermore focus more on action generation than perception in contrast to Ha and Schmidhuber [HS18], that is, we will use a generative model. We will develop a nonlinear manifold learning approach for trajectories and we will perform policy search in the low-dimensional, latent parameter space of trajectories.

4.5.2. Proposed Manifold Learning Approach

Our proposed network architecture is displayed in Figure 4.18. We use a VAE to learn a low-dimensional representation of trajectories, however, we are only interested in the generative model $p(\boldsymbol{X}|\boldsymbol{z})$ after training, where $\boldsymbol{X}$ represents a trajectory and $\boldsymbol{z}$ is a latent vector. We will directly generate trajectories and not have an indirect trajectory representation via movement primitives. For this purpose we develop a special kind of layer to ensure that generated trajectories are smooth. This layer does not have any learnable parameters and can be added at the end of any neural network that should output trajectories. It is a way of integrating prior knowledge in the structure of the neural network without restricting the capacity of a neural network since the new layer implements a linear transformation.

Sequence prediction problems have been solved mostly by recurrent neural networks (for example, in machine translation [SVL14]), however, recurrent modules are nowadays replaced by feedforward architectures (such as transformers [Vas+17]) that can consider all information without relying on faulty memory [Car+20] and are easier to train.

4.5.2.1. Trajectory Layer

Trajectories executed by humans are usually smooth. Trajectories executed by robots should be smooth to avoid high accelerations and velocities. We can encode this knowledge in a layer of a neural network. We will use a trick presented by Kalakrishnan et al. [Kal+11a] to generate smooth and dense trajectories of end-effector poses.

With $\boldsymbol{\epsilon} \in \mathbb{R}^{T \times D}$ and $\boldsymbol{x}_0 \in \mathbb{R}^D$ from previous layers the trajectory layer implements

$$\boldsymbol{X} = g(\boldsymbol{\epsilon}, \boldsymbol{x}_0).$$

$\boldsymbol{x}_0$ represents the initial pose or position in the trajectory $\boldsymbol{X}$ that will be generated by the layer. $\boldsymbol{X} \in \mathbb{R}^{T \times D}$ defines T steps in D dimensions.

The layer performs a matrix multiplication $\boldsymbol{L}\boldsymbol{\epsilon}_{\bullet d}$ for each column $\boldsymbol{\epsilon}_{\bullet d}$ of $\boldsymbol{\epsilon}$ to compute trajectory offsets in each dimension $d \in \{1, \ldots, D\}$ that will be added to the initial position $\boldsymbol{x}_{0d}$ of that dimension. Hence, the layer performs only linear operations.

We will now explain how the constant matrix $\boldsymbol{L}$ is computed. It is based on a covariance matrix of a multivariate Gaussian that generates low acceleration with respect to the quadratic cost function $c(\boldsymbol{x}) = \boldsymbol{x}^T \boldsymbol{A}^T \boldsymbol{A}\boldsymbol{x}$. Note that $\boldsymbol{x}$ here is a sequence of one-dimensional points and not a state vector. $\boldsymbol{A}$ is a second order finite difference matrix. Second order backward differences for a sequence can be used as an approximation of the second derivative, that is,

$$f''(t) \approx \frac{f(t) - 2f(t - \Delta t) + f(t - 2\Delta t)}{\Delta t^2}.$$

With a sequence $\boldsymbol{x} = (x_1, x_2, x_3, \ldots, x_T)^T$ and a temporal difference of Δt, this results in

$$\ddot{x}_t \approx \frac{x_t - 2x_{t-1} + x_{t-2}}{\Delta t^2},$$

which can be written as matrix multiplication $\ddot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x}$, with

$$\boldsymbol{A} = \frac{1}{\Delta t^2} \begin{pmatrix} 1 & 0 & 0 & & 0 & 0 & 0 \\ -2 & 1 & 0 & \ldots & 0 & 0 & 0 \\ 1 & -2 & 1 & & 0 & 0 & 0 \\ & \vdots & & \ddots & & \vdots & \\ 0 & 0 & 0 & & 1 & -2 & 1 \\ 0 & 0 & 0 & \ldots & 0 & 1 & -2 \\ 0 & 0 & 0 & & 0 & 0 & 1 \end{pmatrix} \in \mathbb{R}^{n+2 \times n}$$

to obtain the sequence of acceleration $\ddot{\boldsymbol{x}}$ for a given sequence of positions $\boldsymbol{x}$, so that our cost function reduces to $c(\boldsymbol{x}) = \ddot{\boldsymbol{x}}^T \ddot{\boldsymbol{x}}$.

Kalakrishnan et al. [Kal+11a] propose $(\boldsymbol{A}^T \boldsymbol{A})^{-1}$ as a covariance matrix of a multivariate normal distribution over trajectories $\mathcal{N}\left(\boldsymbol{x} | \boldsymbol{0}, (\boldsymbol{A}^T \boldsymbol{A})^{-1}\right)$ in the context of motion planning for manipulation because samples from this distribution have a low control cost. According to Toussaint [Tou11, Equation 33] this results in an expected cost of $\mathbb{E}_{\boldsymbol{x}}[\boldsymbol{x}^T \boldsymbol{A}^T \boldsymbol{A} \boldsymbol{x}] = \mathrm{Tr}[\boldsymbol{A}^T \boldsymbol{A} (\boldsymbol{A}^T \boldsymbol{A})^{-1}] = \mathrm{Tr}[\boldsymbol{I}_{n+2}] = n+2$, which only depends on the number of steps n in the trajectories. Sampled trajectories start at zero and end at zero. To generate a distribution that diverges from zero in the end we can use the upper left quarter of this covariance matrix.

We can reparameterize a standard normal distribution $\boldsymbol{y} \sim \mathcal{N}(0, \boldsymbol{I})$ to output these trajectories with $\boldsymbol{x} = \boldsymbol{L}\boldsymbol{y}$, where $\boldsymbol{L}$ is obtained by Cholesky decomposition $\boldsymbol{L}\boldsymbol{L}^T = (\boldsymbol{A}^T \boldsymbol{A})^{-1}$ and is a lower triangular matrix. While the Cholesky decomposition has cubic complexity with respect to length of the trajectory and number of task space dimensions, it only has to be computed once during the training phase.

We can use the same trick for other distributions. In our case we apply it to the output of a neural network that transforms a Gaussian distribution. So far we described everything for one-dimensional trajectories, but we can use the same approach for each dimension of a multi-dimensional space individually. Since we produce dense trajectories, we can even approximate trajectories of unit quaternions for orientation representation well enough by handling each dimension of the quaternion individually and normalizing the generated quaternions.

4.5.2.2. Configuration of the Variational Autoencoder

We use a VAE [KW14] as a basis to train our trajectory generator. The complete architecture is displayed in Figure 4.18. We use dense layers followed by nonlinear activation functions. We found that using a leaky ReLU activation function gives better results than tanh or ReLU (refer to Maas et al. [MHN13] for definitions of the activation functions). Furthermore a scaling layer that multiplies a learnable scalar with the output of each layer makes learning more stable. This might be regarded as a simple version of batch normalization [IS15]. Similar simplifications have been used by Zhang et al. [ZDM19] and Kingma and Dhariwal [KD18]. Zhang et al. [ZDM19] use learnable scalar multipliers for residual blocks. Kingma and Dhariwal [KD18] use an actnorm layer, a simplified version of batch normalization for small batch sizes.

Our decoder has a special structure that employs the trajectory layer. Therefore, we need to generate two inputs $\boldsymbol{\epsilon}$ and $\boldsymbol{x}_0$ to the trajectory layer from a latent vector $\boldsymbol{z}$. These will be generated by two separate paths in the decoder.

In addition, we use a modified loss similar to the β-VAE [Hig+17], however, Higgins et al. [Hig+17] suggest to use $\beta > 1$ to learn a better disentanglement of the data while we choose $0 < \beta < 1$ to generate trajectories that are closer to the demonstrated trajectories. Similar configurations have been investigated previously by Hoffman et al. [HRJ17] and Alemi et al. [Ale+18].

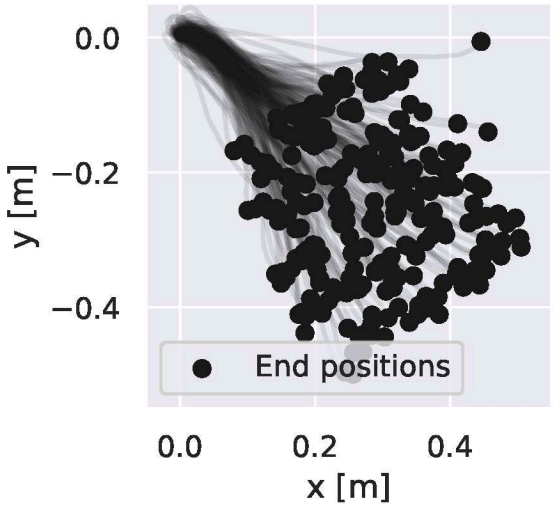


Figure 4.19.: Demonstrated grasping movements projected to x-y plane. Lines indicate trajectories and circles mark end positions of these trajectories.

4.5.3. Experiments

We investigate the following questions: (1) Is the VTAE better for trajectories than other manifold learning algorithms? (2) Can the VTAE learn a manifold in which we can smoothly interpolate between trajectories that are similar to demonstrated trajectories? (3) Can we use the contextual policy search algorithm BO-CPS to learn a mapping from contexts to corresponding parameters in the latent space? (4) Can we do this on a real robot?

4.5.3.1. Dataset

We recorded 249 pick and place movements from one person with XSens Awinda [Xse20], which is a motion capture system based on inertial measurement units. The person had to pick a small cylindrical object from various positions on a table. We were only interested in the grasping movement and manually extracted those movements. We scaled each trajectory temporally to the same length, which is the mode of the gamma distribution fitted on the lengths of the trajectories. Each trajectory consists of 76 steps at a frequency of 60 Hz. We also centered the data so that all trajectories start at the position $(0, 0, 0)$. Orientations were not changed. We only use the end-effector pose since the positions of the fingers are completely different from the positions of the fingers on the robot. Figure 4.19 displays all demonstrated trajectories.

4.5.3.2. Training and Hyperparameters of Variational Trajectory Autoencoder

We reduce the number of dimensions to two since we want to restrict the search space for policy search to improve sample efficiency. We explored different architectures for

the decoder. In the initial architecture we had an additional layer with 266 nodes as the third layer of the decoder, but we found this made the mapping from latent space to trajectory space too complex such that interpolation between trajectories in latent space was not smooth enough to apply contextual policy search. The decoder has to be nonlinear to represent the demonstrated trajectories adequately, but we have to balance the capacity to reconstruct the original dataset and the smoothness of interpolation between trajectories. One way to do this is to set β appropriately, another way is to adapt the architecture of the decoder.

We set $\beta = 0.1$, as this gives us the best compromise between fitting the dataset accurately and approximating a Gaussian distribution in the latent space. We tried $\beta \in [0, 0.01, 0.1, 1, 5]$.

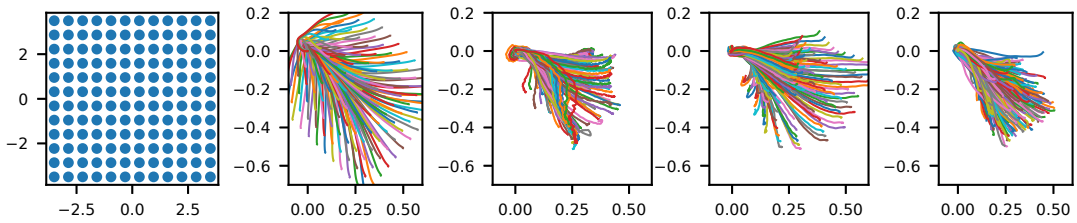
With a batch size of 16, we train the VAE for 500 epochs with Adam [KB15], that is, all samples from the training set are used 500 times. We also trained for 50,000 epochs but did not see relevant progress in the training loss after 500 epochs. We also explored smaller and larger batch sizes and found that a batch size of 16 achieves the lowest loss after 500 epochs. Training was done either on an Intel i7-5960X CPU or on an NVidia Titan X, which was twice as fast.

4.5.3.3. Comparison of Manifold Learning Approaches

Figure 4.20 compares three manifold learning approaches to encode the grasp dataset. We generate a grid with 144 points in latent space and project it with PCA [Pea01], a standard VAE, and a VTAE. PCA is a linear model and apparently does not capture the distribution of the training set accurately, as there are many trajectories that are not in the training set. This extrapolation might be desirable, for instance, with only a few demonstrations but it is not desirable in our case, where we want to guide policy search by demonstrations. The standard VAE uses the same architecture as the VTAE without a trajectory layer for decoding $\boldsymbol{\epsilon}$, however, we can see that the trajectories are more shaky than with the other models. This property is not desirable for trajectories that should be executed on a robot. The VTAE is a good compromise between capturing the distribution and smoothness.

4.5.3.4. Interpolation

We want to explore the learned manifold and the projection from the latent space to trajectory space. Figure 4.21a and 4.21b show interpolations along two feature axes. Those have been identified by learning linear mappings of the form $y = \boldsymbol{w}^T \boldsymbol{z}$ from the latent space to a characteristic feature y of the corresponding trajectories that are generated by the decoder. This approach has been proposed by Guan [Gua18] in the context of generative adversarial networks. Our assumption is that $\boldsymbol{w}$ corresponds to the axis of maximum variation of the trajectory feature y. In Figure 4.21a the feature axis varies the difference between the position on the y-axis between start and goal in trajectory space and in Figure 4.21b the feature axis varies the difference between the position on the z-axis between start and goal in trajectory space.



(a) Latent grid. (b) PCA. (c) VAE. (d) VTAE. (e) Training set.

Figure 4.20.: Projection of grid in latent space to trajectory space with three manifold learning approaches: linear PCA, VAE, and VTAE. Trajectories are projected on the x-y plane. Each colored line corresponds to one trajectory that is projected from one point of the grid in latent space. For comparison we also show all trajectories from the training set.

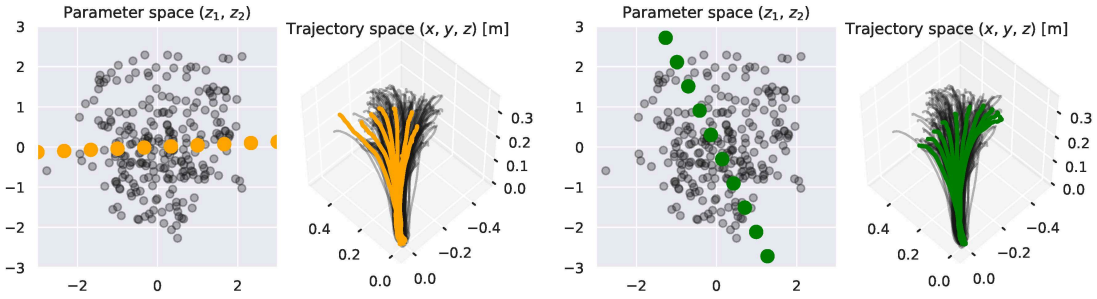
In Figure 4.21c we select two samples from the training set that are clearly different in trajectory space, project them to latent space, interpolate in latent space and project the interpolated latent variables back to trajectory space. Hence, we can identify axes in latent space that have different effect on the shape of trajectories and we can also smoothly interpolate between trajectories.

4.5.3.5. Simulated Contextual Policy Search

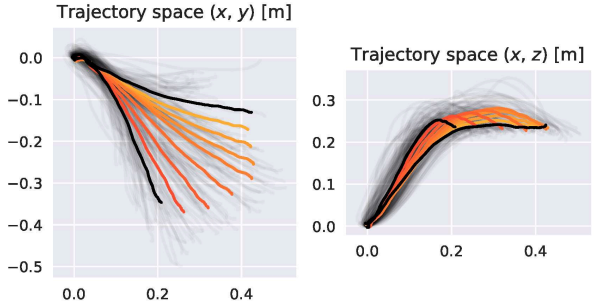
In Section 4.4 we already demonstrated that BO-CPS can be more sample-efficient than C-REPS if we reduce the number of policy parameters drastically. We will first check whether contextual policy search algorithms are generally able to exploit the latent space to do sample-efficient policy search and which algorithm works best. For this purpose we will use a simple reaching task: goal positions for the end effector are located on a line in 3D space. Goals $\boldsymbol{s} = (g_x, g_y)^T$ are varied along one axis, that is, $g_x = 0.35$ and $g_y \in [-0.4, -0.15]$. Goals are used as context for contextual policy search and they define the return, which is the negative distance of the end effector's center after executing the trajectory to the goal. In this and in the following experiment, contextual policy search algorithms learn an upper-level policy $\pi_\omega(\boldsymbol{\theta}|\boldsymbol{s})$, where for our purpose $\boldsymbol{\theta} = \boldsymbol{z}$ are the latent vectors that will be projected to trajectories by the decoder of the VTAE.

Figure 4.22a shows a solution to the problem that has been obtained by BO-CPS after 250 episodes. Note that we only optimize the position distance to the goal. We do not have any penalty on accelerations or velocities. Nevertheless, all trajectories are smooth, even during the learning process.

Hyperparameters: We compare BO-CPS, C-REPS, C-CMA-ES, and aC-REPS. The search space for all algorithms is restricted to $[-3, 3] \times [-3, 3]$ in the latent space. The initial variance of C-CMA-ES and C-REPS is set to 5, we use a quadratic upper-level policy that maps from context $\boldsymbol{s}$ to trajectory parameters $\boldsymbol{z}$. Although BO-CPS is non-

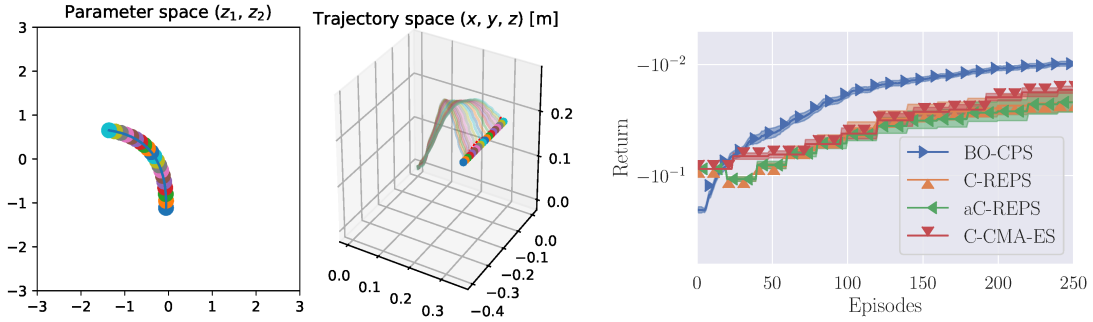


(a) Latent space interpolation along feature axis 1. (b) Latent space interpolation along feature axis 2.



(c) Latent space interpolation between training samples. Projection on x-y plane (left) and x-z plane (right).

Figure 4.21.: Interpolations in latent space. Black circles and lines in the background show the training set.



(a) Learned policy (quadratic mapping from goal positions to the latent space) for the simulated reaching problem. Goal positions are represented by circles in the right plot, corresponding trajectories are indicated by lines in the same color, and their parameters in latent space are shown by circles in the left plot.

(b) Comparison of contextual policy search algorithms in latent space. Mean and standard error over 20 experiments are displayed. The y-axis is logarithmic.

Figure 4.22.: Reaching problem.

parametric, we also learn a quadratic policy from all samples at test time because this speeds up querying parameters for given contexts drastically. The quadratic policy is obtained in the same way as in C-REPS, although we use all samples. After manual

hyperparameter tuning we decide to update the upper-level policy in C-REPS and aC-REPS after 20 episodes with a history of 25 samples. C-CMA-ES determines these parameters automatically. We tried updates after $[5, 10, 20, 25, 30, 50]$ samples and a training set size of $[5, 10, 20, 25, 30, 50, 100, 200, 250]$ for all three algorithms but did not find any better configuration. For BO-CPS we use Gaussian process regression as a surrogate model. We manually designed the kernel $k(\boldsymbol{x}_1, \boldsymbol{x}_2) = c_1 M_\nu(\boldsymbol{x}_1, \boldsymbol{x}_2) + W(\boldsymbol{x}_1, \boldsymbol{x}_2) + c_2$, where M_ν is a Matérn kernel with an anisotropic length scale initialized at 1 and limited to $[0.01, 100]$ and the parameter ν is set to 1.5. c_1, c_2 are constant kernels that are initialized to 100 and are limited to $\left[10^{-3}, 10^5\right]$ and $\left[10^{-2}, 10^5\right]$ respectively. W is a white noise kernel initialized at a noise level of 1 and limited to $\left[10^{-5}, 10^5\right]$. We did not invest much time in tuning the kernel hyperparameters and they could be optimized. We use GP-UCB as acquisition function with the parameter κ to balance exploration and exploitation. $\kappa = 2$ was just enough to avoid too much exploitation. We tried the values $1, 1.5, 2$. For every query of the acquisition function we set the budget of the global optimizer DIRECT [JPS93] to 500 iterations and allow L-BFGS-B [Byr+95b] to run until convergence for fine tuning.

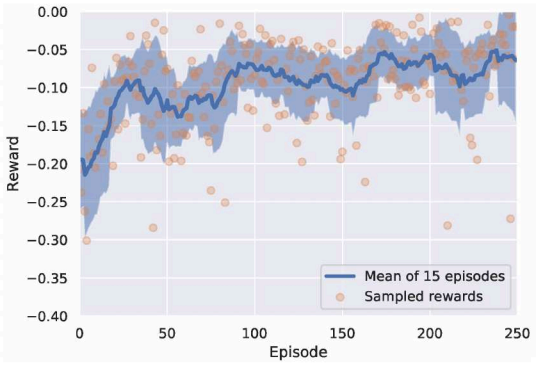
Figure 4.22b shows the learning curves of the four algorithms. BO-CPS clearly outperforms the other algorithms. A detailed analysis can be found in Appendix G. BO-CPS's computational complexity, however, depends on the number of samples that were previously explored. This makes it slow after much more than the 250 episodes that were needed in this task. Hence, it is not an optimal algorithm for learning in simulation. Nevertheless, for learning in reality this is acceptable because learning in the real world with a robot is a tedious task if it cannot be fully automated. Hence, we prefer sample efficiency over low computational complexity.

4.5.3.6. Contextual Policy Search on Real Robot

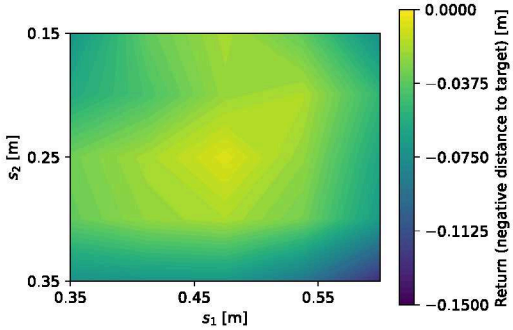
We use a UR5 robot arm (see Appendix F.3 for details) and a Robotiq 2F-140 gripper for this experiment to grasp a can. BO-CPS generalizes grasping behaviors to a predefined area (see Figure 4.23c).

We perform 250 episodes, as it is a good compromise between minimizing episodes on the real robot on the one hand and the amount of samples necessary to learn a useful skill with BO-CPS on the other hand. As the task of grasping is closely related to reaching, we use the same hyperparameters as in the previous experiment.

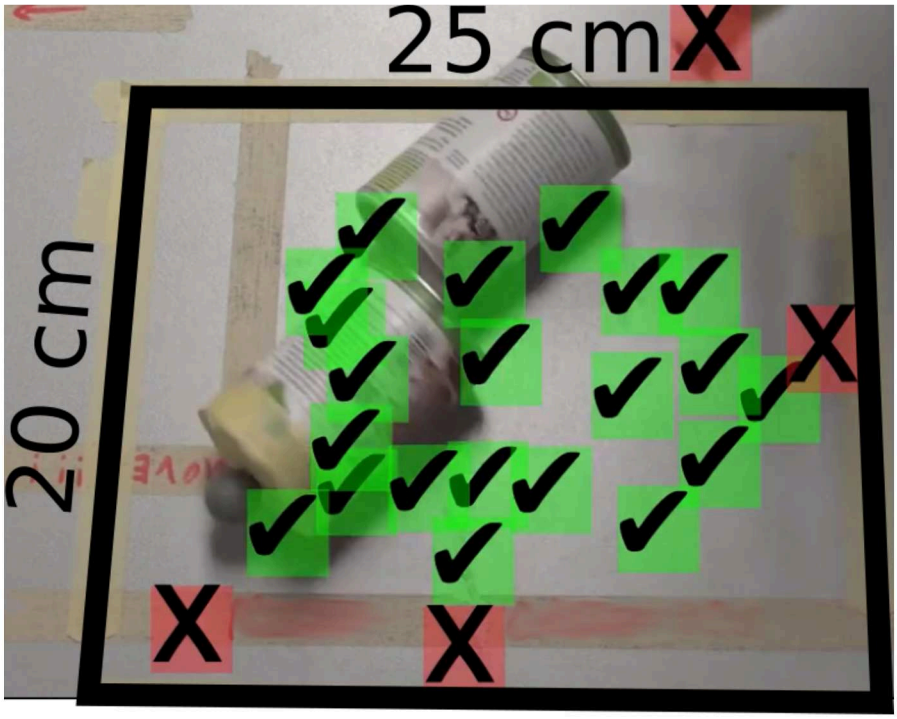
With a marker-based motion capture system from Qualisys AB [Qua20a] we measure the pose of the gripper and learn to reach given points with it. We assume that we do not know the transformation between the coordinate system of the motion capture system and the robot. The robot is blind, that is, it does not receive any information about the problem other than the target location (context) and the return, which is the negative distance between the gripper's center and the target location. The context is sampled uniformly from an area of the size $20\,\text{cm} \times 25\,\text{cm}$ during training. During this phase we do not actually grasp the cans that we use during the testing phase since we only optimize the distance to the goal.



(a) Learning curve on UR5.



(b) Return for various contexts



(c) Successful and failed grasp attempts.

Figure 4.23.: Results of contextual policy search on UR5.

The learning curve is displayed in Figure 4.23a. Note that each return measurement corresponds to a different context. Therefore, we plot the mean and standard deviation of 15 consecutive episodes to get a smoother estimate of the actual learning progress.

The final quality of the obtained policy is displayed in Figure 4.23b. The return (negative distance to the target location) is indicated for various target locations by the color. We sampled a grid of 5×5 target locations to generate this plot. In the middle of the target area the distances to the target are 1 cm or less. We also did an evaluation

were we actually tried to grasp cans (see Figure 4.23c). The gripper is triggered after the execution of the trajectory. Can locations close to the border or outside of the context area do not result in successful grasps while it is possible to successfully grasp objects in the middle of the target area. Although the gripper is forgiving for inaccuracies because of the size of the can compared with the span of the gripper, it is often failing to grasp the can just by a centimeter at the border of the target area.

The main focus of this experiment is not to show that BO-CPS can solve this particular problem better than other approaches, but to demonstrate that the whole approach from manifold learning to contextual policy search works on a real robot and is agnostic to the kind of context parameters. We also see that the generated trajectories are easily executable by a standard robot arm, as they are smooth. Grasping seems to be much simpler than ball throwing at first glance. Part of the difficulty in the throwing problem comes from flexibility of the projectile and of the robot, which we cannot model accurately, however, this just means that the projectile motions are a bit noisy. A similarly complicated part is identifying the transformation between the robot's coordinate system and the measurement coordinate system. We learn a policy that implicitly replaces an explicit chain of mappings from an end position on the ground over the corresponding projectile motion, over the throwing motion, to corresponding policy parameters. In the grasping problem we have a similar problem, as the transformation between the robot's coordinate system and the motion capture system's coordinate system is unknown. It is inferred indirectly by contextual policy search and encoded in the mapping from target object position (context) over latent space parameters to the end pose of the trajectory, which might include several nonlinearities. We think that while the concrete challenges for other skill learning problems might take on a different form, the degree of difficulty is often similar.

4.5.4. Discussion

Complex movement generation without tight coupling to sensors is an underexplored field in reinforcement and imitation learning, particularly in combination with deep learning which mostly focuses on complex, redundant sensory data. With the trajectory layer we provide a building block that generates smooth trajectories, which are particularly suited for robots. It is a module for neural networks that can be used to enforce a specific property without too strong restrictions. Much success in deep learning is based on similarly domain-specific modules, for example, convolutional layers [LeC+89] or transformers [Vas+17].

In this section, we use the trajectory layer with manifold learning to train a low-dimensional representation of grasping movements to demonstrate that this can be combined with contextual policy search algorithms to efficiently learn grasping on a real robotic system. Nonetheless, the approach is not limited to this learning paradigm. We would also like to build a bridge to deep RL, which learns nonlinear, parametric models but currently often needs a tight sensor-actuator coupling, that is, a current sensor measurement is fed into a neural network and the network generates an actuator command (such as joint angles or torques), which is then executed. The decoder of our variational trajectory autoencoder could also be used in combination with deep RL algorithms. We

can couple it with a rich sensor processing network such as an object detection network and train a mapping from camera images to grasping movements.

An interesting application is transfer learning. It is common practice in deep learning to use pretraining and transfer learning to solve harder problems or problems with fewer training data, for instance, the backbone of RetinaNet for object detection is pretrained on ImageNet [Lin+17]. Devlin et al. [Dev+19] establish similar practice in language understanding. We argue that robots have to build on pretrained action primitives that can be extended and refined to reach a high competence level in terms of motor skills. If we want to succeed with this approach we have to collect more datasets of movements, and train more complex models that also combine multiple types of movements, not just grasping motions. Our trajectory layer could be a building block for skill networks that can be trained on larger datasets and form the basis for transfer learning.

The limitation of this approach is that it is not able to represent any possible trajectory. We limit the agent by the mapping that has been learned previously. In future work, the weights of the network could also be adjusted during RL.

4.6. Summary

We investigate several ways to improve contextual policy search algorithms with the goal to increase sample efficiency. These are

- active context selection,
- return normalization and selection of the best training samples (aC-REPS),
- surrogate models (C-ACM-ES, BO-CPS),
- and manifold learning (VTAE).

Table 4.3 summarizes the performance improvements that we measure in experiments. The two most important numerical indicators for improvement are the number of episodes to reach a certain return and the return reached after a certain number of episodes. Note that the numbers are not exact, as they sometimes have to be approximated from learning curves because there is no measurement at the exact point when the threshold is reached. Furthermore, the results between different experiments are not directly comparable because of the different robotic arms, policies, and policy parameterizations that we use in different works. A PA-10 arm and a sequence of two DMPs, of which the metaparameters are learned, are used to compare context selection approaches. In this case ball throwing is learned for the target area $[-3\,m, -5\,m] \times [-3\,m, -5\,m]$, that is, $4\,m^2$. AC-REPS is evaluated with COMPI, one DMP, of which the 60 weights are learned, and the target area $[-3\,m, 3\,m] \times [3\,m, 6\,m]$ $(18\,m^2)$. The reason for these different evaluation processes is that the experiments are performed over a long period of time with an evolving simulation and evaluation procedure. Furthermore, the analysis of the results varies a lot: the learning curves of aC-REPS show quartiles of 30 independent runs while BO-CPS in simulation is only evaluated in one run of which the mean performance and the standard deviation over 16 test contexts are plotted. Note that the result of the latter

Table 4.3.: Improvements of sample efficiency in contextual policy search.

Approach	Comparison (ours vs. baseline)	Criterion	Improvement (over baseline)	(percent)
Active context selection	Ball throwing with monotonic progress heuristic vs. round-robin selection	Episodes to return −0.2	3,200 vs. 4,800	33.3 %
		Return after 10,000 episodes	-0.109 vs. -0.164	33.5 %
AC-REPS	Ball throwing with aC-REPS vs. C-REPS	Episodes to return −0.1	4,000 vs. 8,000	50 %
		Return after 3,500 episodes	-0.991 vs. -2.123	53.3 %
BO-CPS	Ball throwing with BO-CPS vs. C-REPS	Return after 250 episodes	-2,400 vs. -8,000	70 %
VTAE	Grasping with BO-CPS vs. C-REPS	Episodes to return −0.03	63 vs. 180	65 %
		Return after 250 episodes	-0.009 vs. -0.024	59.8 %

experiment can only be understood as preliminary and it has to be backed up by further experiments, which we do with experiments on the real COMPI and in combination with the VTAE.

Active context selection and return normalization are applicable to both C-REPS and C-CMA-ES. The ranking SVM can only be applied to C-CMA-ES as a surrogate model. Manifold learning can be applied to any policy search problem, but it is most effective in combination with a sample-efficient contextual policy search algorithm such as BO-CPS, which uses Gaussian process regression as a surrogate model and is computationally expensive in comparison to C-CMA-ES or C-REPS.

BO-CPS for a low number of policy parameters is sample-efficient enough to generalize behaviors over a certain context space and manifold learning from demonstrations can be used to extract relevant policy parameters. Further improvement of sample efficiency is possible with state transition models, and integration of rich sensor data through neural network policies should be investigated in the future.

Related Publications

[Fab+13] Alexander Fabisch, Yohannes Kassahun, Hendrik Wöhrle, and Frank Kirchner. “Learning in compressed space”. In: *Neural Networks* 42 (2013), pp. 83–93. ISSN: 0893-6080. DOI: 10.1016/j.neunet.2013.01.020.

[Fab+15] Alexander Fabisch, Jan Hendrik Metzen, Mario Michael Krell, and Frank Kirchner. "Accounting for Task-Difficulty in Active Multi-Task Robot Control Learning". In: *KI – Künstliche Intelligenz* 29.4 (2015), pp. 369–377. ISSN: 1610-1987. DOI: 10.1007/s13218-015-0363-2.

[Fab19] Alexander Fabisch. "Empirical Evaluation of Contextual Policy Search with a Comparison-based Surrogate Model and Active Covariance Matrix Adaptation". In: *Genetic and Evolutionary Computation Conference Companion*. Ed. by Manuel López-Ibáñez. GECCO '19. ACM, 2019, pp. 251–252. ISBN: 978-1-4503-6748-6. DOI: 10.1145/3319619.3321935.

[FK20] Alexander Fabisch and Frank Kirchner. "Variational Trajectory Autoencoder for Sample-Efficient Policy Search". In: *Conference on Robot Learning (CoRL)*. Submitted. 2020.

[FM14] Alexander Fabisch and Jan Hendrik Metzen. "Active Contextual Policy Search". In: *Journal of Machine Learning Research* 15 (2014), pp. 3371–3399. URL: http://jmlr.org/papers/v15/fabisch14a.html.

[Gut+18] Lisa Gutzeit, Alexander Fabisch, Marc Otto, Jan Hendrik Metzen, Jonas Hansen, Frank Kirchner, and Elsa Andrea Kirchner. "The BesMan Learning Platform for Automated Robot Skill Learning". In: *Frontiers in Robotics and AI* 5 (2018), p. 43. ISSN: 2296-9144. DOI: 10.3389/frobt.2018.00043.

[MFH15] Jan Hendrik Metzen, Alexander Fabisch, and Jonas Hansen. "Bayesian Optimization for Contextual Policy Search". In: *Machine Learning in Planning and Control of Robot Motion (MLPC) Workshop, IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Ed. by Aleksandra Faust. 2015. URL: https://www.cs.unm.edu/~afaust/MLPC15_proceedings/MLPC15_paper_Metzen.pdf.

Active contextual policy search has been published in Fabisch and Metzen [FM14]. The complete paper is a joint work with Jan Hendrik Metzen and my contributions are included in this chapter: the development of the context selection method and its evaluation on ball throwing. Active training set selection has been presented in Fabisch et al. [Fab+15] and this chapter summarizes my contributions. As mentioned previously, the PUBSVE is a joint development with Mario Michael Krell. The extensions to C-CMA-ES that were presented in this chapter have been published in Fabisch [Fab19].

As I emphasized before, BO-CPS has been developed by Metzen et al. [MFH15]. This thesis merely contributes the evaluation of the algorithm in simulated robotic ball throwing and a discussion of the algorithm's computational complexity. The corresponding experiments on the real robot COMPI were presented in Gutzeit et al. [Gut+18] but were conducted by Jonas Hansen during his Master's thesis, which I supervised.

The idea of reducing the dimensionality of a policy originated from earlier work on neural networks with fixed dimensionality reduction [Fab+13]. This resulted in application of manifold learning to contextual policy search in Fabisch and Kirchner [FK20].

Part III.

A Framework for Robot Behavior Learning

Chapter 5.

A Conceptual Framework for Automatic Robot Behavior Learning

Parts of this chapter were published originally as [Gut+18] and have been revised.

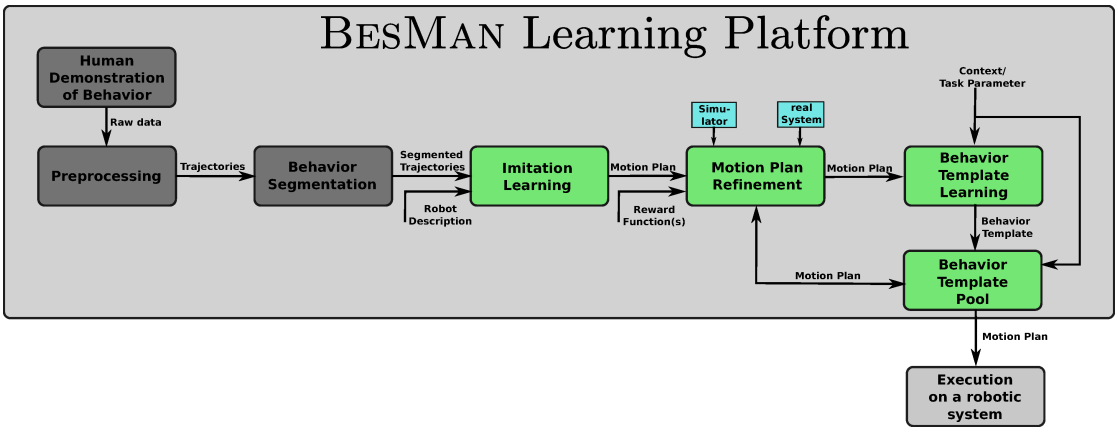


Figure 5.1.: Data flow of the BesMan Learning Platform: a demonstrated behavior is segmented into behavioral building blocks that will be imitated and refined using reinforcement learning and can be generalized to more generic behavior templates. The modules that have been developed in this thesis are marked with a green background.

The BesMan learning platform is a stand-alone solution to automatically learn robotic manipulation behavior for different robotic systems and applications. It is designed for situations, in which the operator has no direct physical contact to a robot. Behavior that is adaptive to task changes and different target platforms can be learned to solve unforeseen challenges and tasks that can occur during deployment of a robot. The learning platform is composed of components that deal with preprocessing of human demonstrations, segmenting the demonstrated behavior into basic building blocks, imitation, refinement by means of reinforcement learning, and generalization to related tasks.

This learning platform is a framework in which the previous chapters are embedded. Note that only parts of the whole learning platform were developed in this thesis. We will describe the whole framework, point out which parts this thesis contributes, and summarize the evaluation of the whole learning platform in this chapter.

5.1. Overview

Learning complex behaviors at once is often difficult or even impossible. Thus, it is often easier to learn small behavioral building blocks separately and combine them to complex behaviors. Studies with rodents [Gra98] and children [Adi+08] indicate that this strategy is efficient. Therefore, we will follow this approach.

Behavior learning often combines various approaches to leverage intuitive knowledge from humans as we have seen in Chapter 2. In particular, imitation learning (IL) and RL are often combined: a standard approach is to learn behaviors by initialization with a demonstrated movement and refinement through RL. Complete descriptions of robot skill learning frameworks are hardly present in the literature. To our best knowledge the only work that gives a complete overview of a learning architecture and is comparable to the work that we present here has been published by Peters et al. [Pet+12]. They do not provide a thorough evaluation of the automation level and time consumption to learn new skills. Their work includes movement primitives, policy search, contextual policy search, and methods to learn low-level control. In this work as well as in the majority of similar works the relevant behaviors are directly presented by kinesthetic teaching so that the correspondence problem [ND02] is neglected. In addition, only the relevant behavior is presented or it is not discussed how the relevant part that should be transferred is extracted. In contrast to that, we would like to let a human demonstrate the behavior as naturally as possible. With this approach, the system can be situated in a far away place or environment hostile to man and could still learn from a human demonstration although direct kinesthetic teaching is not possible or would only be possible indirectly in case that a second identical system would be available. To allow the demonstrator to act naturally, we use behavior segmentation methods and solve the correspondence problem as automatically as possible.

The learning platform (see Figure 5.1) transfers movements from a human teacher to a robotic system. During this process so-called motion plans and behavior templates are generated. Motion plans represent solutions to generate specific behaviors as motor plans that have been introduced in Chapter 1. Behavior templates represent generic movements to generate a flexible behavior that is able to, for example, reach different points in space. They are a form of skill according to the definition given in Chapter 1, as they generalize over a limited set of context variables or task parameters.

In a first step, the learning platform records demonstrations and preprocesses these (see Section 5.2) to generate labeled behavioral building blocks that are independent of the target system. For each relevant building block, we use imitation learning to represent the recorded trajectory segments as motion plans (see Section 5.3). Motion plans describe trajectories that could be executed by the robot and mimic the trajectories presented by the human demonstrator during a single behavioral building block. To account for the correspondence problem (see Section 2.2.1.2), the *Motion Plan Refinement* module can use RL (see Section 5.4) to adapt the motion plan. This requires interaction with the real or simulated target system and the specification of a reward function which tells the learning algorithm how well a motion plan solves the task. Alternatively or in conjunction

with RL, transfer learning as described in Section 5.4.2 can be used to adapt motion plans. Using this method, differences between learned behaviors in simulation and on the real robot are also considered during the motion plan refinement. Motion plans are solutions for specific settings. It is often necessary to learn more generic behavior templates that can be applied to similar settings. This is achieved by the *Behavior Template Learning* module (see Section 5.4). Behavior templates are capable of generating motion plans for new but similar settings. Once a behavior template has been learned, it is added to the *Behavior Template Pool*, which is accessible from the robotic system. The behavior templates in this pool can be used directly during operation.

In the following sections, we give detailed descriptions of the modules. Note that the learning platform provides a framework with various methods, of which not all have to be used for all applications. We will present three examples with different instantiations of the learning platform.

5.2. Motion Capture and Preprocessing

This section describes the motion capture process and the necessary preprocessing steps for the motion capture data. The preprocessing includes marker labeling and behavior segmentation. For this thesis, we will assume that these parts are already available and we just describe them.

We record movements of a person with a motion capture system. For this work we use a marker-based system from Qualisys AB [Qua20a][1] and attach visual markers to the human demonstrator and to objects that are involved in a particular task to also record important changes in the environment.

We place markers at shoulder, elbow, and hand of the human demonstrator. Three markers at one position can be used to infer orientations. This is important in manipulation tasks that require the robot to imitate the orientation of the hand. By placing three markers at the back, all marker positions can be transformed into a coordinate system relative to the demonstrator to make the recordings independent from the global coordinate system of the camera setup. Additional markers can be placed on manipulated objects. An example of a recording setup for ball-throwing behaviors is shown in Figure 5.2.

Because a passive marker-based motion-capture system is used, an automatic marker identification based on the relative positions of the markers to each other is required. Methods are provided by manufacturers of motion capture systems. The Qualisys track manager [Qua20b] offers automatic identification of markers, which is based on previously labeled motion capture data. Other works rely on manually defined or otherwise inferred skeletons. Refer to Meyer et al. [Mey+14] and Schubert et al. [Sch+15b; Sch+16b] for details about some of these approaches.

We have to identify the main movement segments of recorded demonstrations to transfer the movements to a robotic system. Therefore, we segment the demonstration into

[1] Instead of a marker-based motion capture system we could use also other motion capture system that can extract poses of human body parts such as the hand.

its building blocks and classify each building block into a known movement class. Then we can automatically select movement sequences that are required to solve a certain task with a robotic system without user input.

To decompose demonstrated behaviors into simple building blocks, we have to identify the characteristics of the movements. Senger et al. [Sen+14] developed the velocity-based Multiple Change-point Inference (vMCI) algorithm to segment manipulation behavior. It is a fully unsupervised algorithm that requires no parameter tuning as shown in several experiments [Sen+14; GK16].

Segmented motions have to be annotated to select trajectories that should be learned and transferred to the robot. To minimize manual effort, this annotation should work with small training sets. As proposed by Gutzeit and Kirchner [GK16], we use the nearest neighbor based on Euclidean distance on the normalized trajectories transformed to a coordinate system relative to the human demonstrator to assign movement classes to the acquired segments.

5.3. Imitation Learning

We use IL to obtain motion plans from the recorded trajectories. A workflow for learning from demonstrations must address the correspondence problem as well as the representation of the motion plan. In this section we introduce motion plan representations used within the BesMan learning platform and discuss the correspondence problem.

5.3.1. Correspondence Problem

The following two mappings have to be defined: the record mapping, which maps marker trajectories to a sequence of actions or system states, and the embodiment mapping, which maps the recorded sequence to a trajectory that is executable on the target system.

We cannot directly observe the actions of the agent, but we can observe the marker positions, which means that a part of the record mapping is already given. Instead of using the observed marker positions directly, we reduce the marker positions to a representation that is more meaningful to describe manipulation behavior and is independent of the platform, that is, we extract end-effector poses in a reference frame. For this purpose we first have to define which are the markers on the end effector, which is specific for a marker setup. The reference frame depends on the application. For goal-directed manipulation behavior Wirkus [Wir14] proposes to use the target as a reference frame, for example, a box that we want to grasp. For behaviors such as ball-throwing it is better to use a reference frame on the teacher, for example, on the back, because the target object (the ball) will be moved with the end effector. Works by Manschitz et al. [Man+16], Calinon [Cal16], and Niekum et al. [Nie+15] select the reference frame automatically but we did not consider this here.

Although it seems like transferring end-effector poses to the target system is simply an inverse kinematics problem, it can actually be much more complicated, as the target system might not have the same workspace, kinematic structure, and dynamic capabilities

as the teacher. The approach to define the embodiment mapping automatically has been presented in Chapter 3. With the task-agnostic embodiment mapping we only integrated knowledge about our target system. To ensure that the imitated skill has the same effects as the demonstration, we must integrate knowledge about the task. This will be done in the policy refinement step. We have to define a reward function that can be used by reinforcement learning methods to complete the embodiment mapping. Here we account for kinematic and dynamic differences that cannot be resolved easily, for example, a human teacher might have a hand structure that is different from the target system. An example is displayed in Figure 3.1: the target system does not even have an active hand. It has a scoop mounted on the tip of the arm. In other cases, the robot might have a gripper that does not have all of the capabilities of a human hand. Another problem in the ball-throwing domain are the dynamic and kinematic differences between the human demonstrator and the target system. It might be possible for the robot to execute the throwing movement after temporal scaling, but this step can reduce the velocity in Cartesian space so drastically that the ball does not even leave the scoop any more. Details on the methods for this step are given in Section 5.4.

5.3.2. Motion Plan Representation

Dynamical movement primitives [Ijs+13] have a unique closed-form solution for imitation learning, which makes it appealing for our purpose. After imitation, the parameters of the DMP can be adjusted easily which makes it suitable for policy search.

The standard DMP formulation allows to set the initial state, goal state, and execution time as metaparameters. There are several variants of DMPs. See Section 2.2.1.4 for more details. An extension that allows to set a goal velocity has been developed by Mülling et al. [MKP11; Mül+13]. We use this DMP formulation for trajectories in joint space. A solution for orientations has been proposed by Ude et al. [Ude+14] with unit quaternions. To represent trajectories in Cartesian space, we use a combination of a position DMP as formulated by Mülling et al. [MKP11; Mül+13] and an orientation DMP by Ude et al. [Ude+14].

5.4. Refinement and Generalization

The learning platform provides tools to adjust motion plans to specific target systems and to generalize motion plans over specified task parameters based on reinforcement learning (RL).

Depending on the application, we have to decide whether learning will take place in simulation or in reality. Learning in reality would give the best results, however, this might not always be feasible because some problems might require a lot of samples. This depends on the target system and the application. Standard policy search (see Section 5.4.1) is always included in our learning platform to ensure that the embodiment mapping is completed. Furthermore, it is sometimes a good idea to model the relevant aspects of the task in simulation and start with the refinement in simulation. When a good motion

plan is obtained, this can be used to start learning in reality directly or approaches can be applied to handle the simulation-reality gap (see Section 5.4.2).

We can then generalize the obtained motion plan to a behavior template that takes the current context as a parameter to modify its motion plan using contextual policy search (see Section 5.4.3).

5.4.1. Refinement with Policy Search

Policy search is sample-efficient in domains where a good initial policy can be provided, the state space is high-dimensional and continuous, and the optimal policy can be represented easily with a prestructured policy such as a DMP (see Section 2.2.1.4).

Depending on whether we want to optimize only a few metaparameters or the whole set of parameters, we can select the best policy search method. We use several algorithms in the learning platform. Among them are CMA-ES [HO01] and REPS [PMA10]) as local search approaches, which means they need a good initialization provided by imitation learning. In addition, we use Bayesian optimization [BCd10] as a global optimizer. Bayesian optimization is limited to a few parameters because of the computational complexity, however, it is sample-efficient.

We can learn DMPs in joint space or in Cartesian space depending on the task. See Chapter 3 for a discussion.

5.4.2. Simulation-Reality Transfer

Motion plans learned only in simulation often perform worse when they are executed in reality. In certain situations, this performance drop is small, for example, in open-loop control with a robot having accurate and precise actuators. In such a case, it can be sufficient to apply policy search methods in simulation and transfer the result directly to the real system. Often, however, the *simulation-reality gap* is a problem. Hence, the work of Otto [Ott15] is integrated in the learning platform to address this problem.

The ball-throwing task that we investigate in this chapter requires to release the ball at a certain position and velocity. The time of release, speed and direction are not easy to simulate accurately [Ott15] because we would need a detailed model of the ball and the ball mount. We avoid this with the Transferability Approach [KMD13] that refines motion plans with few episodes on the real system and is assisted by a simulation that does not have to be perfectly accurate. It tests only a few motion plans in transfer experiments on the real system and compares them with the simulation to learn a surrogate model that estimates the transferability of motion plans. Thus, testing motion plans with low transferability is not necessary.

5.4.3. Contextual Policy Search

A disadvantage of using prestructured motion plans such as DMPs is that they are designed for a specific situation and generalize only over predefined metaparameters. We can use contextual policy search to generalize over arbitrary task parameters that often

have non-trivial relations to the optimal motion plan. The algorithms for contextual policy search that are developed for the learning platform are presented in detail in Chapter 4.

5.5. Evaluation of the Learning Platform

In this section, we evaluate the learning platform as a whole in a ball-throwing scenario. We transfer human movements to a robotic arm. We are particularly interested in time requirements and level of automation. We show that the learning platform can be run with minimal user interference to learn from different non-expert subjects, who demonstrate motions. The complete experiment builds on multiple components of the learning platform. Segmentation, annotation, and simulation-reality transfer are not a contribution of this thesis. We will focus only on those aspects that have been developed in this thesis or are essential for the evaluation.

5.5.1. Methods

5.5.1.1. Robotic System

We transfer the movements to the robot arm COMPI (see Appendix F.1) that is displayed in Figure 3.1. A scoop is attached to COMPI to hold a ball. The position where the ball hits the ground can not always be identical because of varying positions of the ball in the scoop, varying shape of the deformable and not perfectly round ball, inaccuracies in the execution of the desired trajectory, and measurement errors. How reproducible this position is depends on the throwing movement. For some throwing movements the standard deviation of the position can be more than a meter because the ball sometimes falls down before the throwing movement is finished and sometimes not.

5.5.1.2. Data Acquisition

The recording setup for a demonstration of a throw can be seen in Figure 5.2. Seven cameras tracked eight visual markers attached to the human and the target area. Only five cameras were focused directly on the subject. The recorded marker positions were labeled according to their position on the human body (for example, 'shoulder'). The subjects had to throw a ball to a goal position on the ground, approximately 2 m away. To limit the range of possible throws, they were instructed to throw the ball from above with the hand above the shoulder while throwing (see Figure 5.2). The subjects had to move their arm to a resting position in which it loosely hangs down between the throws. The movement was demonstrated by 10 subjects. All subjects were right-handed and had different throwing skills ranging from non-experts to subjects performing ball sports as a hobby (basketball, volleyball, or handball). Each subject demonstrated 8 throws in 3 experiments which results in total numbers of 24 throws per subject, 30 experiments, and 240 throws for all subjects.

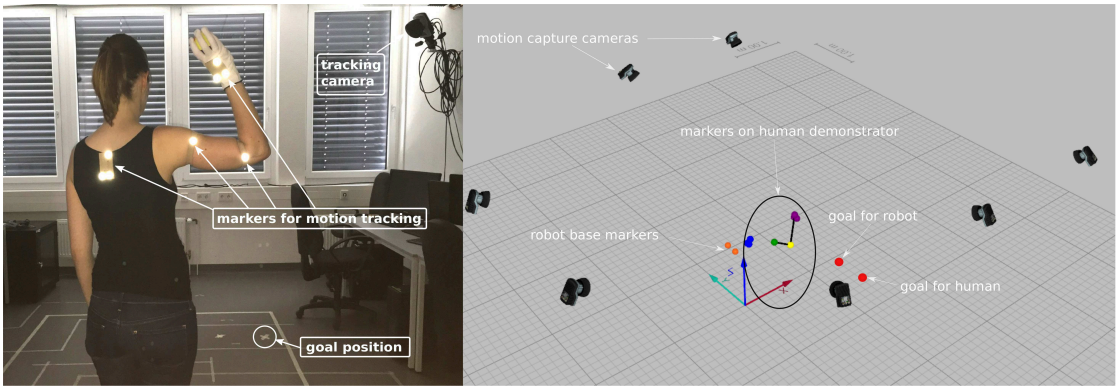


Figure 5.2.: Data acquisition setup. Motion capture cameras, markers, and goal positions for the throws in the setting are displayed. (Illustrated by Lisa Gutzeit in [Gut+18].)

5.5.1.3. Imitation Learning

IL is based on end-effector poses. End-effector trajectories cannot be transferred directly to the robot's coordinate system so that we have to do a synchronization frame optimization to translate and rotate the original trajectory so that it fits into the workspace of COMPI. The end-effector trajectories are transformed into joint trajectories via inverse kinematics. In addition, we scale the joint trajectories so that the joint velocity limits of the target system are respected. In a last step, the throwing movement is represented as a joint space DMP via IL. Moreover, a minimum execution time of 0.95 s is set to reduce the velocity and accelerations, which are penalized during the following optimization.

5.5.1.4. Motion Plan Refinement

While the DMPs resulting from IL can be executed on the robot, they do not necessarily have the same effect on the ball as the movements of the human. The lack of actuated fingers as well as kinematic and dynamic differences to the human lead to the need for adaptation, which we do via policy search. The adapted policy parameters include: initial position, goal, weights and execution time, consisting of 6, 6, 36, and 1 value(s). Following the concept of the Transferability Approach, we aim to minimize two objectives: (a) the target distance of the touchdown position in simulation and (b) the distance between the touchdown position in simulation and in reality. The target distance in reality is not directly optimized, but evaluated during and at the end of the experiment.

The optimization consists of several steps. (1) refinement in simulation, (2) refinement in simulation and reality, (3) refinement of simulation, refinement in simulation and reality, and (5) evaluation of motion plans.

5.5.1.5. Required Time and Level of Automation

For each main module of the learning platform that is needed to learn a new behavior, the required time was measured. The time required to learn a new behavior is strongly influenced by the degree of automation, which is furthermore important to simplify application. Therefore, we evaluated the degree of automation for each module of the learning platform.

5.5.2. Results and Discussion

5.5.2.1. Imitation Learning and Motion Plan Refinement

In this section, we evaluate the target distance measured at several steps of the learning platform. Note that Marc Otto did this evaluation and it is not a contribution of this thesis. It has been published with Gutzeit et al. [Gut+18]. Nevertheless, as it is an interesting part of the complete evaluation, we will present it here.

(0) **Initial performance:** At first, we evaluate the results of the IL in simulation. The IL does not consider suitability for holding and throwing the ball. Hence, several motion plans result in a simple ball drop near the initial position. This is reflected by target distances around 2.15 m in Figure 5.3 a). None of the simulated ball throws is closer than 1 m to the target. Note that there is also a lot of variation in the number of throw segments that are detected in the data containing 24 actual demonstrations per subject. (1) **Refinement in Simulation**: For one subject, the goal (distance is below the 0.1 m threshold indicated by the gray line in Figure 5.3 a)) was not reached. For all remaining subjects, the goal was reached in fewer than 2200 episodes in at least one of the 6 runs. (2) **Refinement in Simulation and Reality**: For 2 out of 9 subjects, the transfer experiments reached target distances below 0.1 m in reality. The deviations of the touchdown positions in reality and in simulation seem to be systematic, that is, for throwing behaviors resulting from the same subject, similar deviations are found. Having a constant offset between simulated and real results contradicts the premise of the Transferability Approach, aiming to find a region in the parameter space that features transferable motion plans. Hence, we decide to adapt the simulation specifically to predict the touchdown positions for some of the movements more accurately. (3) **Simulation refinement**: To reduce the offset of real and simulated results, we minimize the median of the touchdown disparities obtained for the 25 transfer experiments so far. This is done via a simple grid search on the value for the robot height and the ball-release angle. The medians could be reduced to a range of 0.11 m to 0.23 m (depending on subject; compared with 0.23 m to 0.73 m before). (4) **Refinement in Simulation and Reality**: For one subject, this optimization step was aborted after 10 critical transfer experiments, during which joint limits were exceeded and the robot was deactivated consequently. For all of the remaining subjects, target distances below 0.1 m as well as touchdown disparities below 0.1 m occurred in the 50 transfer experiments. The best target distances so far are

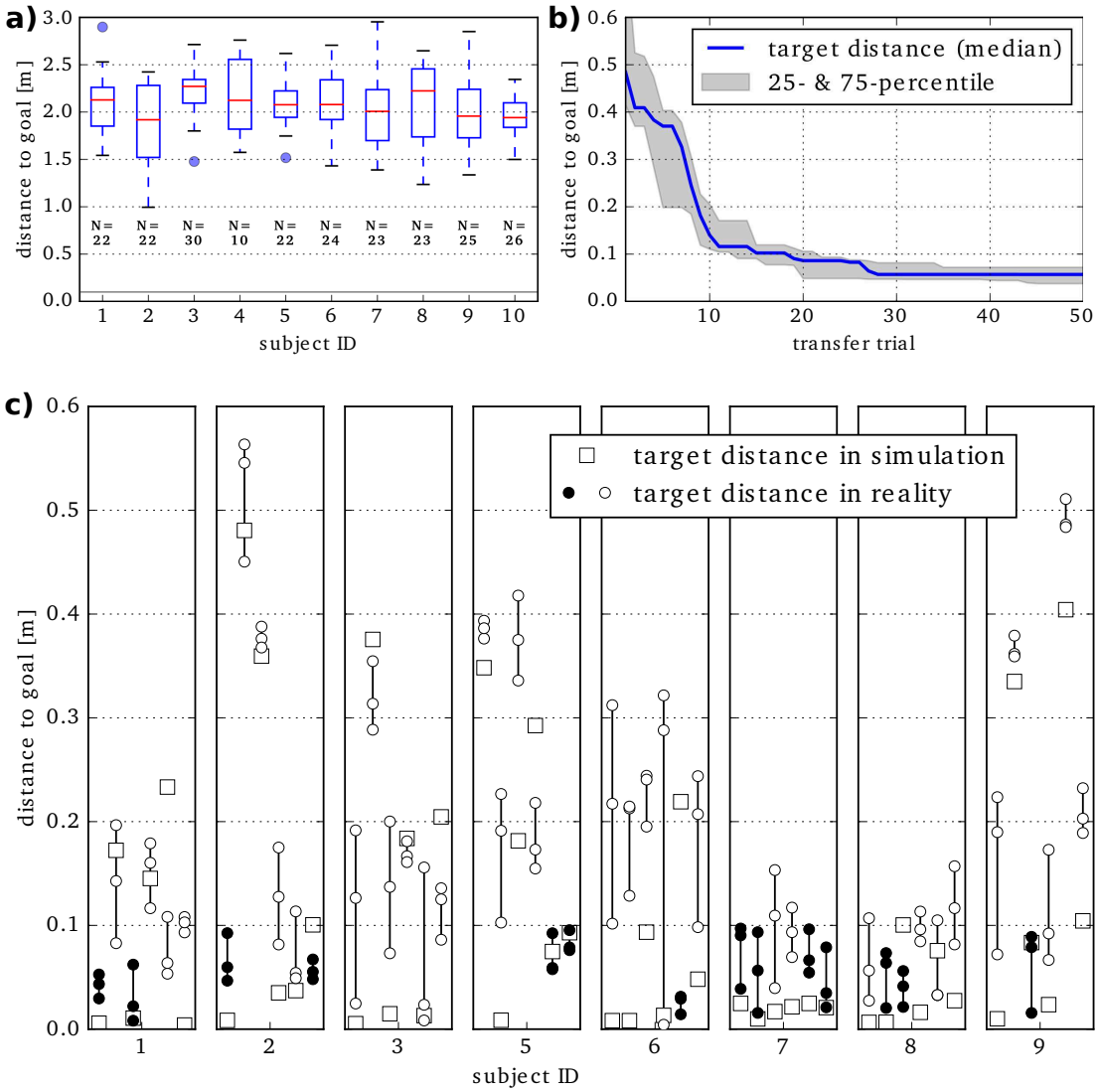


Figure 5.3.: a) Results of IL evaluated in simulation are shown by Tukey box plots. N is the number of throw segments detected in the data of a specific subject. The solid gray line indicates the 0.1 m threshold. Please note the different scaling of the ordinate, compared to b) and c). b) Closest distance to target achieved in transfer episodes during robot-in-the-loop optimization. The median over 8 subjects of the best results so far is shown. For each subject, 50 transfer episodes are executed on the robot during the optimization. c) At the end of the optimization process, for each subject, six motion plans are automatically selected for evaluation. The result from the deterministic simulation is shown by squares. Each motion plan is evaluated three times in reality (circles). If the ball landed closer than 0.1 m to the target in all three repetitions, the circles are filled, otherwise not. (Illustrated by Marc Otto in Gutzeit et al. [Gut+18].)

Table 5.1.: Required time (per experiment, 8 throws) per module.

Step	Time	Automated	Required knowledge
Attaching markers for motion capture	0:55 min	✗	
Motion capture	1:08 min	✗	
Automatic marker labeling	4:58 min	✓	Neighboring markers, initial pose
Manual marker labeling	9:19 min	✗	
Behavior segmentation	0:44 min	✓	
Labeling for movement classification[2]	50 min	✗	
Movement classification	2 sec	✓	
Imitation learning	4:20 min	✓	Robot description
Policy search	10 min	✓	Reward function, simulation
Transferability approach	75 min	(✓)	Reward function, simulation

shown in Figure 5.3 b). The curve indicates that 25 transfers are sufficient to minimize the target distance. (5) **Evaluation of Candidates**: Figure 5.3 c) shows the performance evaluation of 6 automatically selected final candidate solutions (2 with the lowest target distance in reality and 4 from the Pareto front. Up to 4 of these hit the ground reliably, that is, the target distance is below 0.1 m in all three repetitions (marked by filled dots). For seven out of eight (remaining) subjects, at least one selected candidate solution hits the target reliably.

5.5.2.2. Required Time

An overview of the required time for each step can be found in Table 5.1. Note that labeling the dataset for movement classification has to be done only once. The required time for successful automatic marker labeling is much faster than manual labeling even though the automatic labeling is slow for these specific data because of many gaps in marker trajectories. If the markers were always visible, the automatic labeling would have taken only about some seconds. The longest part in the whole process is the refinement for the target platform (imitation, policy search, transfer), which is a difficult problem that involves interaction with the real world.

[2]Dataset from 5 experiments including 40 throws.

5.5.2.3. Level of Automation

Although we have automated the process of acquiring new behaviors, still some human intervention is required in the form of knowledge that has to be given to the system and physical interaction with the system. An overview can be found in Table 5.1.

Apparently, it is necessary that a human demonstrates the movement. Movement classification requires a dataset that is labeled manually, but we minimized the effort by using a classifier which classifies at a high accuracy with small training sets.

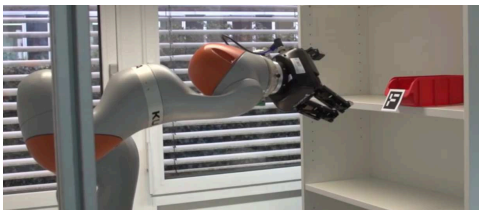
When we set up the learning platform for a new target system and type of manipulation behavior we have to decide which components we combine for embodiment mapping, for example, in the ball-throwing scenario the synchronization frame optimization was useful, which is not always the case, however, IL is completely automated.

The motion plan refinement with the Transferability Approach requires human assistance because the robot has to try throwing movements in the real world and is not able to get the ball back on its own. The process itself is automated so that no knowledge about the system or the task is required from the human at this step. In addition, we have to define a reward function that describes how a solution of a task should look like and because we want to minimize the interaction with the real world we use a simulation which has to be designed. This is a manual process at the moment.

5.5.3. Application of the Learning Platform in Different Scenarios

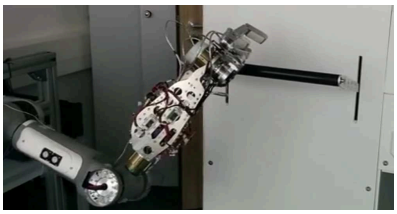
In addition to throwing a ball, the learning platform has been used in other scenarios to transfer movements to other robotic systems. In a pick-and-place scenario a grasping movement was extracted from demonstrations of picking a box from a shelf, placing it on a table and putting it back afterwards. The movements were recorded with a marker based tracking system. After successful segmentation and classification of the grasping movement [GK16], it was imitated using a Cartesian space DMP and adapted to be executed on a Kuka iiwa lightweight robot equipped with a 3-finger gripper from Robotiq (for details see Appendix F.2). Cartesian DMPs were used for easy integration with whole-body control and perception. In this scenario the refinement was done using the CMA-ES algorithm in simulation. After 50–100 iterations, the movement could be successfully transferred to the robotic system. One demonstration of the initial trajectory is sufficient.

In another scenario, the learning platform was used to teach the arm of the robotic system Mantis (see Appendix F.5 for details) to pull a lever. Like in the pick-and-place scenario, policy search was used to adapt the demonstrated movement to the robotic system. REPS and CMA-ES gave good results. After several hundred episodes in simulation, a successful movement could be generated. Learning could be done in parallel from multiple demonstrations with each RL process being initialized with a single demonstration. The position of the lever during training in simulation was varied slightly so that the behavior has to be robust enough, which makes it possible to compensate for position uncertainty in the real world.



(a) Grasping a box, video: [DFK16b]

(b) Throwing



(c) Pulling a lever, video: [DFK16a]

Figure 5.4.: Robotic applications.

As in the ball-throwing scenario, the transfer of the demonstrated movements was partially automated. To imitate and adapt the demonstrations to the system, the embodiment mapping and a reward function have to be selected with regard to the robotic system and the task goal. The robotic systems that we used are shown in Figure 5.4.

5.6. Summary

Our results show that it is possible to learn new skills for robots without specifying the solution directly. The automated learning platform, that is the first complete system of its kind, leverages intuitive knowledge from humans that do not know anything about the target system to automatically transfer skills to robots. The main impediments that can be overcome by the learning platform in this setting are the kinematic and dynamic differences of the demonstrator and the target system.

On the basis of the learning platform, we can build a library of movements that are represented independently of the target system. We could then use methods for embodiment mapping to transfer those skills to several target systems.

Related Publications

[Gut+18] Lisa Gutzeit, Alexander Fabisch, Marc Otto, Jan Hendrik Metzen, Jonas Hansen, Frank Kirchner, and Elsa Andrea Kirchner. "The BesMan Learning Platform for Automated Robot Skill Learning". In: *Frontiers in Robotics and AI* 5 (2018), p. 43. ISSN: 2296-9144. DOI: 10.3389/frobt.2018.00043.

[Gut+19] Lisa Gutzeit, Alexander Fabisch, Christoph Petzoldt, Hendrik Wiese, and Frank Kirchner. "Automated Robot Skill Learning from Demonstration for Various Robot Systems". In: *KI: Advances in Artificial Intelligence*. Ed. by Christoph Benzmüller and Heiner Stuckenschmidt. Springer International Publishing, 2019, pp. 168–181. ISBN: 978-3-030-30179-8. DOI: 10.1007/978-3-030-30179-8_14.

[Met+14] Jan Hendrik Metzen, Alexander Fabisch, Lisa Senger, José de Gea Fernández, and Elsa Andrea Kirchner. "Towards Learning of Generic Skills for Robotic Manipulation". In: *KI – Künstliche Intelligenz* 28.1 (2014), pp. 15–20. ISSN: 1610-1987. DOI: 10.1007/s13218-013-0280-1.

The idea of an automated learning platform has been introduced by Metzen et al. [Met+14]. It has subsequently been implemented and evaluated by Gutzeit et al. [Gut+18] (first two authors contributed equally) and evaluated with a focus on the embodiment mapping by Gutzeit et al. [Gut+19] (first three authors contributed equally). The contributions of this thesis are the embodiment mapping, imitation learning, policy refinement through reinforcement learning, behavior template learning, and integration of the learning platform for specific applications (throwing, grasping, pulling a lever). Lisa Gutzeit contributed to data acquisition and the behavior segmentation and classification modules. Marc Otto contributed the Transferability Approach to alleviate the simulation-reality gap and performed the experiments with COMPI. Jan Hendrik Metzen contributed to imitation learning, policy refinement through reinforcement learning, and behavior template learning, in particular to the implementation of basic algorithms (DMPs, REPS, C-REPS, BO) and the development of BO-CPS and extensions of it.

```python
import numpy as np
import matplotlib.pyplot as plt
from bolero.environment import OpenAiGym
from bolero.behavior_search import BlackBoxSearch
from bolero.optimizer import CMAESOptimizer
from bolero.representation import LinearBehavior
from bolero.controller import Controller


beh = LinearBehavior()
env = OpenAiGym("CartPole-v0", render=False, seed=0)
opt = CMAESOptimizer(variance=10.0 ** 2, random_state=0)
bs = BlackBoxSearch(beh, opt)
controller = Controller(environment=env, behavior_search=bs, n_episodes=300)

rewards = controller.learn()
controller.episode_with(bs.get_best_behavior())

plt.figure()
```

Chapter 6.

BOLeRo: Behavior Optimization and Learning for Robots

This chapter was published originally as [FLK20] and has been revised.

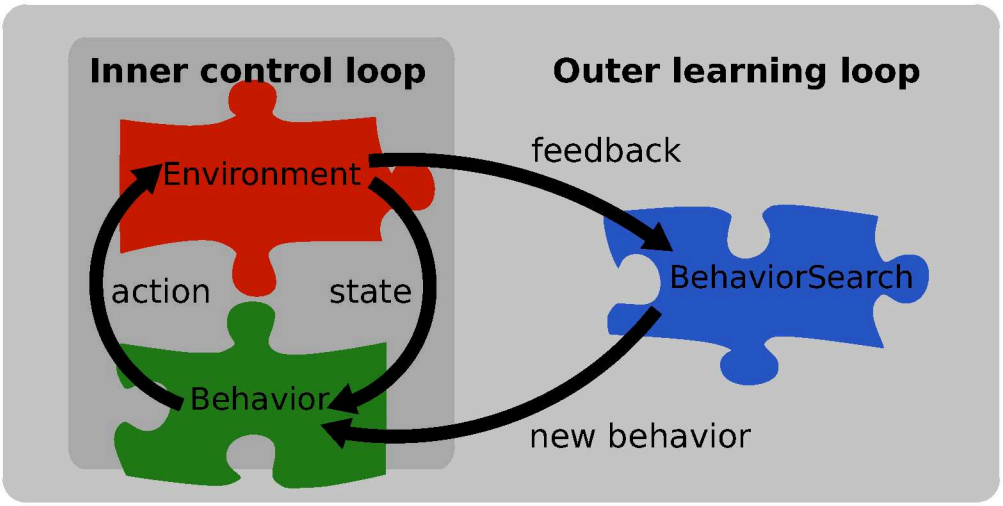


Figure 6.1.: Main cycles during episodic learning process.

The software BOLeRo[1] is designed to support the process of research and development of behavior learning and optimization for robots. Its main goals are (1) to be a benchmark framework for behavior search algorithms (learning and optimization), (2) to provide reference implementations for benchmarks, behavior representations, and well-established and reliable behavior search algorithms, and (3) it should be easy to integrate behaviors or behavior search algorithms in real robotic systems.

6.1. Related Work

Some of BOLeRo's goals overlap with the goals of other open source software packages. Some of these focus on behavior learning algorithms and some on benchmarks. A major problem is that many published experiments are hardly reproducible, particularly if

[1] Available at `https://github.com/rock-learning/bolero`.

robotic simulations are involved. Small changes in the simulation setup can considerably influence the problem complexity and often not all simulation details are published. Contact parameters for a walking system, for instance, might influence the preferred walking pattern. In BOLeRo it is easy to create environments with the MARS simulation software [MAR20]. Hence, it simplifies the publication of reference implementations of robotic learning environments. A recent development that covers the same aspects as BOLeRo is OpenAI Gym [Bro+16]. It provides a range of environments from simple test problems to complex robotic problems to improve reproducibility of experiments and enable rigorous comparison. There are other works that build upon OpenAI Gym, for example, Lopez et al. [Lop+19] extend it with various simulated robotic scenarios.

There are several behavior learning libraries that focus on classical RL [FG13; Ger+15], which often does not work out of the box for robotic problems. Some open source libraries provide implementations of deep RL [Aru+17]: OpenAI Baselines [Dha+17], Dopamine [Cas+18], TF-Agents [Gua+18], and Garage [Dua+16]. Deep RL, however, as we discussed in Section 2 is not easily applicable to real robotic systems yet and its main problem is sample efficiency. Available implementations are often either in the state of research code or are designed to learn behaviors of agents in virtual worlds. An alternative is the recently published library PyRoboLearn [Del20] that provides several algorithms that overlap with BOLeRo's algorithms.

Nevertheless, we see a niche here for a library that provides implementations of established behavior search algorithms that work well for robots and that is designed to support the behavior development workflow for a robot and we propose BOLeRo as a solution. BOLeRo mainly includes policy search and movement primitives. Implementations of these types of algorithms have previously been provided as Matlab scripts without common interfaces or as individual libraries for one algorithm.

6.2. Design and Features

One of the main design decisions of BOLeRo is to keep interfaces simple, which allows to quickly integrate external methods. Decoupling by interfaces also provides flexibility in combining methods and applications. It also enables us to easily combine Python and C++. There are Python bindings for C++ components and, vice versa, it is possible to run Python components from C++ via the BOLeRo interfaces.

BOLeRo can be used as a library that provides behaviors, behavior search algorithms (learning and optimization), or simulation environments that represent behavior learning problems. It is also a framework for comparison of behavior learning algorithms.

Figure 6.1 illustrates how BOLeRo can be used as a benchmark framework. An environment defines the learning problem. A behavior is evaluated in the environment in a control loop: in each step the behavior observes the current state of the environment, computes an action, and the action is executed in the environment. After the control loop finished, which is indicated by the environment, feedback (reward or fitness) is given from the environment to the behavior search algorithm. The behavior search uses feedback to generate new behaviors in a learning loop. This episodic learning process is

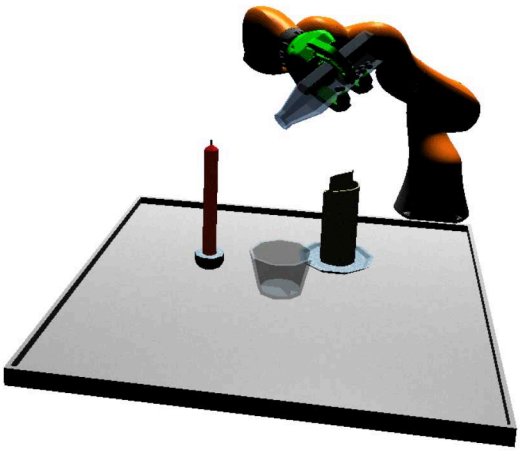


Figure 6.2.: Example of a MARS environment in BOLeRo. The Kuka arm has to pour a couple of marbles.

implemented by a *controller* in BOLeRo. This can be done with a C++ controller that loads benchmark configuration files or we can use Python scripts to define the benchmark (see Algorithm 6 for an example). In this case the control flow is defined by BOLeRo and it is used as a framework, but it is not the only way to use the individual components.

It is not required to change the design of existing software to use BOLeRo. For integration in robotic middlewares we suggest using BOLeRo as a library and let the developer define the control flow based on the robotic framework. The transition from learning in simulation to a robotic system is a major challenge. The choice of languages, methods, dependencies, and interfaces often complicates this process. While most robotic frameworks use C++ as their core language (for example, ROS [Qui+09] or RoCK [Sch+14]), Python is the most important language for machine learning with a large ecosystem for scientific programming. We want to make BOLeRo compatible with both worlds. Therefore, it is possible to use it in both C++ and Python. We describe the BOLeRo interfaces and available implementations in the next paragraphs.

Environments. BOLeRo environments define behavior learning problems. An environment communicates with a behavior via states that it reports and actions that it receives and executes. Furthermore, it reports evaluation feedback to the behavior search.

Having a robotic simulation to test the feasibility of approaches, new methods, or to be able to train complete behaviors that can be used on real robots is necessary in a behavior learning library for robots. There are various simulations that can be used to define robotic environments: MARS [MAR20], Bullet [Cou15], Gazebo [KH04], CoppeliaSim / V-REP [RSF13], or MuJoCo [TET12b]. Communication with simulations is often not easy and not all simulations are suitable for behavior learning. We prepared a base class for environments that use the MARS simulation software in BOLeRo. With graphical tools such as Phobos [SR20], new robotic environments can be defined. An

example is shown in Figure 6.2. Implementing new environments is also possible with the physics engine Bullet via the convenient pybullet API. BOLeRo also a wraps OpenAI Gym [Bro+16] environments since both have the goal to provide reproducible learning environments.

Behaviors. Behavior search algorithms can generate new behaviors. Sometimes behaviors are an integral part of them. Behaviors map observed states to actions.

BOLeRo provides baseline behaviors such as a random and a linear behavior as well as implementations of recent movement primitives such as Cartesian space DMPs [Ude+14] and ProMPs [Par+18].

Behavior Search Algorithms. Behavior search algorithms combine behavior representations (for example, neural networks in deep reinforcement learning and neuroevolution or movement primitives in policy search) with behavior optimization (for example, policy gradient algorithms in deep reinforcement learning or black-box optimization in episodic policy search). They need feedback from the environment to update behaviors.

BOLeRo provides implementations of the following policy search algorithms: episodic REPS [PMA10], C-REPS [Kup+13], CMA-ES [HO01], ACM-ES [LSS10], C-CMA-ES [Abd+17a], and NES [Wie+14]. A wrapper around scikit-optimize [Tim18], a library for model-based optimization, is integrated. An additional package [MF20] for BO [BCd10] and BO-CPS [MFH15] is available. Step-based reinforcement learning and deep reinforcement learning algorithms are planned as next features.

6.3. Examples and Applications

6.3.1. Simple Example

An episodic learning process can be organized by a controller in a simulation (see Figure 6.1). A simple source code example is shown in Algorithm 6. The behavior is a DMP in this case (Python class: `DMPBehavior`), the behavior search is a black-box search (`BlackBoxSearch`), which uses the black-box optimization algorithm CMA-ES (`CMAESOptimizer`) to modify weights of the DMP. The environment `OptimumTrajectory` is a simple 2D trajectory planning problem: three circular obstacles have to be avoided on the path from start to goal while minimizing acceleration. The corresponding learning curve is displayed in Figure 6.3 together with the environment, the final trajectory, and several intermediate solutions.

6.3.2. Other Applications

BOLeRo has been used in various research projects[2] to learn skills for different robots and in various publications [Met+14; FM14; Fab+15; MFH15; Met16; Gut+18; LSK14; Det+14; Fab20; Fab19a]. In addition to the skills displayed in Figure 5.4, BOLeRo has also been used to learn walking behaviors for a mantis-like robotic system.

[2]For example, BesMan [DFK16c] and LIMES [DFK16d].

Algorithm 6 Short version of an example from BOLeRo's documentation: https://rock-learning.github.io/bolero/auto_examples/behavior_search/plot_obstacle_avoidance.html

```
import numpy as np
from bolero.environment import OptimumTrajectory
from bolero.behavior_search import BlackBoxSearch
from bolero.optimizer import CMAESOptimizer
from bolero.representation import DMPBehavior
from bolero.controller import Controller
x0, g = np.zeros(2), np.ones(2)  # initial and goal state
execution_time = 1.0
dt = 0.01
env = OptimumTrajectory(
    x0, g, execution_time, dt, obstacles=[[.5, .5], [.6, .8], [.8, .6]],
    penalty_goal_dist=1.0, penalty_obstacle=1000.0, penalty_acc=1.0)
bs = BlackBoxSearch(DMPBehavior(execution_time, dt, n_features=10),
                    CMAESOptimizer(variance=100.0 ** 2, random_state=0))
controller = Controller(environment=env, behavior_search=bs,
                        n_episodes=500, record_inputs=True)

rewards = controller.learn(["x0", "g"], [x0, g])
controller.episode_with(bs.get_best_behavior(), ["x0", "g"], [x0, g])
X = np.asarray(controller.inputs_[-1])  # get last trajectory
```

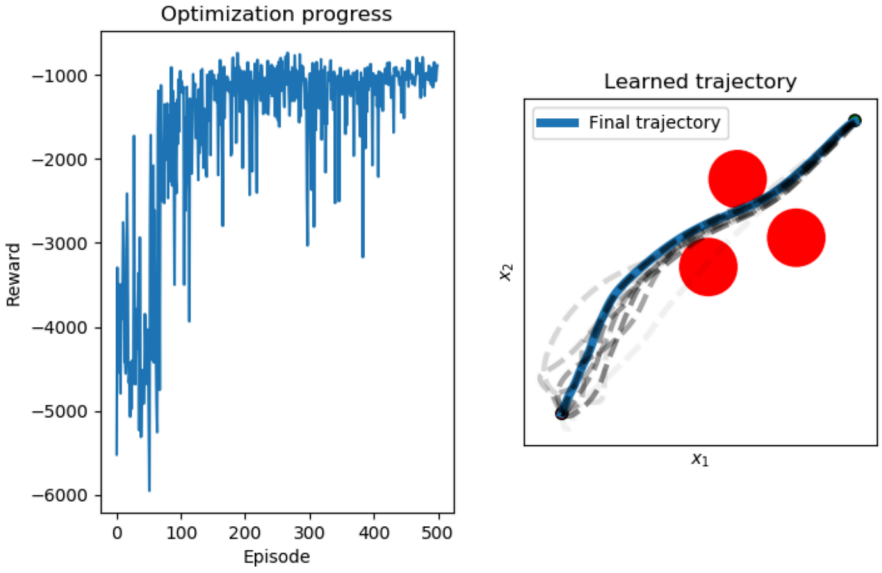


Figure 6.3.: Left: learning curve for 2D planning problem. Right: 2D environment. Big circles represent obstacles, a small circle represents the start position, a small green circle the goal, a blue line indicates the final path, and a dashed lines show intermediate solutions. More transparency indicates higher age.

6.3.3. Reproducible Research

Reproducing results is a notoriously hard problem in the domains of robotics and machine learning. One goal of BOLeRo is to provide well-tested implementations of algorithms and applications. Furthermore, is should be possible to publish the exact learning configuration used in publications based on BOLeRo. When a simulation tool and model are used to produce scientific results, the experiments can often not be reproduced without publishing tools and models with their exact configuration.

Another idea that we pursue with BOLeRo is to produce reference implementations of existing algorithms. An example of how we imagine this to be done is the reimplementation of C-CMA-ES [Abd+17a]. The learning curves for C-REPS and C-CMA-ES in Figure 1 (a) and (b) of the original publication of Abdolmaleki et al. [Abd+17a] could be reproduced with BOLeRo's implementation of the algorithm (see Figure 6.4).

6.4. Related Software

In addition to BOLeRo the following open source software has been produced and released during this thesis.

pytransform3d: Learning robotic behavior from human demonstration requires a heterogeneous ecosystem with components such as a motion capture system, the learning platform, and a robot middleware. All of these have their own definitions of 3D rotations and transformations that are often not explicitly documented. pytransform3d [Fab19b] has clearly documented conventions and can be used to convert between those definitions. It has been released at

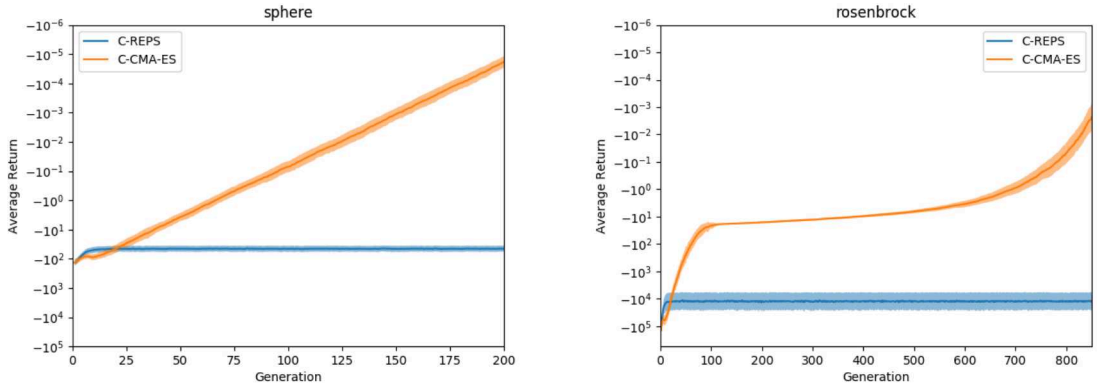
`https://github.com/rock-learning/pytransform3d`

approxik: The code for our version of the approximation of inverse kinematics that has been presented in [Fab20] is released as open source software at

`https://github.com/rock-learning/approxik`

6.5. Summary

BOLeRo provides a development and test environment for new learning approaches and scenarios. We split learning problems, learning algorithms, and behavior representation via defined interfaces (see Figure 6.1) which makes BOLeRo open for extensions and its parts reusable. We provide easy-to-use interfaces that support many of the common learning setups used in reinforcement learning and evolutionary computation. Additionally, by using the learning controller, BOLeRo is best suited to perform benchmarks of learning methods.



(a) (b)

Figure 6.4.: Reproduction of experiments from Figure 1 of Abdolmaleki et al. [Abd+17a]. Code and documentation is available at `https://github.com/rock-learning/bolero/tree/master/benchmarks/ccmaes`.

Related Publications

[Fab19] Alexander Fabisch. "pytransform3d: 3D Transformations for Python". In: *Journal of Open Source Software* 4.33 (2019), p. 1159. DOI: 10.21105/joss.01159.

[Fab20] Alexander Fabisch. "A Comparison of Policy Search in Joint Space and Cartesian Space for Refinement of Skills". In: *Advances in Service and Industrial Robotics*. Ed. by Karsten Berns and Daniel Görges. Springer, 2020, pp. 301–309. ISBN: 978-3-030-19648-6. DOI: 10.1007/978-3-030-19648-6_35.

[FLK20] Alexander Fabisch, Malte Langosz, and Frank Kirchner. "BOLeRo: Behavior Optimization and Learning for Robots". In: *International Journal of Advanced Robotic Systems* 17 (3 2020). DOI: 10.1177/1729881420913741.

The software BOLeRo is presented in Fabisch et al. [FLK20]. It is a joint work together with Malte Langosz who is responsible for the C++ interface, interface to the MARS simulation, and works with evolutionary algorithms. I am responsible for the Python interface and reinforcement learning algorithms. The content that has been reused in this chapter is entirely my own work.

pytransform3d is presented in Fabisch [Fab19]. Our solution to approximate inverse kinematics is described in Fabisch [Fab20].

Part IV.

Conclusion

Chapter 7.

Discussion

7.1. Contributions

The focus of this thesis is to develop machine learning algorithms and approaches to assist generation of discrete manipulation behaviors with an emphasis on trajectory generation. This thesis investigates several dynamic as well as static discrete manipulation behaviors such as throwing and grasping.

The first main contribution is the development of a procedure to automatically derive the embodiment mapping for policies that represent state space trajectories from motion capture data. The second contribution consists of several enhancements for contextual policy search algorithms: an active context selection approach and an active training set selection for C-REPS, an extension of C-CMA-ES with a surrogate model and an active covariance update, and a manifold learning approach that assists BO-CPS or any other contextual policy search algorithm with the goal to increase sample efficiency. These contributions are combined in a coherent framework that is easy to apply with a limited amount of expert knowledge.

The underlying categories of algorithms are the most robust and reliable ones from imitation learning and reinforcement learning that currently exist. This is not only demonstrated by presenting the best configuration and random seed but by repeating experiments in simulation 20 or more times and investigating the distribution of learning curves.

7.2. Experiments

This thesis evaluates imitation learning and contextual policy search on several problems such as grasping, throwing, pouring, or pulling a lever to show that the group of algorithms that was selected for this thesis is not just applicable to one specific problem.

Table 7.1 summarizes all experiments and all applications that are part of this thesis. As the focus is on robotic manipulation, all robotic systems that have been used are robotic arms with either 6 (UR5/10, COMPI) or 7 (Kuka iiwa 7/14, PA 10) degrees of freedom. UR5 and COMPI were used most often for applications with real systems since they are frequently available.

Table 7.1.: Overview of experiments and applications that have been presented in this thesis. See Appendix F for details on the robotic systems.

Section	Problem	IL / RL	Robot	Real system
3.1.3.2	Touhu throw	IL	UR5/10, Kuka iiwa 7, COMPI	✗
3.1.3.3	Touhu throw	IL	UR5	✓
3.2.3.2	Via point	RL	Kuka iiwa 14	✗
3.2.3.3	Obstacle avoidance	RL	Kuka iiwa 14	✗
3.2.3.4	Pouring	RL	Kuka iiwa 14	✗
4.1.3	Ball throw	RL	PA 10	✗
4.2.2.1	Black-box optimization	RL	✗	✗
4.2.2.2	Catapult	RL	✗	✗
4.2.2.3	Ball throw	RL	COMPI	✗
4.3.2.1	Black-box optimization	RL	✗	✗
4.3.2.4	Via point	RL	✗	✗
4.4.2.1	Ball throw	RL	COMPI	✗
4.4.2.2	Ball throw	RL	COMPI	✓
4.5.3.5	Reach	RL	✗	✗
4.5.3.6	Grasp	RL	UR5	✓
5.5	Ball throw	IL	COMPI	✓
5.5.3	Grasp	IL, RL	Kuka iiwa 14	✓
5.5.3	Pull lever	IL, RL	Mantis' arm	✓

7.3. Evaluation of Objectives

In Section 1.4 we define the objectives of this thesis. The first objective demands usability of behavior learning approaches. Chapter 5 contributes a framework to almost automatically generate manipulation behaviors and support robot behavior generation for non-experts to address this objective. Chapter 6 presents the underlying open source software. An important element of this framework is the automatic embodiment mapping that we discuss in Chapter 3. Although this framework does not solve the problem finally, it is a crucial step forward. Chapter 1 contributes a discussion of when behavior learning should be used and Chapter 2 contributes an extensive discussion of the state of the art in behavior learning for robots, which does not reduce the required expert knowledge but can be used to increase knowledge of non-experts.

The second objective is to learn behaviors with a maximum of 100–300 episodes and the third objective postulates that interesting skills generalize over several task parameters. We achieve both goals with BO-CPS in combination with the VTAE. So far our approaches are limited to only a few context variables and we do not incorporate high-dimensional, complex sensors or continuous sensor feedback. BO-CPS is sample-efficient (throwing: 80 episodes; grasping: 250 episodes) and generalizes over a specified context space but it requires a low-dimensional policy representation, which can be provided by the VTAE. For the scope of this thesis we achieved our goals, but the general problem of sample-efficient and generalizing behavior learning is far from being solved and it is unclear if it will ever be, since we currently have no idea of how behavior complexity and sample efficiency are related. We have no effective way of measuring behavior complexity, but it certainly depends on sensors and actuators of the robot as well as on the problem that we try to solve. Here we should improve quantitative evaluation to measure scientific progress better.

Although BO-CPS is evaluated only in two scenarios, the category of algorithms is evaluated, as we have just seen, in a variety of problems and robotic systems, which we demand in the fourth objective. Only robotic arms are considered, but these exhibit diverse structures as well as diverse kinematic and dynamic properties.

7.4. Limitations

7.4.1. Policies with Continuous and High-Dimensional Sensor Input

We learn reference trajectories that, for example, whole-body control [de +17] can use. Unlike many deep RL approaches, we do not learn reactive behaviors that map sensor measurements to actions. Instead, our approach is more deliberative, as we integrate one sensor measurement in the beginning of an episode as a context vector to generate a full trajectory. We also did not use high-dimensional contexts such as images in this thesis.

The episodic policy search approaches that are used in this thesis can be easily combined with smooth and stable trajectory representations such as DMPs or the VTAE, and they have no clear disadvantage compared with step-based RL, when reward is sparse and typically occurs at the end of an episode, or compared with value-function based RL, when temporal credit assignment is difficult, a sufficiently good policy is simple in comparison to its value function, or imitation learning can provide a good initialization. Hence, good examples of problems that can be solved well by episodic policy search with stable trajectory representations are grasping, throwing, and pulling a lever. Counter-examples are peg-in-a-hole and obstacle avoidance, which are better solved with continuous sensor feedback, and problems with multiple via points, which makes temporal credit assignment simple.

Consequently, we would like to learn reactive behaviors that handle high-dimensional inputs with neural networks and combine it with reliable learning and stable trajectory generation, which has been achieved with contextual policy search and DMPs. There is more potential here and the VTAE could be a first step to combine both approaches.

7.4.2. Automation of the Learning Platform

For automated behavior recording, marker free approaches could be tested and compared with respect to accuracy and achievable automation level. Also some prior knowledge is implicitly integrated in the design of the learning platform. There is not one combination of methods that works for all applications. For instance, simulation-reality transfer is only required in challenging applications such as throwing.

7.4.3. Reward

Besides markers for motion capture or simulations some prior knowledge has to be defined in form of reward functions. This can be complicated. The reward function for pancake flipping [KCC10b] is the most complicated one in a robotic context so far: besides requiring a motion capture system (which is acceptable), it considers the maximum altitude of the pancake, the orientation of the pancake, and the position of the pancake with respect to the center of the frying pan. Although the reward function is complicated, defining a solution is even more difficult. Nevertheless, for complex problems with complex reward functions it is desirable to have a more intuitive way of defining reward. There are several ways to take the burden of specifying a reward function from the human.

Reward functions can be optimized [SLB09], which shifts the problem of defining the optimization criterion to another level, that is, leads to the question: optimized with respect to what? This is, however, the way evolution works. A binary fitness function—extinction or not—results in the fine-tuned reward system that is integrated in each creature's body to increase its gene's ability to survive.

More practical approaches are the following. With active reward learning [Dan+15] the reward is directly given by a human and a surrogate model for the human's reward is trained to decrease the number of queries from the human. Inverse reinforcement learning [NR00] defines a reward function based on demonstrations. Kim et al. [Kim+17] avoid querying the human directly by reading error-related potentials, a signal that is implicitly generated by a human's brain, when an error occurs.

7.5. Impact and Relation to Other Fields

Let us take a look at how the publications on which this thesis is based have influenced other research and how this thesis is related to other fields.

Black-Box Optimization. There is a strong connection between policy search and black-box optimization. This led to the development of contextual CMA-ES [Abd+17a] based on the black-box optimizer CMA-ES [HO01] and BO-CPS [MFH15] based on Bayesian optimization [BCd10]. Furthermore, active contextual ACM-ES [Fab19a], contextual CMA-ES with a surrogate model, has been presented at the Genetic and Evolutionary Computation Conference (GECCO), one of the most important conferences of the black-box optimization community.

Goal-Conditional Deep RL. Contextual policy search is a multi-task reinforcement learning problem that is similar to goal-conditional reinforcement learning, which has been explored by the deep RL community. Relevant works in this context have been published by Schaul et al. [Sch+15a] about goal-conditional universal function approximators for DQN, Andrychowicz et al. [And+17] about hindsight experience replay for DDPG, Rauber et al. [Rau+19] about hindsight policy gradients, and Florensa et al. [Flo+18] about goal-generating neural networks for exploration.

Active Context Selection. After our initial publication about active context selection [FM14] several other works followed. We summarize the most interesting approaches. Metzen [Met15] builds on BO-CPS and extends it with active context selection based on entropy search [HS12]. Pinsler et al. [Pin+19] present a version of BO-CPS with a form of experience replay and active context selection. They test their algorithm in environments that are similar to those in which HER [And+17] has been tested. Although their reward is more informative than it was for HER, they were able to learn in orders of magnitude fewer episodes, which suggests that HER is not optimal for these tasks. Forestier and Oudeyer [FO16] extend our idea of using a multi-armed bandit algorithm to high-dimensional spaces of sensorimotor models for tool use. Florensa et al. [Flo+18] apply active context selection to policy gradient algorithms with neural networks. Although they cite our work, they develop a different approach based on a generative adversarial network [Goo+14] that generates goals. Char et al. [Cha+19] present a new approach that is similar to BO-CPS with active context selection and apply it to the domain of nuclear fusion.

Bayesian Optimization for Contextual Policy Search. Besides the previously mentioned works that build on Bayesian optimization for contextual policy search, Metzen [Met16] extended BO-CPS with minimum regret search, an acquisition function similar to the entropy search acquisition function [HS12]. Yang et al. [Yan+17] build on BO-CPS and apply it to the domain of walking with a simulated, tiny six-legged robot. They learn to walk at different speeds, uphill, and in a curve. Follow-up work has been published by Liao et al. [Lia+19] in which the robot's morphology is optimized in steps that are interleaved with learning to walk.

Manifold Learning for Optimization. Besides reinforcement learning for robots there are other potential application areas for manifold learning with contextual or standard black-box optimization. An interesting similar work in a different field has been published by Gómez-Bombarelli et al. [Góm+18], who use a VAE to encode molecule structures in the latent space and then train a model that predicts properties of the molecules with GPR based on a large dataset. Their main goal was not to reduce the dimensionality of the search space but to make it continuous by manifold learning. Continuous optimization on properties can then be performed to identify new molecules that might have desired properties, for instance, being a potential new drug.

PUBSVE. The majority of regression algorithms approximate a hypothetical function that generates the training set. Sometimes Gaussian measurement noise is assumed to be part of the training set so that in addition to the mean we estimate the (co)variance of the noise. In reality this is often a drastic simplification. We have to think of the reality more as a distribution that is not Gaussian or not any other simple, parametric distribution because it is often more complicated. Think of the state transition model in reinforcement learning, which is inherently stochastic, and if we approximate it, we often assume a Gaussian (for example, [DR11; Lev+16]). If, however, the distribution has two modes this will not work. We have to think of other ways to approach regression and find other descriptions of the conditional distribution that generates the training set.

The PUBSVE, which has been developed in collaboration with Krell [Kre15], approximates the upper bound of the output distribution instead of its mean. There is no algorithm that solves the exact same problem to the best of our knowledge. Using negative target values for training, we can approximate the lower bound, too. It is also possible to implement a similar loss function for a neural network, which would have the advantage that both, the lower and the upper bound, can be predicted in a single forward pass with multiple heads and shared internal representation. This approach is similar to recent work of Rodrigues and Pereira [RP20] in a field of research that is called quantile regression. Similarly, Meinshausen [Mei06] use a tree-based ensemble approach to predict several quantiles of a conditional distribution.

Learning Platform. Peng et al. [Pen+20] use a framework to learn locomotion behaviors that is similar to the framework introduced in Chapter 5. They learn locomotion behaviors for a quadrupedal robot from motion capture of dogs. Their work fits in our framework, as they use imitation learning of Cartesian trajectories with embodiment mapping and refinement based on reinforcement learning as well as simulation-reality transfer. Their methods are similar, but the details differ. They transfer manually defined Cartesian key points from the dog to a similar robotic system and use inverse kinematics to obtain joint angles. They optimize individual joint angles to produce a similar Cartesian trajectory and minimize the distance of each angle to a default pose at the same time to make trajectories simpler and smoother. Policy refinement by reinforcement learning then makes simulated postures and velocity of the robot during locomotion more similar to the demonstrated reference. Domain randomization is used to make the policy robust against changes in the dynamics model before transfer to a real robot.

Computer Graphics and Video Games. A common problem that we see in computer graphics is character animation from motion capture data, which often requires complicated manual motion retargeting. Although full automation is not possible yet, we see potential to assist and speed up the process of motion retargeting with the methods that we develop here.

While all the behavior learning algorithms and applications that we considered here are always evaluated with respect to their usefulness for robotics, they can all be applied to character animation for computer graphics or to solve video games as well.

Then, however, we have to evaluate algorithms differently. In computer graphics it is often cheaper to do more episodes than to use a computationally expensive algorithm. Parallelization becomes more important, as we can scale training to large hardware resources. Therefore, evolution strategies, a simple but highly parallelizable algorithm, is popular for solving video games [Sal+17; CLH18; Fuk+19].

7.6. Insights

Apart from its main matter of research, each dissertation teaches its author several lessons that may be forgotten if they are not written down. The following list summarizes these.

- **Deep RL doesn't work yet.**[1] Although deep RL receives a lot of attention, policy search and black-box optimization are robust, sample-efficient, and work well. Deep learning is a promising direction of research in behavior learning, but we should always compare it with these methods and, ideally, combine approaches and ideas from both worlds.
- **Black-box optimization is policy search.** There is a strong connection between policy search and black-box optimization. These fields should exchange more ideas. Currently, black-box optimization seems to have more to offer for RL than vice versa.
- **Prior knowledge is important.** Trajectory generators such as movement primitives can be an important building block, even for deep RL. Integration of prior knowledge (inverse kinematics, movement primitives, imitation learning) is still an underestimated tool to make reinforcement learning more robust and sample-efficient. We can use much more prior knowledge in robotics.
- **Unsupervised learning is the bridge to deep RL.** Manifold learning (which is unsupervised) for policy parameters can be used for transfer learning and could be a bridge between imitation learning with policy search and deep reinforcement learning. Deep reinforcement learning could become sample-efficient through transfer learning.

7.7. Publications

Journal Articles

[Fab+13] Alexander Fabisch, Yohannes Kassahun, Hendrik Wöhrle, and Frank Kirchner. "Learning in compressed space". In: *Neural Networks* 42 (2013), pp. 83–93. ISSN: 0893-6080. DOI: 10.1016/j.neunet.2013.01.020.

[1] An allusion to Irpan [Irp18].

[Fab+15] Alexander Fabisch, Jan Hendrik Metzen, Mario Michael Krell, and Frank Kirchner. "Accounting for Task-Difficulty in Active Multi-Task Robot Control Learning". In: *KI – Künstliche Intelligenz* 29.4 (2015), pp. 369–377. ISSN: 1610-1987. DOI: 10.1007/s13218-015-0363-2.

[Fab+20] Alexander Fabisch, Christoph Petzoldt, Marc Otto, and Frank Kirchner. "A Survey of Behavior Learning Applications in Robotics—State of the Art and Perspectives". In: *International Journal of Robotics Research* (2020). Submitted.

[Fab19] Alexander Fabisch. "pytransform3d: 3D Transformations for Python". In: *Journal of Open Source Software* 4.33 (2019), p. 1159. DOI: 10.21105/joss.01159.

[FLK20] Alexander Fabisch, Malte Langosz, and Frank Kirchner. "BOLeRo: Behavior Optimization and Learning for Robots". In: *International Journal of Advanced Robotic Systems* 17 (3 2020). DOI: 10.1177/1729881420913741.

[FM14] Alexander Fabisch and Jan Hendrik Metzen. "Active Contextual Policy Search". In: *Journal of Machine Learning Research* 15 (2014), pp. 3371–3399. URL: http://jmlr.org/papers/v15/fabisch14a.html.

[Gut+18] Lisa Gutzeit, Alexander Fabisch, Marc Otto, Jan Hendrik Metzen, Jonas Hansen, Frank Kirchner, and Elsa Andrea Kirchner. "The BesMan Learning Platform for Automated Robot Skill Learning". In: *Frontiers in Robotics and AI* 5 (2018), p. 43. ISSN: 2296-9144. DOI: 10.3389/frobt.2018.00043.

[Met+14] Jan Hendrik Metzen, Alexander Fabisch, Lisa Senger, José de Gea Fernández, and Elsa Andrea Kirchner. "Towards Learning of Generic Skills for Robotic Manipulation". In: *KI – Künstliche Intelligenz* 28.1 (2014), pp. 15–20. ISSN: 1610-1987. DOI: 10.1007/s13218-013-0280-1.

Conference Publications

[Fab19] Alexander Fabisch. "Empirical Evaluation of Contextual Policy Search with a Comparison-based Surrogate Model and Active Covariance Matrix Adaptation". In: *Genetic and Evolutionary Computation Conference Companion*. Ed. by Manuel López-Ibáñez. GECCO '19. ACM, 2019, pp. 251–252. ISBN: 978-1-4503-6748-6. DOI: 10.1145/3319619.3321935.

[Fab20] Alexander Fabisch. "A Comparison of Policy Search in Joint Space and Cartesian Space for Refinement of Skills". In: *Advances in Service and Industrial Robotics*. Ed. by Karsten Berns and Daniel Görges. Springer, 2020, pp. 301–309. ISBN: 978-3-030-19648-6. DOI: 10.1007/978-3-030-19648-6_35.

[FK20] Alexander Fabisch and Frank Kirchner. "Variational Trajectory Autoencoder for Sample-Efficient Policy Search". In: *Conference on Robot Learning (CoRL)*. Submitted. 2020.

[Gut+19] Lisa Gutzeit, Alexander Fabisch, Christoph Petzoldt, Hendrik Wiese, and Frank Kirchner. "Automated Robot Skill Learning from Demonstration for Various Robot Systems". In: *KI: Advances in Artificial Intelligence*. Ed. by Christoph Benzmüller and Heiner Stuckenschmidt. Springer International Publishing, 2019, pp. 168–181. ISBN: 978-3-030-30179-8. DOI: 10.1007/978-3-030-30179-8_14.

Workshop Publications

[MFH15] Jan Hendrik Metzen, Alexander Fabisch, and Jonas Hansen. "Bayesian Optimization for Contextual Policy Search". In: *Machine Learning in Planning and Control of Robot Motion (MLPC) Workshop, IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Ed. by Aleksandra Faust. 2015. URL: https://www.cs.unm.edu/~afaust/MLPC15_proceedings/MLPC15_paper_Metzen.pdf.





Chapter 8.

Outlook

> What percentage of human's manipulative repertoire have robots mastered? Nobody can answer this question.
>
> *(Mason [Mas12])*

Parts of this chapter were published originally as [Fab+20].

We close this thesis with an outlook on the future of behavior learning. Let us discuss how we can further improve sample efficiency and which problems we should approach.

8.1. Ways to Simplify Learning Problems

Learning problems can be simplified by learning (1) not everything from scratch (knowledge transfer), (2) not everything end to end (combination with other methods), (3) while a safe, deliberative method is operating, or in a controlled environment (bootstrapping).

Knowledge transfer: Knowledge can be transferred from similar tasks, similar systems, or similar environments. In the optimal case we use multiple almost identical robots to learn the same task in the same environment [Gu+17; Lev+18]. Levine et al. [Lev+18] show that transferring a training set from one robot to another robot, which is solving a similar task in the same environment, is beneficial (if actions are represented in task space). Levine et al. [Lev+16] show that pretraining neural networks on different but related tasks is the key to success when complex behaviors are trained end to end.

More research should be done on lifelong learning:

> Lifelong Machine Learning, or LML, considers systems that can learn many tasks over a lifetime from one or more domains. They efficiently and effectively retain the knowledge they have learned and use that knowledge to more efficiently and effectively learn new tasks. [SYL13]

It could lead to robust, sample-efficient artificial intelligence that is able to solve a multitude of tasks and share knowledge between them. We believe that this can be much more efficient than learning from scratch. Coming back to the example of in-hand manipulation [Ope+20], perceiving the object's pose or several strategies used in the manipulation behavior are components that could be shared with many other tasks that are related to manipulation of movable objects.

We have to find ways to share knowledge between similar and dissimilar robots and tasks. In theory, sharing knowledge between robots in form of training sets or pretrained models is much easier than sharing knowledge between humans that can only absorb knowledge through their senses. Bozcuoglu et al. [Boz+18] propose a similar approach: they share ontologies and execution logs on the cloud platform openEASE. The knowledge can be transferred to other environments or other robots. The same approach could be used to share pretrained models or training data to learn behaviors.

Combination with other methods: Combining existing approaches for perception and state estimation with machine learning has been shown to be effective by Mülling et al. [Mül+13] and Parisi et al. [Par+15]. Similarly, combining existing approaches for planning and machine learning has been shown to be effective by Lenz et al. [LLS15]. Also model predictive control has been combined with a learned uncertainty-aware perception model by Kahn et al. [Kah+17]. Nemec et al. [NŽU17] combine machine learning and structured search with physical constraints. To generate walking behaviors, often classical models such as a linear inverted pendulum [Kaj+01] are used and a zero moment point (ZMP) [VB05] is computed. Only parts of complex walking behaviors are learned. We think this is a good method to verify and understand what is happening on the system, to reduce the amount of physical interaction with the world that is required to learn the behavior, and to obtain solutions that are safe. Geng et al. [GPW06] confirm this for their application: "Building and controlling fast biped robots demands a deeper understanding of biped walking than for slow robots." Englert and Toussaint [ET18] write: "One way to reduce [...] difficulties is by exploiting the problem structure and by putting prior knowledge into the learning process." Although Loquercio et al. [Loq+18] show remarkable results with an approach that learns collision avoidance on a drone almost end to end, they do not want to replace map-localize-plan approaches and believe that "learning-based and traditional approaches will one day complement each other".

Nonetheless, we must not restrict the amount of learnable behaviors by introducing too strong constraints or too simple models. For example, requiring the ZMP to be in a support polygon is a strong limitation. It is an artificially constructed, simple model of dynamical stability, which has been developed to avoid at all costs that expensive robots fall and break. It limits the capabilities of a robot, as running would be difficult to implement with a ZMP approach. Yang et al. [YKL17] argue that this approach prohibits advanced balancing behaviors. Making basic physical knowledge available to the learning algorithm can be beneficial without restricting the amount of learnable behaviors though. As an alternative to the ZMP approach, we can compute the centroidal momentum [OG08; OGL13] and make it available to the learning algorithm. When a translation from joint space to Cartesian space is required or useful, we can use the Jacobian. For dynamics we can use the equations of motion.

Boostrapping: A situation, in which the combination of behavior learning with another method is safer, is manipulation with a superimposed collision avoidance behavior.

While the robot is learning to grasp, it can safely be guided around obstacles. These safety mechanisms could also be used to bootstrap learning and collect data safely before we shift to the pure learned behavior that might perform better. It is even possible to use additional equipment or a controlled environment to provide additional information to bootstrap learning. This has been done, for example, by Levine et al. [Lev+16] to reduce the required amount of data. Englert and Toussaint [ET18] also demonstrate that a combination of optimal control, episodic reinforcement learning, and inverse optimal control in the training phase can be safe and efficient. The problem of safe exploration has also been discussed in more detail by Amodei et al. [Amo+16, pages 14–17].

8.2. Integration of Prior Knowledge in Deep Learning

Among the previously proposed paths, combining behavior learning with existing methods is the most promising path to sample-efficient learning of complex behaviors. We cannot stress enough the need for the right inductive biases to simplify the learning process. Deep learning specifically can benefit the most from this. We will take a closer look at how prior knowledge can be integrated in deep neural networks for behavior learning.

Ever since the term deep learning [Sch14; LBH15] has been popularized, the field generated impressive results. Outstanding successes in computer vision [KSH12] with subdisciplines such as object detection [Lin+17; RF18] and semantic segmentation [SLD17; YCW19], speech recognition [Han+14], machine translation [Vas+17], and playing video [Mni+15] and board games [Tes95; Sil+16] usually have the following unsolved problems: they require huge amounts of data and large computational resources, they are hardly explainable, and they often do not generalize well [Sze+14; Jac+19].

In the last years deep reinforcement learning has been driven to an extreme. Silver et al. [Sil+16] implement a distributed AlphaGo that uses 1,202 CPUs and 176 GPUs to play a single game. OpenAI et al. [Ope+19a] learned to play the video game Dota 2 with 45,000 years of game experience and beat professional humans [Ope19] with 128,000 CPU cores and 256 P100 GPUs [Ope18]. OpenAI et al. [Ope+19b] learn low-level hand control skills for Rubik's cube with roughly 13 thousand years of experience in simulation. This development prevents normal researchers from generating state-of-the-art results because of outrageous requirements on computational resources. Such high computational requirements have a negative environmental impact [Sch+19]. In addition, we mostly solve perception problems, although in robotics we will have to solve much more complex problems that combine perception and action while autonomous robots only have limited resources.

What makes a neural network? It is a hierarchical, parameterized function. It has a loss of which we compute the gradient with respect to the parameters of the network by backpropagation and a gradient-based optimizer reduces the loss. Generalized backpropagation is a form of automatic differentiation: the reverse accumulation mode [Bay+18]. It is efficient for scalar functions with lots of parameters. Neural networks were a major driver for the development of automatic differentiation engines for tensors. Examples are Theano [The16] (first release in 2009), MXNet [Che+15b], TensorFlow

[Aba+15], PyTorch [Pas+19] (first release in 2016), or JAX [Bra+18]. Because gradient-based optimization can be used for more than machine learning this led to the advent of a new programming paradigm—*differentiable programming*. The idea can be traced back to Olah [Ola15] who used the term differentiable functional programming. Karpathy [Kar17] developed a similar idea under the term Software 2.0. It means that we can write software that has parameters and everything is differentiable so that we can optimize it. With this approach we can, for example, optimize a browser for a specific user. This demands not only for powerful automatic differentiation tools but also for an integration in the programming language. Scientific programming languages such as Julia [Bez+17] are predestined to popularize differentiable programming. Innes et al. [Inn+19] introduced Julia libraries that enable differentiable programming. They focus on the links of machine learning and scientific programming. Since problem definitions that are implemented in Julia are differentiable we can also train neural networks that solve these problems by gradient-based optimization. In 2019, the *Differentiable Programming Manifesto* of the programming language Swift has been released [Wei+19], which is an outline for the introduction of differentiable programming as a core concept of a compiled language that is not mainly used by researchers.

We will give examples of prior knowledge that can be integrated in differentiable programs or neural networks. The first deep neural networks that could be trained end to end are convolutional neural networks [LeC+89]. They exploit the 2D structure of the data that they process (mostly images). Similarly, attention mechanisms in deep neural networks are a successful architecture pattern for machine translation [Vas+17]. Kassahun et al. [Kas+08] integrate a simplified version of a *Kalman filter* in a neural network that is trained without gradients. A Kalman filter [Kal60] can be used to track objects that are recognized by a perception component. Jonschkowski et al. [JRB18] integrate a *particle filter* that can be used for state estimation in a neural network. The integration of this well-known algorithm fosters explainability and the combined system of neural network and particle filter generalizes better. Jonschkowski et al. [JRB18] conclude that "the use of algorithms as algorithmic priors will help to realize the potential of deep learning in robotics". Degrave et al. [Deg+19] present a differentiable *rigid body physics engine* to simulate robots. Hu et al. [Hu+20] present a differentiable *physics simulation* of elastic objects, incompressible fluids, a mass-spring system, billiard, rigid body collisions, water rendering, volume rendering, and electric fields. Engel et al. [Eng+20a] present a differentiable *digital signal processing* library. They integrate it in a neural network that generates sound. This strong inductive bias does not reduce the expressive power of the network in this domain. It even enables extrapolation, which is unusual for deep neural networks. Lutter et al. [LRP19] integrate *Lagrangian mechanics* to learn inverse dynamics. Pavllo et al. [PGA18] use *quaternions* in a network and *forward kinematics* in a loss. Graves et al. [GWD14; Gra+16] integrate a differentiable *memory module* and Grefenstette et al. [Gre+15] introduce network modules that represent data structures such as *stacks*, *queues*, and *deques*. Blondel et al. [Blo+20] propose differentiable *sorting* and *ranking* algorithms and Berthet et al. [Ber+20] present differentiable *optimization*. All of these works show that we can integrate prior knowledge in neural networks to in-

crease sample efficiency, foster explainability, and generalize in a more controllable way. These properties are important for robotics, but also adaptation through learning, since often not everything can be modeled perfectly [TM95].

Previously, the robotics community often preferred imitation learning and policy search with simpler function approximators over deep learning [DNP13; Gut+18]. Nevertheless, it is desirable to use neural networks, as they are powerful function approximators and we can benefit from the successes of deep learning such as extraordinary sensor processing, especially in computer vision. There are previous success stories that rely either on pretraining with simpler models or huge amount of data. Levine et al. [Lev+16] demonstrate that deep reinforcement learning based on policy search can successfully learn various manipulation skills end to end from camera images to torque commands. Levine et al. [Lev+18] show impressive grasping behaviors of a simple two-finger gripper that have been learned in a self-supervised fashion. More than 800,000 grasp attempts have been used to generate this result. We would like to learn with fewer samples to avoid costs and high energy consumption not just during training but also during execution. For mobile robots this will be particularly important because the amount of energy is limited to the battery that they carry.

It is important to integrate prior knowledge in a way that does not limit the expressive power of neural networks. Different domains require different forms of prior knowledge that will be integrated in neural networks. Prior knowledge can be integrated in forms of new layer types, architecture patterns, loss functions, or pretraining. We should use it to improve sample efficiency and generalization in deep learning. We expect gradient-based training from supervision or reinforcement learning to complement classical approaches from robotics extraordinarily well, since these often have many parameters that have to be set, which is often done manually.

8.3. Comparability and Reproducibility

Shadmehr and Wise [SW05] convey the idea that the same computational principles that allow earlier forms of life to move in their environment later enabled higher forms of intelligence: language and reasoning. The intelligence of animals and humans evolves with the complexity of the problems that it solves. An example for this is confirmed by Faisal et al. [Fai+10], who investigate production of early prehistoric (Oldowan) and later (Acheulean) stone tools. Oldowan tools are simpler and their production require less complex behaviors. The production of Acheulean tools requires the activation of brain regions associated with higher-level behavior organization. The development of more complex behavior coordination is even linked to development of more complex forms of communication, since the development of complex manipulation behaviors required more intellectual capacities and these could also be applied to language. It is an important finding for us as roboticists. Translating this to our work, this means more complex problems require the development of better behavior learning algorithms. These algorithms could potentially also be used in other domains for which they have not been designed originally. Hence, advancing at both frontiers could benefit the whole field.

Artificial intelligence has advanced by setting challenging goals. For example, the problem of playing chess against a human or the RoboCup initiative, which has a similar goal but combines AI with robotics: "The Robot World-Cup Soccer (RoboCup) is an attempt to foster AI and intelligent robotics research by providing a standard problem where a wide range of technologies can be integrated and examined." [Kit+97] In recent years we have seen major advances in reinforcement learning also because clearly defined benchmarks are available, for instance, the Atari learning environment [Bel+13] and OpenAI Gym [Bro+16]. These benchmarks make comparisons of existing approaches easier. It is also simpler to reproduce results because it is easy to check if a reimplementation of an algorithm gives the same result as in the original publication. Hence, we recommend to define benchmarks for robotic behavior learning.

A problem is that often similar problems are solved but with varying conditions. In the context of grasping we observed that the objects are often different, although there are standardization efforts: the YCB object and model set is an example [Cal+15a; Cal+15b; Cal+17]. These efforts have to be fostered and supported. Also new benchmarks have to be created. For these we can draw inspiration from the diagnosis and treatment of human patients. An example of a benchmark for humans is the box and block test [MFW85], in which a patient has to move colored blocks from one box to another as fast as possible. We think that a set of benchmark problems should be selected, standardized, formalized, and described in detail so that results are easily comparable.

Games and sports are particularly good candidates for benchmark problems because they have a clear set of rules, standardized material, they are usually easy to understand, and offer a variety of challenging problems. We have seen that a large number of behavior learning problems that have been tackled already come from this domain. Mostly subproblems, for example, kicking or batting a ball, have been extracted and learned. More advanced benchmarks would also include tasks with less strict rules, for example, setting a table.

Benchmarking in the context of robotics, however, is difficult because software cannot be tested in isolation. Simulations could be used to address this problem, but they often lead to solutions that are not transferable to reality, neither the learned behavior nor the learning algorithm. The RoboCup Standard Platform League (SPL) solves this problem by requiring that each competing team uses the same hardware. This is not an optimal solution because most robots are expensive and most research institutes are not able to buy a new robot just to compete in a specific benchmark. We can offer no perfect solution for this problem. We can only propose that a cheap robotic platform that is sufficient enough for a variety of benchmarks should be developed.

8.4. The Future of Behavior Learning Problems in Robotics

Mason [Mas12] writes: "What percentage of human's manipulative repertoire have robots mastered? Nobody can answer this question." We can say exactly the same about any other category of robotic behaviors. At least we have a rough overview of behaviors that

have been learned from Chapter 2 and Appendix A. We will now talk about what is still missing.

At the moment, most behaviors are learned in isolation. On a complete system, the learned behavior will interfere with high-level behaviors and other behaviors on the same level that might even have higher priority such as balancing or collision avoidance. There might even be other learned behaviors, for example, a learned walking behavior and a learned throwing behavior could be executed in parallel. Executing multiple behaviors in parallel has effects on the whole system. These problems are neglected if behaviors are learned in isolation. Throwing a ball while walking makes the balancing part of the walking behavior more difficult and grasping an object while collision avoidance is active might result in different reaching trajectories. Sometimes combining two behaviors might require one of these behaviors to be changed completely. For example, in the case of throwing while running, the whole locomotion and balancing behavior might have to be altered to anticipate and absorb high forces that are exerted during the throw (for example, in a javelin throw).

Figure 8.1 illustrates two possible roadmaps for walking behaviors. Currently, we are able to learn walking for robots with four or more legs. There are two alternative routes illustrated that we could take from there: the *ball sports route* and the *parkour route*. Ball sports in this example include soccer, basketball, or handball. It is to some extent possible to learn bipedal walking, which requires more advanced balancing behavior than walking with more legs. Fast bipedal running is already a much more complex task because it is a highly dynamic behavior that cannot easily be solved with classical stability criteria and control approaches. Running and dribbling a ball requires to solve a much more complex perception problem and precise foot placement or hand movements. Combining this behavior with the requirement to throw or kick a ball will introduce a difficult coordination problem: throwing will have an impact on the balancing part of the running behavior. A good solution will predict this impact and counteract already while performing the throw. Nevertheless, throwing a ball to a fixed goal is easy in comparison to passing the ball to a teammate, when the robot has to anticipate the behavior of the teammate to pass the ball to a location where the teammate will be able to use it. Another future research direction could be over climbing to parkour. Legged robots unfold their full potential in rough and irregular terrain, where precise perception of the environment, foot placement, and robust balancing is required. This has been learned already to some extent. A more difficult scenario would be climbing up a mountain with steep slopes, where not only feet but all body parts must be controlled, for example, a humanoid would have to use its arms. The robot must be flexible enough to balance on steep and rough terrain. A next possible step would be among the most difficult sports that humans are able to perform: parkour. It requires to *understand* the environment, that is, know what you can do with it to find the fastest and direct way by overcoming obstacles. The whole body is involved and it is often required to turn off basic safety mechanisms, for example, to perform a double kong vault where the body is almost turned upside down with the hands on the obstacle directing momentum and the feet above the head to get out of the way.

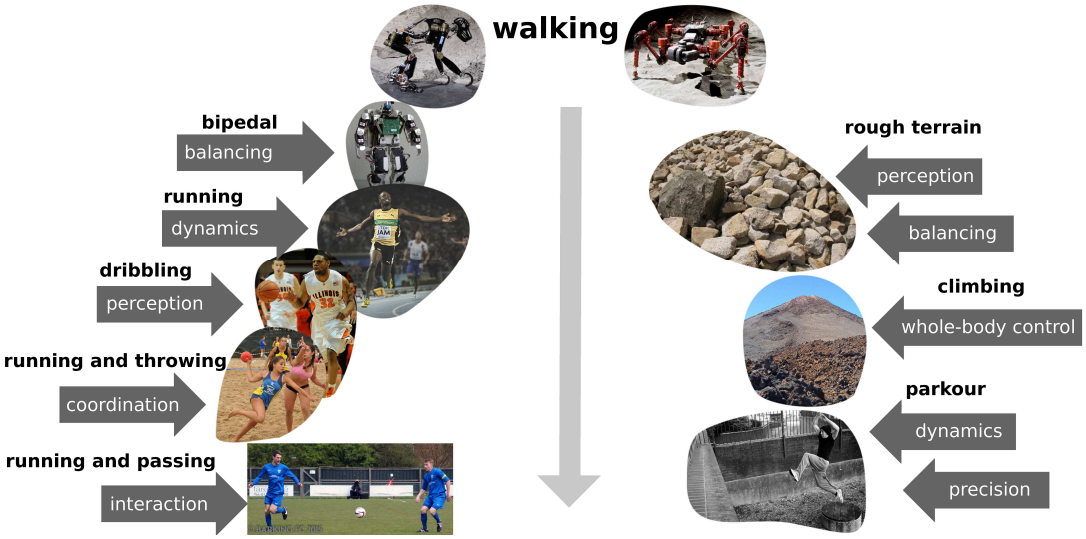


Figure 8.1.: Roadmaps for walking robots.[1]

There are low-hanging fruits to increase the spectrum of learned behaviors. Examples are the locomotion behaviors running, climbing ladders, jumping over obstacles, jumping precisely or jumping as high as possible with one or two legs, front or back flip, swimming, and paddling. In the kitchen domain: stirring, chopping, opening cans or bottles. In the household domain: the problems of folding sheets or clothes can be challenging because these problems are hard to model. In the manufacturing domain: the skills of hammering, screwing, sewing, shoveling, and tool use in general are relevant. While perception has been fully learned for grasping and collision avoidance, this has not been considered so far for dynamic problems such as catching, batting, or kicking balls. There is a limited amount of publications concerned with learning high-level game playing in real physical games, for example, to learn coordination of multiple robots in soccer. For interaction with humans, performing gestures and other physical interaction behaviors such as various forms of hand shaking could be learned. Interesting balancing problems often come from sports, for instance, surfing, skating, or skiing.

[1] Image sources: running from Stephane Kempinaire (URL: http://www.mynewsdesk.com/se/puma-nordic/images/puma-aw14_ff_bolt-325510; license: CC BY 3.0), dribbling from flickr user tsavoja (URL: https://www.flickr.com/photos/tsavoja/4106568938/; license: CC BY-SA 2.0), throwing while running from flickr user RFEBM Balonmano (URL: https://www.flickr.com/photos/125948220@N02/14826033503/; license: CC BY-SA 2.0), passing while running from flickr user Terry Gilbert (URL: https://flic.kr/p/QDhaKN; license: CC BY 2.0), parkour from flickr user THOR (URL: https://www.flickr.com/photos/geishaboy500/3090363361/; license: CC BY 2.0), all other photos are from DFKI RIC and can be found at https://robotik.dfki-bremen.de/

There are not many learned behaviors that require advanced spatiotemporal and causal reasoning beyond unscrewing a light bulb. More examples for this kind of problems are assembling furniture, tidying up a room, cooking a complete meal, or solving puzzles.

Creating a system that solves not just one problem but a variety of complex tasks is even more difficult. It involves integration of hardware components, software components, and behaviors. Building complex systems is a challenge in itself, but it is required to create more sophisticated complex behaviors.

Learned behaviors can usually not be explained. Robots cannot reason about them. They cannot explain why they selected a certain action or why it works. We have not yet seen robots that combine existing learned behaviors to new sequences or combinations of behaviors to solve tasks that they have not seen before.

As we already argued in the introduction, given the current development in behavior learning and in computer vision, we expect that the next big steps will be made by deep learning and by solving more and more complex perception problems. This direction of artificial intelligence research has its justification in Moravec's paradox: "it is comparatively easy to make computers exhibit adult level performance on intelligence tests or playing checkers, and difficult or impossible to give them the skills of a one-year-old when it comes to perception and mobility" [Mor88, page 15]. We emphasize, however, that for complex behaviors not only complex perception but also complex control is required. It is not sufficient to control a 7 DOF arm to realize a versatile, flexible, and autonomous humanoid robot. We should strive towards pushing the limits of kinematic complexity. A good example is the work of OpenAI et al. [Ope+20], who control a complex hand that is similar to a human hand.

In summary, there is still a long way to go to build robots that are able to perform as good as humans in these tasks, but we think that learning behaviors is the best way that we have to acquire these skills, when the robotic hardware is sufficient. Mason [Mas12] formulated a conjecture about robotics research: "[I]t is just possible that our field is still in its infancy. I do not have a compelling argument for this view, but it is telling that we have no effective way to measure our progress toward long-range goals." Our outlook on which skills we should try to master by behavior learning in the future, particularly the discussion of the roadmap displayed in Figure 8.1, also is a confirmation of this.

Although the influence of one researcher is minuscule in this domain, this thesis postulates the basis for future developments in behavior learning and explores basic algorithms and concepts that can be extended and developed further in the future.

Part V.

Appendix

Appendix A.

Survey of Behavior Learning Problems

This appendix was published originally as [Fab+20] and has been revised.

This appendix includes a more detailed discussion of the publications that have been presented in Section 2.1 *Robotic Behavior Learning Problems*. It was first published by Fabisch et al. [Fab+20] and thus is a joint effort of the authors of this survey paper. The remainder of this section is separated in manipulation behaviors, locomotion behaviors, and behaviors that do not fit in any of these categories. Table A.1 summarizes the behavior learning problems, corresponding publications, and their categorization.

We capture a large variety of robotic behavior learning problems according to the presented definition of behavior. We group problems according to the categories that we introduced and point out similarities and differences between and difficulties of these problems.

A.1. Manipulation Behaviors

A.1.1. Fixed Objects (A)

Flipping a light switch: Buchli et al. [Buc+11] investigate the task of flipping a light switch. The switch essentially is a via point that has to be passed through precisely in this kind of task. In addition to high accuracy, the flipping process itself requires the exertion of forces. In their work, the robot learns to be compliant when it can be and be stiff only when the task requires either high precision or exertion of forces. The problem could be extended to the recognition of the switch, which is not done here.

Open door: In contrast to flipping a switch, opening a door does not require precise trajectories. Additionally, more than just a via-point problem has to be solved: opening a door involves grasping the handle, closing the kinematic chain between gripper and the handle and finally moving the handle. The movements of the robot after grasping are restricted by the structure of the handle. Opening a door requires significant force exertion from the robot to the environment. Nemec et al. [NŽU17] ignore the problem of grasping and only consider the problem of learning the unconstrained DOFs while the kinematic chain from the robot to the door is closed. Chebotar et al. [Che+17b] and Gu et al. [Gu+17] consider the problem of learning this behavior end to end from camera images to motor torques. Nemec et al. [NŽU17] and Englert and Toussaint [ET18] ignore the perception part of the problem and assume known relative positions. Kalakrishnan et al. [Kal+11b] and Kormushev et al. [KCC11] use force sensors. The door considered

by Kormushev et al. [KCC11] does not have a handle but a horizontal bar that has to be pushed with a larger force than a standard door handle. It is also the only work in which the door has been pushed and not pulled. Nemec et al. [NŽU17] and Englert and Toussaint [ET18] consider not only horizontal but also vertical handles.

Turning objects: Several manipulation problems involve turning fixed objects, for example, turning a valve [Car+12], or a crank [Pet+14], or screwing a cap on a (pill or water) bottle [Lev+16]. The challenge is to reach a via point and then hold and move an object on a circular path. These behaviors can be realized as rhythmic movements [Pet+14] or discrete movements [Car+12; Lev+16]. They can be discrete when the object has to be turned only by a small angle (for example, 90 degrees, Carrera et al. [Car+12]) or when the robot can spin its wrist [Lev+16]. Some works focus more on robustly reaching the target object [Car+12; Lev+16] and others on robustly turning the object itself [Pet+14]. Carrera et al. [Car+12] exclude perception from learning, Levine et al. [Lev+16] learn perception and action, and Petrič et al. [Pet+14] follow previously learned torque profiles.

A.1.2. Spatially Constrained Behavior (B)

Peg-in-a-hole: Inserting a peg in a hole is one of the most basic manipulation skills that we discuss in this article. It is the most frequent assembly operation [GFB94]. The behavior can benefit from both visual [Lev+16] and force sensors [GFB94; Ell+12; Kra+16], but it can also be done without any sensors [Che+17a]. While the most obvious application of this skill is found in assembly tasks [GFB94; Ell+12; Kra+16; Lev+16], it can also be used to, for example, plug in a power plug [Che+17a]. The problem can be solved end to end from visual data to motor torques [Lev+16] or from force measurements to Cartesian positions [GFB94] as a purely reactive behavior. Alternatively, learning can be combined with search heuristics for the hole based on force measurements [Ell+12; Kra+16]. In the simplest case, the behavior is learned for a fixed relative transformation between robot and target [Che+17a].

A more advanced assembly operation that involves multiple instances of the peg-in-a-hole problem has been learned by Laursen et al. [Lau+18] to connecting a pipe for a heating system. In this task, a passively compliant gripper holds a tool extension and has to use a tube feeder, nut feeder, and crimping machine. Only actions were learned and a safety mechanism prevented the system from serious collisions. Apart from that, the system learns blindly without any sensors.

Wiping: The motion required to solve sweeping, wiping, ironing or whiteboard cleaning tasks can be either discrete or rhythmic. Further, all these tasks require environmental interaction by exerting (specific) forces on external objects. Learning mostly focuses on finding parameters for the representation of the movement. Kormushev et al. [KCC10a; KCC11] let a robot learn a discrete ironing skill from demonstrated trajectories and additional force profiles. They also evaluated their work on a whiteboard cleaning task

[Kor+11b]. A similar task is surface wiping which is investigated by Urbanek et al. [UAS04] and Gams et al. [Gam+14]. Both works represent the wiping skill as a periodic movement. In this case, rhythmic motions are advantageous, as the complete surface can be wiped easily by executing the motion several times while shifting only the center point. The work from Gams et al. [Gam+14] also uses force feedback to maintain contact with the surface. Besides the aforementioned household tasks, there are also industrial operations that require constant environmental contact. From these, grinding and polishing tasks have been investigated by Nemec et al. [Nem+18]. The goal of these tasks is to keep contact with a specific force exertion between a polishing/grinding machine and the treated object, which is manipulated by a robot with a desired orientation. Therefore, their approach reproduces the relative motion between object and tool. The contact point is estimated using measured forces and torques and can be changed to optimize a defined criterion, for example, minimize joint velocities. Sweeping has been considered by Alizadeh et al. [ACC14]. The position of dust is obtained using computer vision and the behavior is adapted accordingly. Pervez et al. [PML17] train a sweeping behavior end to end from visual inputs to collect trash placed at various positions between a fixed initial and goal position.

Handwriting: The goal of handwriting tasks is to resemble human writing as precise and smooth as possible. Complete words have been reproduced and generalized on real robots: Manschitz et al. [Man+18] learn to generalize a handwriting skill to unseen locations of a whiteboard which is defined as the target writing position. Berio et al. [BCL16] learn to dynamically draw graffiti tags. In comparison to the above mentioned behavior, these drawings particularly require fluid and rapid manipulation of the pen to produce elegant and smooth sequences of letters. Precision is less important for this behavior.

A.1.3. Movable Objects (C)

Grasping: Grasping is a good example for a high diversity of similar but different task formulations. The problem of grasping is tightly coupled with perception, but it can be separated into perception and movement generation. Continuous feedback can be used to verify the grip although it can also be sufficient to perceive the target before the grasp attempt. Problem formulation for grasping varies in the degree of automation and amount of other methods used in the process. Sometimes perception is learned and movement generation is done with other approaches and vice versa. Some approaches learn full reaching and grasping movements for known object locations [GSB10; Kal+11b; Stu+11; Amo+12], others just learn to predict grasp poses [LLS15; JLD16; PG16]. Steil et al. [Ste+04] only consider the problem of defining hand postures and Kroemer et al. [Kro+09] the problem of learning hand poses relative to objects. A full grasping movement includes a reaching trajectory, positioning the gripper at the correct position, closing the gripper, and sometimes objects have to be moved in the right position before the gripper can be closed. From the works that are mentioned here, Gräve et al. [GSB10], Steil et al. [Ste+04], and Stulp et al. [Stu+11] do not learn to use feedback from sensors,

Kroemer et al. [Kro+09] use features obtained from images, Kalakrishnan et al. [Kal+11b] use force measurements, Lenz et al. [LLS15] use RGB-D images, Johns et al. [JLD16] and Mahler et al. [Mah+17] use depth images, and Lampe and Riedmiller [LR13], Pinto and Gupta [PG16], and Levine et al. [Lev+18] use RGB images. Figure A.1 illustrates possible inputs and outputs of a component that generates grasping behavior. A classification proposed by Bohg et al. [Boh+14] distinguishes between grasping of known, familiar, and unknown objects. Familiar means that the robot did not encounter the objects before, but has seen similar objects. Most of the works that we present here fall into this category. For grasping, other factors that influence the difficulty of the problem are the used hand or gripper and the objects that should be grasped. Promising results are shown by Levine et al. [Lev+18] and Mahler et al. [Mah+17]. A large variety of different objects can be grasped with a two-finger gripper just based on images or depth images respectively. However, there are still many options for improvements. The gripper can only grasp objects with top-down movements. In the real world, not all problems can be solved with this kind of grasp. The gripper only has two fingers. Hands with more fingers have better control over grasped objects. Using force feedback and tactile sensors would improve grasping in some situations. In a box full of objects, the approach of Levine et al. [Lev+18] just picks a random object. In practice, this should be a parameter of the behavior. Also, it is not clear where and in which orientation the gripper holds the object. This does not seem to be a problem because most works just consider the grasping phase but not what happens afterwards. In a real application, most probably the object will have to be placed in some other location. Since the grasping is not as accurate as one would expect in many cases, knowing the orientation of the object inside the gripper is a useful information to prepare the placing behavior. This can be done either by in-hand manipulation, which requires more fingers, or by adjusting the final target position of the arm taking into consideration the object's orientation.

Pick and place: A skill that is similar to grasping is pick and place. Some works assume that picking the object is already solved and learn only object placement [Ijs+13; Fin+17a], others learn both pick and place in one policy [STS12; Rah+18; Che+17b]. Some works only focus on movement execution [Ijs+13], others generalize from object features to trajectories [KS17], or even learn camera-based perception and action end to end for one specific object [Fin+17a; Che+17b]. An interesting work from Stulp et al. [STS12] considers the special case of this problem under uncertainty. It assumes a state estimation approach to track the object's location which does not yield perfect results. In addition, a sequence of movements is learned. A variant of pick and place is placing coat hanger on a rack. Levine et al. [Lev+16] learned to perform this task end to end from camera images to motor torques.

The next level of difficulty for simple pick and place tasks is placing objects precisely, for example, stacking boxes. An interesting work shows that this can be learned even with a low cost manipulator that has play in its joints and a wobbling base [DFR15]. While this can be easily interpreted as noise from a machine learning perspective, other methods often fail without any informative prior knowledge. In their study, perception

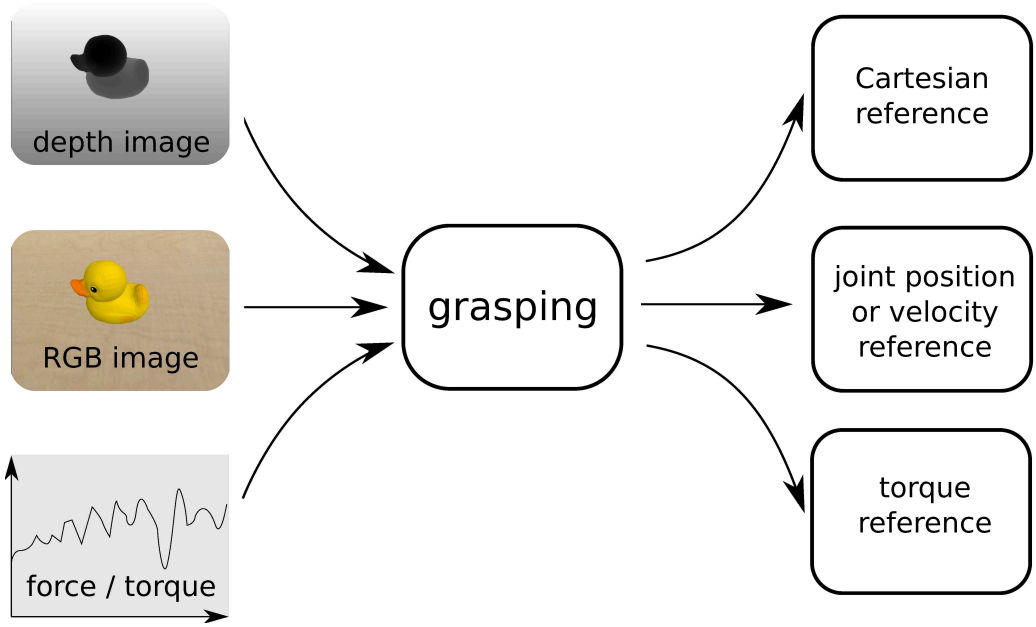


Figure A.1.: Learning grasping from sensory information. Exemplary sensor data that could be used to generate grasping behaviors and possible outputs of a skill.

has not been learned but continuous feedback from a vision system has been used to generate appropriate action. Duan et al. [Dua+17] tackle a more difficult problem by learning a direct mapping from visual input to actions. In their work, however, a more precise robotic system has been used.

In-hand manipulation: As objects cannot always be picked up in a specific configuration, in-hand manipulation may be necessary to reposition the objects within a robot's hand. In general, this is a dexterous manipulation skill that requires a gripper with multiple fingers that can be driven individually. Hoof et al. [Hoo+15] learn robot in-hand manipulation with unknown objects by using a passively compliant hand with two fingers and exploiting tactile feedback. They investigate an in-hand object rolling task and learn a control policy that generalizes to novel rollable cylindrical objects that differ in diameter, surface texture and weight. In their work, dynamics and kinematics of the compliant robot hand are unknown to the learning algorithm.

The hand used by Rajeswaran et al. [Raj+17a] has five fingers and has pneumatic actuation. They consider the problem of learning in-hand rotation of elongated objects with and without the use of a wrist joint under varying initial conditions. The object can either be in the hand at the start of the behavior or picked up and moved to the desired configuration. Learning this skill is shown to be possible with only proprioceptive feedback. This includes pressure measurements, positions, and velocities of each joint.

OpenAI et al. [Ope+20] learn a complex in-hand manipulation skill: changing the orientation of a cube to any desired orientation in a robotic hand with five fingers. Two components are learned: a vision component that computes the object's pose from three camera images from significantly different, fixed perspectives and a policy component that uses the finger tip positions and the object pose to generate motion commands for the fingers. The finger tip positions are measured with a motion capture system which unfortunately makes the learned skill in its current form not suitable for a humanoid robot outside of the lab.

Tumbling / tilting an object: The challenge in quasi-static manipulation tasks like tumbling or tilting objects from one face to another is to control the position of the respective object over a period of time. Pollard and Hodgins [PH04] generalize an object-tumbling skill to novel object sizes, shapes and friction coefficients. Kroemer and Sukhatme [KS17] further enhance the difficulty by learning to tilt objects exactly around their defined pivotal corners. This task requires a high accuracy during the whole skill execution because the object's corner has to stay continuously in contact with the desired pivot point.

A.1.4. Deformable Objects (D)

Knot tying and untying: Tying a knot is a behavior that is frequently required, for instance, during surgical operations, in the household domain, for search and rescue, or for sailing where threads or ropes are often used. Berg et al. [Ber+10] demonstrate that a combination of behavior learning and optimal control can be used to learn fast and smooth knot tying with two manipulators consisting of 14 motors. This would be a particularly challenging task for planning algorithms that would have to reason about a three-dimensional soft body.

Similarly, untangling ropes and untying knots is required in the same domains as well as for technical applications in which cables unintentionally tangle up. Wen Hao Lui and Saxena [WS13] learn to predict the rope configuration and use it to choose several actions from a predefined set to untangle the rope.

Handling Garments: Corona et al. [Cor+18] learn to handle garment, that is, arranging garment from an unknown configuration to a reference configuration from which further steps can be executed, for example, folding it or dressing a person. The difficult part is the prediction of suitable grasp points from camera images. A bimanual setup has been used: one arm grasps a garment and presents it to an RGB-D camera, the garment is recognized, and two grasping points for the arms are identified to bring the garment to a reference configuration. Jeans, T-shirts, jumpers, and towels can be handled by the system.

Colomé and Torras [CT18] learn to fold a polo shirt with two robotic arms. Each arm has 7 DOF. Only trajectories for two arms are learned. An accurate model of the polo shirt and its interaction with the grippers of the arms is not available. The

learned trajectories minimize wrinkles in the shirt and make it look as close to a reference rectangle as possible.

Erickson et al. [Eri+18] consider the problem of robot-assisted dressing: while a human is holding his arm up and holds his posture strictly, a PR2 robot pulls a hospital gown onto the arm of human. Physical implications of actions on people are learned from simulation. The learned model predicts forces on a person's body from the kinematic configuration and executed actions. The model is combined with model predictive control to solve the task. Hence, neither action, nor perception are learned completely.

A.1.5. Divisible Objects (E)

Cutting: Cutting objects is a complex task as dynamics are induced during the process of object cutting. Cutting tasks can be found in various domains. For example, Lioutikov et al. [Lio+16] consider the task of cutting vegetables in a kitchen scenario. In their work, the movement is composed of multiple steps that are executed autonomously in a sequence. The learned behavior generalizes to changed cutting positions. However, they do neither consider the required forces to cut the objects nor the involved dynamics. As a result, the cutting motion has to be executed multiple times to finally slice the vegetable. Therefore, while Lioutikov et al. [Lio+16] represent cutting motions as discrete behaviors, they recommend to represent them as rhythmic behaviors in future work. The difficulty of food-cutting tasks is further exacerbated, if vegetables with different stiffness and shape are evaluated. In this case, the (non-linear) dynamics vary not only with time but also with different object types. As the hand-designing of such dynamics models is infeasible, Lenz et al. [LKS15] aim to learn the prediction of these dynamics and the respective controllers directly from a dataset of about 1500 cuts. In the medical field, Thananjeyan et al. [Tha+17] investigate surgical pattern cutting of deformable tissue phantoms in the context of laparoscopic surgery. As the task requires simultaneous tensioning and cutting, they learn a tensioning policy which depends on the specific cutting trajectory and maps the current state of the gauze to output a direction of pulling. Similar to the work from Lenz et al. [LKS15], the dynamical deformation is difficult to observe or to model analytically. Therefore, they directly learn the cutting policy in an end-to-end fashion.

A similar task is peeling which has been learned by Medina and Billard [MB17]. It is, however, modeled as a sequence of reaching, peeling and retracting. Only with one arm the peeling motion for a zucchini has been learned while another arm holds it.

A.1.6. Movable Objects, Dynamic Behavior (F)

Batting, throwing and kicking: For many games some sort of batting or throwing behavior is required, for example, hockey [Dan+13; Che+17a; RK17; Par+18], golf [Mae+16], minigolf [KKB12], billiard [AMS97; Pas+11], baseball [PVS05; PS08b], badminton [Liu+13], tennis [INS02], table tennis [Kob+10; MKP11; Kob+12; Mül+13], tetherball [DNP12b; Par+15], darts [Kob+12], throwing [Gam+10; Ude+10; Kob+12; da +14; Gut+18], and kicking [BL17; HQS10; Asa+96]. These are dynamic manipulation

behaviors because momentum from the end effector has to be transferred to the manipulated object. We can distinguish between settings where a specific goal has to be reached by hitting or throwing an object directly [Che+17a; KKB12; RK17; Par+18; Gam+10; Ude+10; da +14; Gut+18] or indirectly [Dan+13; AMS97], or the distance or velocity has to be maximized [Pas+11; PVS05; PS08b]. Sometimes performing the motion was enough [Mae+16; Liu+13; INS02; DNP12b; BL17]. Winning the game was the goal in the case of tetherball [Par+15], or scoring a goal in the case of soccer [HQS10; Asa+96]. An extension to the problem of hitting a specific goal is to hit a given goal from a target space, for example, along a line [KKB12], from an area [Kob+12; Gam+10; Ude+10; da +14; RK17; Gut+18], or from a discrete set of targets [Kob+12]. In some cases specialized machines have been used, for example, Atkeson et al. [AMS97] use a simple billiard robot or Liu et al. [Liu+13] use a badminton robot with three DOF. In contrast, Pastor et al. [Pas+11] use a humanoid robot to play billiard or Mülling et al. [Mül+13] use robotic arms to play table tennis. In some works, only serve motions [Liu+13] or hitting static objects [PVS05; HQS10] are learned, in other works a moving object has to be hit [Mül+13; Par+15]. Perception and state estimation is not learned in any of the presented works, hence, behaviors that rely on perception and state estimation of moving targets [Par+15; Mül+13] can be considered as deliberative. Most of these problems, however, have been solved without exterioceptive sensors. Kicking a ball with a legged humanoid represents a particular challenge because the robot has to keep balance. Böckmann and Laue [BL17] execute a learned kick with manually implemented balancing and Hester et al. [HQS10] learn to perform a kick that avoids falling over while scoring a goal. State estimation uncertainty and noise is an issue if perception is involved in the skill although this has not been mentioned explicitly in the works of Parisi et al. [Par+15] and Mülling et al. [Mül+13] in which state estimation methods have been used. Hence, we assume this has not been considered to be a significant problem. Learning the perception part of these behaviors has not been considered so far and would significantly increase the difficulty of the problems.

More dynamic manipulation behaviors: In ball-in-a-cup, a ball is attached to a cup by a string. The goal is to move the cup to catch the ball with it. A robot has to swing the ball up and catch it. The movements of the ball are sensitive to small perturbations of the initial conditions or the trajectory of the end effector [KMP08]. Successful behaviors are learned so that they take into account the ball position [KMP08; KP09] to compensate for perturbations, however, the perception part is not learned in any of these works. Kober et al. [KMP08] state that it is a hard motor learning task for children.

Another remarkable work is published by Kormushev et al. [KCC10b]. The goal is to flip a pancake with a frying pan. It is a dynamic task and the pancake is susceptible to the influence of air flow which makes it hard to predict its trajectory.

Zhao et al. [Zha+18b] learn nunchaku flipping, which is a dynamic behavior. A nunchaku is a weapon that consists of two sticks that are connected by a chain. A hand with haptic sensors and five fingers has been used. Zhao et al. [Zha+18b] emphasize that the

task requires compound actions that have to be timed well, contact-rich interaction with the manipulated object, and handling an object with multiple parts of different materials and rigidities.

Balancing: A typical balancing example which is often used as a sample problem is balancing an inverted pendulum. Marco et al. [Mar+16] and Doerr et al. [Doe+17] investigate this problem in a real-world manipulation scenario by utilizing a robotic arm with seven DOF to balance an inverted pendulum. In their work, they learn parameterizations of a PID controller or a linear-quadratic regulator (LQR), respectively, while a motion capture system is used to track the angle of the balanced pole.

A.1.7. Granular Media and Fluids (G)

Scooping: For humans, reasoning about fluids and granular media is no more difficult than reasoning about rigid bodies. Not many researchers try to tackle these problems with robots. Schenck et al. [Sch+17a] learn scoop and dump actions of granular media. Both are executed in sequence and they are encoded with nine parameters that tell the robot where and how to scoop and where to dump the granular media. The problem that is solved is to scoop pinto beans from one tray and dump it to another tray to create a desired shape in the target tray. A Gaussian-shaped pile and the letter G have been selected as target shapes. The robot was allowed to execute 100 scoop and dump actions. A depth camera is used to measure the current state of the granular media. The part of the behavior that has been learned is a model that predicts the effect of actions which will then be used to select good actions.

Pouring: An application which requires (weak) dynamical movements with moderate precision is pouring liquids from a bottle into a cup. Learning focuses on the generalization of the movement to new goals (position of the cup [PHS08]), changed initial positions (position of the bottle [Chi+17]), or different object shapes and sizes [BKP14; Tam+11]. Tamosiunaite et al. [Tam+11] learn both, the shape of the trajectory and the goal position to generalize a trajectory to a different bottle. Similar to the pick-and-place applications detailed above, the elementary pouring problem can also be extended to a pick-and-pour task [Cac+18; Chi+17]. In contrast to the above mentioned works which acquire the pouring trajectories from human demonstrations, robotic pouring behaviors can also be learned in an end-to-end fashion directly from videos [Ser+18].

A.1.8. Collision Avoidance (H)

Robotic manipulation behaviors can result in collisions with the robot's own body, other agents or the environment. The latter is often termed obstacle avoidance, where the obstacles can be both static or dynamic. While static objects in the environment can be modeled well within a world model, dynamic obstacles are often circumnavigated with reactive behaviors. Both, collision and obstacle avoidance are important in real-world manipulation scenarios. Koert et al. [Koe+16] learn adaptation of trajectories in case of

unforeseen static obstacles represented by a point cloud that has been obtained from a depth camera.

A.1.9. Miscellaneous (I)

There are also some more unusual behaviors that have been learned but we will not discuss them in detail. Among these are archery [Kor+10], which is similar to throwing a ball or darts but does not involve an accelerating trajectory, playing with the Astrojax toy [Par+18], playing maracas [Par+18], drumming [Ude+10], and calligraphy [OPL15].

A.2. Locomotion Behaviors

A.2.1. Walking (A)

The prime example of the category locomotion is walking. Walking is a diverse robotic behavior learning problem. Its diversity stems on the one hand from the variety of different walking machines: six-legged [MB90; Kir97], quadrupedal [KS04; KAN08; BLE07; KN09; Kal+09; Zuc+11; Bar+16], or biped systems [BF97; Mat+05; GPW06; Kor+11b; MB15] have been considered for this paper. On the other hand, the problem formulation can be made more difficult by requiring the system to walk up stairs [KN09] or walk on irregular or rough terrain [KAN08; Kal+09; Zuc+11]. In principle, the problems of walking as fast [KS04], straight [BLE07], energy-efficient [Kor+11b], or stable [MB15] as possible can be distinguished. While six-legged and quadrupedal systems are stable enough to prevent falling over in most situations and, hence, qualify for static behaviors, bipedal systems are often unstable and it is a hard problem to prevent them from falling over. Hence, bipedal walking can be considered a dynamic learning problem. Walking is a rhythmic and active behavior. It is an elementary skill that can be used in many application domains, however, walking robots are in competition to wheeled robots which are much more energy-efficient and precise in flat terrain. While walking itself is a rhythmic behavior, precise foot placement is a discrete behavior. Precise foot placement is required for climbing stairs [KN09] and walking on rough terrain [KAN08; Kal+09; Zuc+11] on a lower level of behavior abstraction (see Figure 2.4). Those behaviors also combine learning methods with other planning and control methods. Bipedal robots are leaner than other walking machines and they are able to move like humans and in the same environment, for example, go through narrow paths [BF97]. Because bipedal walking is not statically stable per se, controllers have to compensate disturbances continuously. Either static stability or dynamic stability can be the goal of a bipedal walk. Often the problem of learning bipedal walking is restricted by supporting structures to the sagittal plane to simplify the balancing problem [BF97; Mat+05; GPW06] but not always [Kor+11b; MB15]. However, behaviors are often prestructured to restrict and, hence, simplify the learning problem. For example, Missura and Behnke [MB15] only learn the balancing part of the walk. Using sensory feedback is particularly important for bipedal walking. Apart from proprioceptive sensors [Mat+05], ground contact sensors

have been used [GPW06]. Robustness to slightly irregular surfaces and changes of the robots dynamics have also been considered [Mat+05] for bipedal walking.

A more difficult version of bipedal walking is riding a pedal racer. In principle, it is comparable but it is crucial to exert a controlled force on the pedals. Hence, Gams et al. [Gam+14] use a 6-DOF force-torque sensor in each foot of the bipedal robot to generate feedback to the learned behavior.

A.2.2. Dribbling (B)

Walking or running while controlling a ball is called dribbling. It can be used, for example, in basketball, handball, or soccer. Latzke et al. [LBB07] learned dribbling for soccer with a humanoid toy robot by walking against the ball. The walking behavior is simple because it only uses three motors. The goal is to learn how to score a goal with dribbling, starting from ten different initial ball positions at the middle of the field. Only high-level control, that is, setting a walking direction has been learned. Positions of the ball and the goal are obtained from a world model.

A.2.3. Jumping (C)

If the walking robot is too small and the terrain too rough, jumping is sometimes necessary. Kolter and Ng [KN09] show that this can be used for climbing up large stairs with a small quadrupedal robot. With the same robot, Theodorou et al. [TBS10b] learn to jump across a gap by maximizing the distance of the jump while jumping straight to prevent falling over. Unfortunately, Theodorou et al. [TBS10b] could not evaluate their approach on the real system.

A.2.4. Standing Up (D)

A stand-up behavior is important for any biped robot acting in the real world. In general, the difficulty is that there exists no static solution, as there is no joint linking the robot to the ground. For many robots, a robot-specific, preprogrammed stand-up movement is used instead of acquiring the skill by learning. However, Morimoto and Doya [MD01] learn a dynamical stand-up policy both in simulation and on a real two joint robot. The robot (incrementally) learns a skill to stand up dynamically by utilizing the momentum of its body mass. An inclination sensor measures the current state of the system and motor torques are produced by the learned motor skill. The hierarchical learning architecture learns to generate postures by means of an upper level policy and the movements to achieve the next posture (sub-goals) by means of a lower level policy.

A.2.5. Balancing (E)

Keeping balance is a fundamental locomotion requirement and has been achieved with various approaches by modifying different aspects of the motion. For example, balancing a walking humanoid by modifying the gait [MB15], using arm motions [KGB11] or control motor torques [Vla+09] to balance a robot on two wheels. Often behavior learning is

combined with classical control approaches: Kuindersma et al. [KGB11] use an existing balance controller for normal balancing and only activate arm motions for postural recovery when the inertial measurement unit (IMU) detects perturbations through impacts of an external weight.

A.2.6. Collision Avoidance (F)

Learning collision avoidance seems to play a secondary role in manipulation (see paragraph *Manipulation: Collision Avoidance*). There are, however, many works in the context of locomotion, where it is mainly related to navigation problems. The publications discussed in this paragraph directly use images and vision systems. They present learned reactive collision avoidance behaviors. In the field of navigation, Tai et al. [TLL16] learn a collision avoidance strategy based on depth images in an indoor obstacle avoidance scenario. They use a mobile, wheeled robot that learns to move in corridors with a set of discrete actions. However, the robot only encounters static obstacles. Loquercio et al. [Loq+18] investigate a civilian drone flight application. In their work, the drone learns to safely fly in the streets of a city by mapping each single input image directly to a drone steering angle and a collision probability to react to unforeseen obstacles. The behavior for navigation and obstacle avoidance is trained for urban environments from the viewpoint of bicycles and cars but can be generalized to novel situations like indoor environments or high altitudes without retraining. The outputs of the perception model are not directly used to control the drone but converted to movement commands with fixed rules. Similarly, Gandhi et al. [GPG17] also learn to navigate an unmanned aerial vehicle while avoiding obstacles. They use negative experiences, that is, a visual dataset of more than 11,500 crashes in various environments with random objects, in conjunction with positive data to learn to fly even in cluttered, dynamic indoor environments. The behavior is learned end to end by taking camera images and outputting probabilities of the motion commands go *left*, *right*, or *straight*. Kahn et al. [Kah+17] learn uncertainty-aware collision avoidance, that is, given a camera image and a sequence of controls the learned model will output a collision probability together with an estimate of uncertainty. The approach proceeds cautiously in unfamiliar environments and increases velocity in areas of higher confidence. Model predictive control is used to generate actions, while the cost model incorporates collision probability and uncertainty. The approach has been tested with a quadrotor and an RC car.

A.2.7. Navigation (G)

Assuming the robotic system knows how to walk or drive, where should it move? High-level locomotion behaviors like navigation and exploration are concerned with local direction generation, for example, navigation through complex natural environments [SBS10], navigation to visually presented targets [Zhu+17], navigation to targets with known relative location [Pfe+17], lane following [Chu+18b], reducing state estimation uncertainty in navigation [OHB10] and navigating to a target position [CP07]. Most of the works discussed here are concerned with wheeled robots but are in principle transferable to

walking robots. Classical navigation through natural terrain has been considered by Silver et al. [SBS10]. They use planning to generate driving directions but the generation of cost maps for the planner are learned. The cost maps are generated based on perceptual data: static data sources like satellite images or onboard sensors like cameras and Light Detection and Ranging (LiDAR). Zhu et al. [Zhu+17] consider the problem of visual navigation: actions in a 3D environment are predicted based on the current image from the robot's camera and an image of the target. The predicted actions result in a minimum path length to reach the goal. They show that navigation to different targets in a scene can be learned without retraining. The approach has been tested on a wheeled robot in an office environment. Pfeiffer et al. [Pfe+17] learn navigation to a given relative target location end to end from 2D-laser range findings without a map. Steering commands are directly generated by the learned behavior. The goal was to navigate safely through obstacle-cluttered environments with a mobile platform. A similar problem is to learn lane following from camera images end to end. This has been done by Chuang et al. [Chu+18b]. Oßwald et al. [OHB10] consider the problem of navigation with a humanoid robot that has noisy actuators and sensors. Motion commands are executed more inaccurately with walking robots compared to wheeled robots and camera images are affected by motion blur. A navigation behavior has to trade off quality of pose estimation and walking speed. A vision-based pose estimation has been used and navigation actions (forward, rotate left / right, stand still) for the robot have been learned and take into consideration distance and angle to the goal and pose uncertainty. The goal is to reach the destination reliably as fast as possible. Conn and Peters [CP07] solve a classical grid-world navigation problem in the real world. The laser scan data and orientation information are used by the behavior to generate one of the commands stop, turn left, turn right, or move forward.

As a side note, we would like to mention here that autonomous driving behaviors for cars also fall into the category of navigation. These behaviors can also be learned as shown by Chen et al. [Che+15a] and Bojarski et al. [Boj+16]. Because this topic is broad and it is not of utmost importance for humanoid robots, we will not further investigate it here. The behaviors are often specific for the domain, for example, Bojarski et al. [Boj+16] present an approach to learn lane and road following and Chen et al. [Che+15a] learn driving in a car racing game.

A.2.8. Exploration (H)

Exploration behaviors use (lower level) locomotion behaviors to gain knowledge on the robot's environment. Cocora et al. [Coc+06] successfully transfer exploration behavior from other environments to a new environment to find the entrance of an office. The general problem that they try to solve is navigating to a room with an unknown location. While searching for it, only labels for neighboring rooms are provided to the robot. The required exploration behavior is achieved by learning an abstract navigation policy choosing actions based on the provided local knowledge. Kollar and Roy [KR08] learn an exploration behavior for an unknown environment to maximize the accuracy of a map that is built with simultaneous localization and mapping (SLAM).

A special case of exploration behaviors are sampling routines aimed at acquiring relevant sensory input often referred to as active sensing or active perception. Chen et al. [CLK11] state that "active perception mostly encourages the idea of moving a sensor to constrain interpretation of its environment". For example, a camera has a limited field of view, thus, the goal of an active sensing behavior is to move the part of the robot to which the camera is attached (or the whole robot) to reduce uncertainty about the scene.

Kwok and Fox [KF04] demonstrate how active sensing can be learned in the domain of robotic soccer: a quadrupedal robot has to determine its own location, the location of the ball, and the location of opponents on a soccer field with a camera to finally score a goal. The behavior considers the current estimate of the world state and its uncertainty from the state estimation component. It generates head motions to change the camera position. The robot is trained to score a goal. The active sensing behavior is executed while the normal soccer behavior is running.

A.3. Other Behaviors

Some behaviors cannot generally or not at all be classified as locomotion or manipulation. We will discuss these behaviors in this section.

A.3.1. Human-robot Interaction

Human-robot interaction has become a feasible application through safe, compliant robot control and design. Robots can come into physical contact with humans in these scenarios. Robots that assist humans with their tasks are particularly appealing in the household and manufacturing domains. They can hold objects for a human [Ewe+15], hand over objects to a human [Ewe+15; Mae+17], assist a human in putting on a shoe [Can+18], lift [Evr+09] or carry objects in collaboration with a human [Ber+12; Roz+15], or drill screws placed by a human [Nik+13], hence, show collaborative behavior. They can even interact socially with humans, for example, by giving a high five [Amo+14] or shaking hands [Hua+18]. These behaviors are dynamic because they have to be synchronized with the human. Challenging tasks are the recognition of the human's intention and acting accordingly. Some authors focus on the intention recognition: Amor et al. [Amo+14] only consider the problem of recognizing one interaction scenario by observing the human's motion, whereas Ewerton et al. [Ewe+15] and Maeda et al. [Mae+17] consider the problem of distinguishing between several possible interaction scenarios. In these works, only marker-based motion capture systems have been used to provide motion data from the human counterpart. The presented behaviors are active, discrete manipulation behaviors and perception has not been considered. What makes carrying special is that it is a collaborative behavior which requires continuous observation of the co-worker's state and intention because both agents are indirectly physically connected during the whole activity. Carrying an object in collaboration of a robotic arm and a human might require exerting a specific force on the object, and therefore, a method to measure the forces. Rozo et al. [Roz+15] use a 6-axis force/torque sensor for this. In their application, the

object can only be carried if both agents apply a force in opposite directions. In contrast, Berger et al. [Ber+12] consider collaborative carrying as a whole body problem with a humanoid. They adapt the walking direction of a robot according the movement of its human counterpart. Deviations from learned expected movements are recognized and the motion is adjusted accordingly. In this case only part of the perception is learned. Carrying behavior is often done with the robot following the human leader. They can be considered passive. The similar problem of lifting an object in collaboration has been considered by Evrard et al. [Evr+09]. They additionally learn to recognize if the robot should take the leader or follower role during task execution. Hence, the learned behavior can be both active or passive. Canal et al. [Can+18] provide an example of a deliberative system, where low-level actions have been learned and high-level symbolic planning is used to organize communication and interaction with a human. They study the application of assisting a human in putting on a shoe. The social acceptance of robots is an important aspect for future robots interacting with humans. One of the key factors in this context are natural motions, that is, the robot should not only reach a certain pose of the end effector but also execute the motion in a human-like manner. To achieve this, Huang et al. [Hua+18] present a hybrid space learning approach that learns and adapts robot trajectories in Cartesian and joint space simultaneously while taking into account various constraints in both spaces. They evaluate their approach on a humanoid robot in a hand-shaking task, consisting of a discrete reaching and a rhythmic waving motion, and adapt the movement to different areas for shaking hands. Nikolaidis et al. [Nik+13] present results in a simplified human-robot collaboration scenario. The scenario should model the human-robot interaction challenges that occur in a hybrid team of a human and a robot that has to drill screws. The human has to place screws and the robot drills them. Although in the real world scenario there are no real screws and not a real drill, the robot learns to execute its motions in an order favored by the human. The problem of perceiving the human's current state is simplified by using a motion capture system.

A.3.2. Behavior Sequences

The specific task of unscrewing a light bulb is a good example for sequential tasks that need to be decomposed into smaller subtasks to achieve the overall objective. Manschitz et al. [Man+16] infer an unknown number of such subtasks automatically from demonstrations of the overall task and learn how to sequence the subtasks in order to reproduce the complete task. In their work, the taught task sequence consists of approaching the light bulb, closing the end effector, unscrewing the bulb by rotating the wrist stepwise (after each turning, the fingers are opened and the wrist is rotated back), pulling the light bulb out of the holder and finally putting it into a box.

Besides the applications of pouring, cutting and wiping, another typical kitchen task is cooking (see also pancake flipping described in paragraph *More dynamic manipulation behaviors)* or, more specifically, food preparation. The preparation of food requires structured behaviors with a fixed chronological order of actions. Therefore, the complete task has to be segmented into smaller subtasks. The order of these subtasks is managed by a higher-level monitoring system. Caccavale et al. [Cac+18] picked the tasks of coffee and

tea preparation to present their work on learning the execution of structured cooperative tasks from human demonstrations (respectively, though only in simulation, Caccavale et al. [Cac+17] investigated pizza preparation). A similar approach was presented by Figueroa et al. [FUB16] on pizza dough rolling task with the goal to achieve a desired size and shape of the pizza dough.

A.3.3. Soccer Skills

Soccer is one of the most extensively studied games in robotics. Besides walking, dribbling and kicking, more high-level skills have been learned with simpler robotic systems or in simulation. For example, Müller et al. [Mül+07] learn ball interception on a wheeled robot with known poses and velocities of the ball and the robot, Riedmiller et al. [Rie+09] learn an aggressive defense behavior also based on these information and the pose and velocity of the opponent but only in simulation, Riedmiller and Gabel [RG07] learn cooperative team behavior also in simulation. Another example of a low-level behavior that has been learned for robotic soccer is capturing a ball with the chin of a dog-like robot [FS04].

A.3.4. Adaptation to Defects

A kind of learned behavior that does not fit into any category because it is more general and can be used in combination with any underlying behavior is presented by Cully et al. [Cul+15]. The robot learned to adapt to defects. A walking behavior of a six-legged robot as well as pick and place with a manipulator with redundant joints have been considered.

A.4. Table

Table A.1.: Overview of learned behaviors.

Behavior	Publication	Perception †	Action †	Deliberative ‡	Reactive ‡	Discrete	Rhythmic	Static	Dynamic	Active	Passive	Locomotion	Manipulation
flipping a light switch	[Buc+11]	✗	✓	✓	✗	✓	✗	✓	✗	✓	✗	✗	✓
open door						✓	✗	✓	✗	✓	✗	✗	✓
	[Kal+11b]	✓	✓	✗	✓	·	·	·	·	·	·	·	·
	[Gu+17]	✓	✓	✗	✓	·	·	·	·	·	·	·	·
	[KCC10a; KCC11]	✓	✓	✗	✓	·	·	·	·	·	·	·	·
	[NŽU17]	✗	✓	✓	✗	·	·	·	·	·	·	·	·
	[Che+17b]	✓	✓	✗	✓	·	·	·	·	·	·	·	·
	[ET18]	✗	✓	✗	✗	·	·	·	·	·	·	·	·
valve turning	[Car+12]	✗	✓	✓	✗	✓	✗	✓	✗	✓	✗	✗	✓
crank-turning	[Pet+14]	✓	✓	✗	✓	✗	✓	✓	✗	✓	✗	✗	✓
screw cap on bottle	[Lev+16]	✓	✓	✗	✓	✓	✗	✓	✗	✓	✗	✗	✓
peg-in-a-hole						✓	✗	✓	✗	✓	✗	✗	✓
	[GFB94]	✓	✓	✗	✓	·	·	·	·	·	·	·	·
	[Ell+12]	✗	✓	✓	✗	·	·	·	·	·	·	·	·
	[Lev+16]	✓	✓	✗	✓	·	·	·	·	·	·	·	·
	[Kra+16]	✓	✓	✓	✗	·	·	·	·	·	·	·	·
⊢ power plug	[Che+17a]	✗	✓	✗	✗	·	·	·	·	·	·	·	·
⊢ connect a pipe	[Lau+18]	✗	✓	✓	✗	·	·	·	·	·	·	·	·
ironing	[KCC10a; KCC11]	✓	✓	✗	✓	✓	✗	✓	✗	✓	✗	✗	✓
whiteboard cleaning	[Kor+11a]	✓	✓	✗	✓	✓	✗	✓	✗	✓	✗	✗	✓
grinding / polishing	[Nem+18]	✓	✓	✗	✓	✓	✗	✓	✗	✓	✗	✗	✓
wiping													
	[UAS04]	✗	✓	✗	✗	✗	✓	✓	✗	✓	✗	✗	✓
	[Gam+14]	✗	✓	✗	✓	✗	✓	✓	✗	✓	✗	✗	✓
sweeping													
	[ACC14]	✗	✓	✓	✗	✓	✗	✗	✓	✓	✗	✗	✓
	[PML17]	✓	✗	✗	✓	✓	✗	✗	✓	✓	✗	✗	✓
handwriting										✓	✗	✗	✓
	[Man+18]	✗	✓	✗	✗	✓	✗	✓	✗	·	·	·	·
	[BCL16]	✗	✓	✗	✗	✓	✗	✗	✓	·	·	·	·
calligraphy	[OPL15]	✓	✓	✗	✗	✓	✗	✗	✓	✓	✗	✗	✓
grasping						✓	✗	✓	✗	✓	✗	✗	✓
	[Ste+04]	✗	✓	✗	✗	·	·	·	·	·	·	·	·
	[Kro+09]	✗	✓	✗	✓	·	·	·	·	·	·	·	·
	[GSB10]	✗	✓	✗	✗	·	·	·	·	·	·	·	·
	[Stu+11]	✗	✓	✗	✗	·	·	·	·	·	·	·	·
	[Kal+11b]	✓	✓	✗	✓	·	·	·	·	·	·	·	·
	[Amo+12]	✗	✓	✓	✗	·	·	·	·	·	·	·	·

Table A.1.: Overview of learned behaviors *(continued)*.

Behavior	**Publication**	**Perception †**	**Action †**	**Deliberative ‡**	**Reactive ‡**	**Discrete**	**Rhythmic**	**Static**	**Dynamic**	**Active**	**Passive**	**Locomotion**	**Manipulation**
	[LR13]	✓	✓	✗	✓	·	·	·	·	·	·	·	·
	[LLS15]	✓	✗	✓	✗	·	·	·	·	·	·	·	·
	[PG16]	✓	✗	✓	✗	·	·	·	·	·	·	·	·
	[JLD16]	✓	✗	✓	✗	·	·	·	·	·	·	·	·
	[Lev+18]	✓	✓	✗	✓	·	·	·	·	·	·	·	·
	[Mah+17]	✓	✓	✗	✓	·	·	·	·	·	·	·	·
pick & place						✓	✗	✓	✗	✓	✗	✗	✓
	[STS12]	✗	✓	✗	✗	·	·	·	·	·	·	·	·
	[Ijs+13]	✗	✓	✓	✗	·	·	·	·	·	·	·	·
	[Rah+18]	✗	✓	✓	✗	·	·	·	·	·	·	·	·
	[Che+17b]	✓	✓	✗	✓	·	·	·	·	·	·	·	·
	[KS17]	✗	✓	✓	✗	·	·	·	·	·	·	·	·
	[Lev+16]	✓	✓	✗	✓	·	·	·	·	·	·	·	·
	[Fin+17a]	✓	✓	✗	✓	·	·	·	·	·	·	·	·
block stacking						✓	✗	✓	✗	✓	✗	✗	✓
	[DFR15]	✗	✓	✓	✗	·	·	·	·	·	·	·	·
	[Dua+17]	✓	✓	✗	✓	·	·	·	·	·	·	·	·
in-hand manipulation										✓	✗	✗	✓
	[Hoo+15]	✗	✓	✗	✓	✓	✗	✓	✗	·	·	·	·
	[Raj+17a]	✓	✓	✗	✓	✓	✗	✓	✗	·	·	·	·
	[Ope+20]	✓	✓	✓	✗	✓	✗	✓	✗	·	·	·	·
tumbling / tilting objects													
	[PH04]	✗	✓	✓	✗	✓	✗	✓	✗	✓	✗	✗	✓
	[KS17]	✗	✓	✗	✗	✓	✗	✓	✗	✓	✗	✗	✓
hockey						✓	✗	✗	✓	✓	✗	✗	✓
	[Dan+13]	✗	✓	✗	✗	·	·	·	·	·	·	·	·
	[Che+17a]	✗	✓	✗	✓	·	·	·	·	·	·	·	·
	[RK17]	✗	✓	✗	✗	·	·	·	·	·	·	·	·
	[Par+18]	✗	✓	✗	✗	·	·	·	·	·	·	·	·
knot tying	[Ber+10]	✗	✓	✗	✗	✓	✗	✓	✗	✓	✗	✗	✓
knot untying	[WS13]	✓	✗	✗	✗	✓	✗	✓	✗	✓	✗	✗	✓
folding a shirt	[CT18]	✗	✓	✗	✗	✓	✗	✓	✗	✓	✗	✗	✓
holding garment	[Cor+18]	✓	✗	✓	✗	✓	✗	✓	✗	✓	✗	✗	✓
dressing assistance	[Eri+18]	✗	✗	✓	✗	✓	✗	✓	✗	✓	✗	✗	✓
cutting										✓	✗	✗	✓
	[Lio+16]	✗	✓	✗	✗	✓	✗	✓	✗	·	·	·	·
	[LKS15]	✓	✗	✓	✗	✗	✓	✗	✓	·	·	·	·
	[Tha+17]	✗	✓	✓	✗	✓	✗	✓	✗	·	·	·	·
peeling	[MB17]	✗	✓	✗	✗	✗	✓	✓	✗	✓	✗	✗	✓

Table A.1.: Overview of learned behaviors *(continued)*.

Behavior	**Publication**	**Perception †**	**Action †**	**Deliberative ‡**	**Reactive ‡**	**Discrete**	**Rhythmic**	**Static**	**Dynamic**	**Active**	**Passive**	**Locomotion**	**Manipulation**
scooping	[Sch+17a]	✓	✗	✗	✓	✓	✗	✓	✗	✓	✗	✗	✓
pouring						✓	✗	✗	✓	✓	✗	✗	✓
	[PHS08]	✗	✓	✓	✗	·	·	·	·	·	·	·	·
	[Tam+11]	✗	✓	✗	✗	·	·	·	·	·	·	·	·
	[BKP14]	✓	✓	✗	✗	·	·	·	·	·	·	·	·
	[Chi+17]	✗	✓	✗	✗	·	·	·	·	·	·	·	·
	[Ser+18]	✗	✓	✗	✗	·	·	·	·	·	·	·	·
	[Cac+18]	✗	✓	✓	✗	·	·	·	·	·	·	·	·
collision avoidance	[Koe+16]	✗	✓	✓	✗	✓	✗	✓	✗	✓	✗	✗	✓
golf	[Mae+16]	✗	✓	✗	✗	✓	✗	✗	✓	✓	✗	✗	✓
minigolf	[KKB12]	✗	✓	✓	✗	✓	✗	✗	✓	✓	✗	✗	✓
billiard						✓	✗	✗	✓	✓	✗	✗	✓
	[AMS97]	✗	✓	✓	✗	·	·	·	·	·	·	·	·
	[Pas+11]	✗	✓	✗	✗	·	·	·	·	·	·	·	·
baseball	[PVS05]	✗	✓	✗	✗	✓	✗	✗	✓	✓	✗	✗	✓
badminton	[Liu+13]	✗	✓	✗	✗	✓	✗	✗	✓	✓	✗	✗	✓
tennis	[INS02]	✗	✓	✓	✗	✓	✗	✗	✓	✓	✗	✗	✓
table tennis						✓	✗	✗	✓	✓	✗	✗	✓
	[Kob+10]	✗	✓	✓	✗	·	·	·	·	·	·	·	·
	[MKP11]	✗	✓	✓	✗	·	·	·	·	·	·	·	·
	[Kob+12]	✗	✓	✓	✗	·	·	·	·	·	·	·	·
	[Mül+13]	✗	✓	✓	✗	·	·	·	·	·	·	·	·
tetherball						✓	✗	✗	✓	✓	✗	✗	✓
	[DNP12b]	✗	✓	✗	✗	·	·	·	·	·	·	·	·
	[Par+15]	✗	✓	✓	✗	·	·	·	·	·	·	·	·
darts	[Kob+12]	✗	✓	✗	✓	✓	✗	✗	✓	✓	✗	✗	✓
throwing						✓	✗	✗	✓	✓	✗	✗	✓
	[Gam+10]	✗	✓	✗	✗	·	·	·	·	·	·	·	·
	[Ude+10]	✗	✓	✗	✗	·	·	·	·	·	·	·	·
	[Kob+12]	✗	✓	✗	✗	·	·	·	·	·	·	·	·
	[da +14]	✗	✓	✗	✗	·	·	·	·	·	·	·	·
	[Gut+18]	✗	✓	✗	✗	·	·	·	·	·	·	·	·
kicking						✓	✗	✗	✓	✓	✗	✗	✓
	[BL17]	✗	✓	✓	✗	·	·	·	·	·	·	·	·
	[HQS10]	✗	✓	✓	✗	·	·	·	·	·	·	·	·
	[Asa+96]	✗	✓	✓	✗	·	·	·	·	·	·	·	·
ball-in-a-cup						✓	✗	✗	✓	✓	✗	✗	✓
	[KMP08]	✗	✓	✗	✓	·	·	·	·	·	·	·	·
	[KP09]	✗	✓	✗	✓	·	·	·	·	·	·	·	·
pancake flipping	[KCC10b]	✗	✓	✗	✓	✓	✗	✗	✓	✓	✗	✗	✓

Table A.1.: Overview of learned behaviors *(continued)*.

Behavior	**Publication**	**Perception †**	**Action †**	**Deliberative ‡**	**Reactive ‡**	**Discrete**	**Rhythmic**	**Static**	**Dynamic**	**Active**	**Passive**	**Locomotion**	**Manipulation**
nunchaku flipping	[Zha+18b]	✓	✓	✗	✓	✓	✗	✗	✓	✓	✗	✗	✓
archery	[Kor+10]	✗	✓	✗	✗	✓	✗	✓	✗	✓	✗	✗	✓
astrojax	[Par+18]	✗	✓	✗	✗	✗	✓	✗	✓	✓	✗	✗	✓
maracas	[Par+18]	✗	✓	✗	✗	✗	✓	✗	✓	✓	✗	✗	✓
drumming	[Ude+10]	✗	✓	✗	✗	✗	✓	✗	✓	✓	✗	✗	✓
balancing on wheels	[Vla+09]	✗	✓	✓	✗	✗	✗	✗	✓	✗	✓	✗	✗
postural recovery	[KGB11]	✗	✓	✗	✗	✗	✗	✗	✓	✗	✓	✗	✗
balancing inv. pendulum						✗	✗	✗	✓	✗	✓	✗	✓
	[Mar+16]	✗	✓	✗	✓	·	·	·	·	·	·	·	·
	[Doe+17]	✗	✓	✗	✓	·	·	·	·	·	·	·	·
walking													
⊢ six legs						✗	✓	✓	✗	✓	✗	✓	✗
	[MB90]	✗	✓	✗	✗	·	·	·	·	·	·	·	·
	[Kir97]	✓	✓	✗	✓	·	·	·	·	·	·	·	·
⊢ quadrupedal						✗	✓	✓	✗	✓	✗	✓	✗
	[BLE07]	✗	✓	✓	✗	·	·	·	·	·	·	·	·
	[KS04]	✗	✓	✓	✗	·	·	·	·	·	·	·	·
	[Bar+16]	✗	✓	✗	✓	·	·	·	·	·	·	·	·
⊢ biped						✗	✓	✗	✓	✓	✗	✓	✗
	[BF97]	✗	✓	✗	✓	·	·	·	·	·	·	·	·
	[Mat+05]	✗	✓	✗	✓	·	·	·	·	·	·	·	·
	[GPW06]	✓	✗	✗	✓	·	·	·	·	·	·	·	·
	[Kor+11b]	✗	✓	✓	✗	·	·	·	·	·	·	·	·
	[MB15]	✗	✓	✓	✗	·	·	·	·	·	·	·	·
walking up stairs	[KN09]	✗	✓	✗	✗	✓	✓	✗	✓	✓	✗	✓	✗
walking on rough terrain										✓	✗	✓	✗
	[KAN08]	✓	✗	✓	✗	✓	✓	✓	✗	·	·	·	·
	[Kal+09]	✓	✗	✓	✗	✓	✓	✓	✗	·	·	·	·
	[Zuc+11]	✓	✗	✓	✗	✓	✓	✓	✗	·	·	·	·
pedal racer	[Gam+14]	✓	✓	✗	✓	✗	✓	✗	✓	✓	✗	✓	✗
jumping								✗	✓	✓	✗	✓	✗
	[KN09]	✗	✓	✗	✗	✓	✓	·	·	·	·	·	·
	[TBS10b]	✗	✓	✗	✗	✓	✗	·	·	·	·	·	·
dribbling	[LBB07]	✗	✓	✓	✗	✗	✓	✗	✓	✓	✗	✓	✗
standing up	[MD01]	✓	✓	✗	✓	✓	✗	✗	✓	✓	✗	✓	✗
collision avoidance						✓	✗			✓	✗	✓	✗
	[TLL16]	✓	✓	✗	✓	·	·	✓	✗	·	·	·	·
	[Loq+18]	✓	✗	✗	✓	·	·	✓	✗	·	·	·	·

Table A.1.: Overview of learned behaviors *(continued)*.

Behavior	**Publication**	**Perception †**	**Action †**	**Deliberative ‡**	**Reactive ‡**	**Discrete**	**Rhythmic**	**Static**	**Dynamic**	**Active**	**Passive**	**Locomotion**	**Manipulation**
	[GPG17]	✓	✓	✗	✓	·	·	✓	✗	·	·	·	·
	[Kah+17]	✓	✗	✓	✗	·	·	✗	✓	·	·	·	·
ball interception	[Mül+07]	✗	✓	✓	✗	✗	✗	✗	✓	✓	✗	✗	✗
defense behavior	[Rie+09]	✗	✓	✓	✗	✗	✗	✗	✓	✓	✗	✗	✗
cooperative behavior	[RG07]	✗	✓	✓	✗	✗	✗	✗	✓	✓	✗	✗	✗
capturing a ball	[FS04]	✗	✓	✓	✗	✓	✗	✗	✓	✓	✗	✗	✓
visual navigation	[Zhu+17]	✓	✓	✗	✓	✓	✗	✓	✗	✓	✗	✓	✗
navigation	[SBS10]	✓	✗	✓	✗	✓	✗	✓	✗	✓	✗	✓	✗
navigation	[CP07]	✓	✓	✗	✓	✓	✗	✓	✗	✓	✗	✓	✗
navigation	[Pfe+17]	✓	✓	✓	✓	✓	✗	✓	✗	✓	✗	✓	✗
lane following	[Chu+18b]	✓	✓	✗	✓	✓	✗	✓	✗	✓	✗	✓	✗
navigation and estimation	[OHB10]	✗	✓	✓	✗	✓	✗	✗	✓	✓	✗	✓	✗
navigation with exploration	[Coc+06]	✗	✓	✓	✗	✓	✗	✓	✗	✓	✗	✓	✗
exploration	[KR08]	✗	✓	✓	✗	✓	✗	✓	✗	✓	✗	✓	✗
active sensing	[KF04]	✗	✓	✓	✗	✓	✗	✓	✗	✓	✗	✗	✗
unscrewing a light bulb	[Man+16]	✓	✗	✗	✓	✓	✗	✓	✗	✓	✗	✗	✓
coffee / tea preparation	[Cac+18]	✗	✓	✓	✗	✓	✗	✓	✓	✓	✗	✗	✓
pizza preparation	[Cac+17]	✗	✓	✓	✗	✓	✗	✓	✗	✓	✗	✗	✓
pizza dough rolling	[FUB16]	✗	✓	✗	✓	✓	✗	✓	✗	✓	✗	✗	✓
high five	[Amo+14]	✗	✓	✗	✓	✓	✗	✗	✓	✓	✗	✗	✓
hand shaking	[Hua+18]	✗	✓	✗	✗	✓	✓	✗	✓	✓	✗	✗	✓
hand-over													
	[Ewe+15]	✗	✓	✗	✓	✓	✗	✗	✓	✓	✗	✗	✓
	[Mae+17]	✗	✓	✗	✓	✓	✗	✗	✓	✓	✗	✗	✓
holding	[Ewe+15]	✗	✓	✗	✓	✓	✗	✗	✓	✓	✗	✗	✓
carrying													
	[Roz+15]	✓	✓	✗	✓	✓	✗	✗	✓	✗	✓	✗	✓
	[Ber+12]	✓	✗	✗	✓	✓	✗	✗	✓	✗	✓	✗	✓
lifting													
	[Evr+09]	✗	✓	✗	✓	✓	✗	✗	✓	✓	✓	✗	✓
putting on a shoe	[Can+18]	✗	✓	✓	✗	✓	✗	✗	✓	✓	✗	✗	✓
collaborative drilling	[Nik+13]	✗	✓	✓	✗	✓	✗	✗	✓	✓	✗	✗	✓

Table A.1.: Overview of learned behaviors *(continued)*.

Behavior	Publication	Perception †	Action †	Deliberative ‡	Reactive ‡	Discrete	Rhythmic	Static	Dynamic	Active	Passive	Locomotion	Manipulation

† **Perception and Action:** Refers to the part of the behavior that has been learned.
‡ **Deliberative and Reactive:** Refers to the complete behavior. Behaviors are considered to be deliberative if models of the world or the robot in the world are constructed.
Symbols:

- ⊢ Indicates that the behavior is an instance of the more general behavior above.
- ✓ Behavior has this property.
- ✗ Behavior does not have this property.
  We cannot state that the behavior generally has this property.
- · Property is inherited from the behavior category.

Appendix B.

Other Behavior Learning Algorithms

In this appendix we will discuss several behavior learning algorithms that were not relevant enough for this thesis to be included in Chapter 2.

B.1. Hierarchical Reinforcement Learning

A way of reducing the difficulty of learning complex skills is to break it down into smaller subproblems. Hierarchical RL [BM03] is concerned with this problem. An early work that applies a hierarchical version of Q-learning in robotics has been published by Kirchner [Kir97]. A goal-directed walking behavior for the six-legged walking machine SIR ARTHUR with 16 DOF and four light sensors has been learned. The behavior has been learned on three levels: (i) bottom: elementary swing and stance movements of individual legs are learned first, (ii) middle: these elementary actions are then used and activated in a temporal sequence to perform more complex behaviors like a forward movement of the whole robot, and (iii) top: a goal-achieving behavior in a given environment with external stimuli. The top-level behavior was able to make use of the light sensors to find a source of maximum light intensity. On the lowest level, individual reward functions for lifting up the leg, moving the leg to the ground, stance the leg backward, and swinging the leg forward have been defined.

B.2. Meta Learning

Meta-learning approaches have been proposed by Duan et al. [Dua+17] for block stacking and Finn et al. [Fin+17b] and Yu et al. [Yu+18] for pick-and-place tasks to improve the sample efficiency of imitation learning for manipulation skills with camera inputs. The idea is to transfer knowledge about how to solve a specific task to similar new tasks with a low number of demonstrations.

Also in reinforcement learning additional improvements in sample efficiency can be obtained when we try to solve multiple similar problems with Model-Agnostic Meta Learning (MAML) [FAL17]. It can be combined with REINFORCE and TRPO to solve new reinforcement learning tasks faster after training on a set of similar tasks. Recently, Song et al. [Son+20] presented a MAML variant that does not need policy gradients but uses evolution strategies [Rec71].

B.3. Model-Based Reinforcement Learning

Many RL algorithms do not use a model, that is, they are model-free RL algorithms. This refers to the state transition model and the reward model. Those are assumed to be unknown. If they are known, an RL problem will become an optimal control problem, hence, RL and optimal control have a strong connection [SBW92]. There is a middle ground: we can learn a model from interaction of the agent with its environment. With a state transition model, we can exchange computational effort for sample efficiency. We can alternate between model estimation and planning steps on the model. The Dyna architecture Sutton [Sut90] is the earliest example of this concept and can be used with Q-learning to form the algorithm Dyna-Q. Planning in this case means that we sample state-action pairs to query the model for successor state and reward so that we can update the state-action value function with this virtual experience tuple.

Model-based deep RL with a partially known model has been responsible for a major breakthrough of artificial intelligence in the domain of board games. The game Go has been extremely demanding for traditional search-based approaches. AlphaGo [Sil+16] was the first computer program to beat a professional Go player and later on even the best human Go player. AlphaGo combines planning by Monte Carlo tree search (MCTS) and RL. After imitation learning from human players, REINFORCE policy gradients [Wil92] are used to learn a policy represented by a neural network. Furthermore, a value function network is learned from human games and self-play with the policy network. The learned value function and policy network are then used to guide and evaluate planning steps in MCTS and, thus, reduce computational cost. MCTS is a stochastic planning approach that is often used in the domain of board games. The combined approach runs on a platform with 48 CPUs and 8 GPUs to play a single game. A distributed version uses 1,202 CPUs and 176 GPUs. AlphaGo Zero [Sil+17] extends this work. It is trained without any human knowledge and consistently beats AlphaGo. AlphaGo Zero uses MCTS also during training of the policy and value function networks to obtain better estimates of their objective functions.

Direct policy search can also benefit from a model. Deisenroth and Rasmussen [DR11] present Probabilistic Inference for Learning Control (PILCO), a policy gradient algorithm that uses GPR to represent a probabilistic state transition model. It does not approximate a value function. PILCO can propagate Gaussian state distributions through its model of the dynamics and outputs a corresponding Gaussian distribution of the next state. PILCO has been used to learn block stacking with an inaccurate low-cost manipulator [DRF11; DFR15]. The algorithm requires that the reward function is known and can be computed analytically for a given state $\boldsymbol{x}$, which is a strong restriction. Linear policies were used in the presented experiments. PILCO is still considered to be a sample-efficient algorithm.

The original publication that introduces C-REPS [Kup+13] also presents a model-based extension that is called GPREPS because it uses GPR to represent robot and environment dynamics as well as the reward model. Further details are provided by Kupcsik et al. [Kup+17]. To learn a dynamic behavior such as ball throwing the model

was split into a dynamic model of the robot, a release model of the ball, a ball flight model, and a reward model. Hence, there is a lot of task-specific modeling and knowledge required to learn the state transition model and reward model.

State transition models can be learned with Bayesian methods as in PILCO or with local linear models as in GPS. Using more powerful function approximators such as neural networks is difficult because they tend to overfit and need a lot of samples. Nagabandi et al. [Nag+17] use medium-sized neural networks to learn state transition models for MPC. They notice that the model-based approach is better than model-free training particularly at the beginning of learning. After a while the model-based approach does not make progress anymore. Its asymptotic performance is worse because the model is biased. Nagabandi et al. [Nag+17] propose to initialize the model-free algorithm TRPO with a policy learned by DAgger from their model-based approach. In comparison to standard TRPO this combined approach achieves its final performance with three to five times fewer samples in the locomotion tasks of Open AI gym [Bro+16]. Probabilistic ensemble with trajectory sampling (PETS) [Chu+18a] addresses the shortcomings of the work of Nagabandi et al. [Nag+17] by using high-capacity, probabilistic state transition models. They use an ensemble of neural networks to learn epistemic uncertainty, which indicates the lack of data, and each neural network learns aleatoric uncertainty, which captures the stochasticity and noise in the data. Through the probabilistic ensemble we propagate several potential trajectories for a finite time horizon with actions maximized by CEM. Chua et al. [Chu+18a] show with Open AI gym benchmarks that this probabilistic model-based method is able to reach the same performance as model-free algorithms such as PPO, SAC, and TRPO with 8 to 125 times fewer samples. Nagabandi et al. [Nag+17] and Chua et al. [Chu+18a] do not address the problem of learning the reward model though. They assume that it is available and can be queried while sampling the state transition model. We cannot assume that this is possible in general. Suppose we want to test the quality of a grasp. We could lift the grasped object, move it around and measure whether the gripper still holds the object. This clearly cannot be easily expressed in an analytical reward function.

Appendix C.

Overview of Mathematical Notation

We will use bold lower case for vectors and bold upper case for matrices. The following table gives an overview of the most important symbols and functions that we use throughout this thesis. The second table gives an overview of the notation for mathematical operators.

Symbol	Explanation
$\boldsymbol{x}$	State; fully describes the state of an agent in some environment
$\mathcal{X}$	State space; often $\mathcal{X} \subseteq \mathbb{R}^n$ with $n \in \mathbb{N}$
$\boldsymbol{u}$	Action; fully describes the action of an agent in some environment
$\mathcal{U}$	Action space; often $\mathcal{U} \subseteq \mathbb{R}^n$ with $n \in \mathbb{N}$
$\boldsymbol{s}$	Context, task parameter(s), goal parameter(s)
$\mathcal{S}$	Context space; often $\mathcal{S} \subseteq \mathbb{R}^{n_s}$ with $n_s \in \mathbb{N}$
π	Policy; defines the behavior of an agent
$\boldsymbol{\theta}$	Parameters of a policy $\pi_{\boldsymbol{\theta}}$, $\boldsymbol{\theta} \in \mathbb{R}^n$ with $n \in \mathbb{N}$
ω	Parameters of an upper-level policy $\pi_{\omega}(\boldsymbol{\theta}\|\boldsymbol{s})$ that defines a distribution over policy parameters given a context
R	Return (accumulated rewards in an episode); might be a function $R(\boldsymbol{s}, \boldsymbol{\theta})$ that depends on context and policy parameters
ρ^{π}	Policy-dependent state visitation distribution
N	We will often use $N \in \mathbb{N}$ for the number of samples that are collected to update an upper-level policy, to update a search distribution, or in supervised learning
$\boldsymbol{q}$	Joint angles
$V(\boldsymbol{x})$	State value function; we write $V^{\pi}(\boldsymbol{x})$ to indicate that the value function is defined with respect to some policy π
$Q(\boldsymbol{x}, \boldsymbol{u})$	State-action value function
$A(\boldsymbol{x}, \boldsymbol{u})$	Advantage function, $A(\boldsymbol{x}, \boldsymbol{u}) = Q(\boldsymbol{x}, \boldsymbol{u}) - V(\boldsymbol{x})$
d_i	The weight of the i-th sample in weighted regression; often we organize these weights in a diagonal matrix $\boldsymbol{D} \in \mathbb{R}^{N \times N}$, where $\boldsymbol{D}_{ii} = d_i$
$\mathcal{L}(x\|y)$	Likelihood of x given y; $\mathcal{L}(x\|y) = p(y\|x)$

Symbol	Explanation
$L(\boldsymbol{w})$	Loss or error function; objective function that will be minimized in supervised learning
$D_{\mathrm{KL}}(p \parallel q)$	KL divergence of probability distributions given by their probability density functions p and q; note that we sometimes write $\mathbb{E}\left[D_{\mathrm{KL}}(p(\cdot\|x) \parallel q(\cdot\|x))\right]$ if we take the expectation of the KL divergence of conditional distributions to indicate that the expectation is computed over x while the KL divergence is computed for a fixed x
$\mathrm{clip}(a, b, c)$	Clipping function; limits the value of a to the interval $[b, c]$, that is, values of a less than b will become b and values of a greater than c will become c
$\mathbb{1}_{\{i=j\}}$	Kronecker delta; one if the equality holds and zero otherwise
δ_{ji}	Kronecker delta; in this case the value depends on equality of the indices

Operator	Explanation
$\mathbb{E}\left[X\right]$	Expected value; if it is used with a subscript $\mathbb{E}_{p(x)} = \mathbb{E}_{x\sim p}$ this means the expectation is computed with respect to x with the probability density function $p(x)$, other variables are assumed to be constant in this case
$\boldsymbol{A} \circ \boldsymbol{B}$	Element-wise multiplication of matrices or vectors
$\boldsymbol{q} * \boldsymbol{p}$	Quaternion product
$\hat{V}$	Hat operator; indicates estimates of quantities, for example, $\hat{V}$ is an estimate of V
$Tr\left[\boldsymbol{A}\right]$	Trace of a matrix
$a \leftarrow b$	This operator assigns the value of b to the variable a. The same variable can be used on the left and on the right side as in a programming language.

Appendix D.

Derivation of Cost-Regularized Kernel Regression

First we will take a look at how kernel ridge regression can be derived from linear regression and ridge regression. CrKR can be derived in a similar way from weighted linear regression or RWR that has been presented by Peters and Schaal [PS07]. CrKR also predicts the standard deviation similar to Gaussian process regression.

Our model is $\boldsymbol{y} = \boldsymbol{\Phi w}$, where $\boldsymbol{y} \in \mathbb{R}^N$ contains the target values of a one-dimensional regression problem, $\boldsymbol{\Phi} \in \mathbb{R}^{N \times D'}$ contains features of input vectors $\boldsymbol{\phi(x)}^T$ in each row, $\boldsymbol{w} \in \mathbb{R}^{D'}$ are the model parameters. For a dataset $(\boldsymbol{\Phi}, \boldsymbol{y})$ we want to find

$$\arg\min_{\boldsymbol{w}} L(\boldsymbol{w}) = \arg\min_{\boldsymbol{w}} \frac{1}{2} \left(\boldsymbol{y} - \boldsymbol{\Phi w}\right)^T \left(\boldsymbol{y} - \boldsymbol{\Phi w}\right).$$

We can rewrite the objective function (loss) to

$$L(\boldsymbol{w}) = \frac{1}{2}\boldsymbol{y}^T\boldsymbol{y} - \left[\frac{1}{2}\boldsymbol{y}^T\boldsymbol{\Phi w} + \frac{1}{2}\boldsymbol{w}^T\boldsymbol{\Phi}^T\boldsymbol{y}\right] + \frac{1}{2}\boldsymbol{w}^T\boldsymbol{\Phi}^T\boldsymbol{\Phi w}.$$

Because $\boldsymbol{y}^T\boldsymbol{\Phi w} = \sum_n \sum_d \boldsymbol{y}_n \boldsymbol{\Phi}_{\boldsymbol{n},\boldsymbol{d}} \boldsymbol{w}_{\boldsymbol{d}} = \sum_n \sum_d \boldsymbol{w}_d \boldsymbol{\Phi}^T_{d,n} \boldsymbol{y}_n = \boldsymbol{w}^T\boldsymbol{\Phi}^T\boldsymbol{y}$, we can write

$$L(\boldsymbol{w}) = \frac{1}{2}\boldsymbol{y}^T\boldsymbol{y} - \boldsymbol{w}^T\boldsymbol{\Phi}^T\boldsymbol{y} + \frac{1}{2}\boldsymbol{w}^T\boldsymbol{\Phi}^T\boldsymbol{\Phi w}.$$

Now it is easier to compute the derivatives. The gradient is

$$\nabla L(\boldsymbol{w}) = -\boldsymbol{\Phi}^T\boldsymbol{y} + \frac{1}{2}\left(\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \left(\boldsymbol{\Phi}^T\boldsymbol{\Phi}\right)^T\right)\boldsymbol{w},$$

where $\boldsymbol{\Phi}^T\boldsymbol{\Phi}$ is symmetric, because $\boldsymbol{\Phi}^T\boldsymbol{\Phi} = \left(\boldsymbol{\Phi}^T\left(\boldsymbol{\Phi}^T\right)^T\right)^T = \left(\boldsymbol{\Phi}^T\boldsymbol{\Phi}\right)^T$, hence we can simplify the gradient to

$$\nabla L(\boldsymbol{w}) = -\boldsymbol{\Phi}^T\boldsymbol{y} + \boldsymbol{\Phi}^T\boldsymbol{\Phi w}$$

The second derivative, the Hessian matrix, is

$$\nabla\nabla L(\boldsymbol{w}) = \boldsymbol{\Phi}^T\boldsymbol{\Phi}.$$

A convex quadratic function would have a positive definite Hessian. $\boldsymbol{\Phi}^T\boldsymbol{\Phi} \in \mathbb{R}^{D'\times D'}$ is positive definite iff $\boldsymbol{\Phi}$ has a full column rank, that is, we have enough samples. The optimum (minimum error) can be found at the zero of the gradient:

$$\begin{aligned} & \mathbf{0} = -\boldsymbol{\Phi}^T\boldsymbol{y} + \boldsymbol{\Phi}^T\boldsymbol{\Phi}\boldsymbol{w} \\ \Leftrightarrow \quad & \boldsymbol{\Phi}^T\boldsymbol{y} = \boldsymbol{\Phi}^T\boldsymbol{\Phi}\boldsymbol{w} \\ \Leftrightarrow \quad & \left(\boldsymbol{\Phi}^T\boldsymbol{\Phi}\right)^{-1}\boldsymbol{\Phi}^T\boldsymbol{y} = \boldsymbol{w}. \end{aligned}$$

The extension to ridge regression (linear regression with regularization) is straightforward. Minimizing the loss

$$\arg\min_{\boldsymbol{w}} L(\boldsymbol{w}) = \arg\min_{\boldsymbol{w}} \frac{1}{2}\left(\boldsymbol{y}-\boldsymbol{\Phi}\boldsymbol{w}\right)^T\left(\boldsymbol{y}-\boldsymbol{\Phi}\boldsymbol{w}\right) + \frac{\lambda}{2}\boldsymbol{w}^T\boldsymbol{w}$$

leads to the solution

$$\left(\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \lambda\boldsymbol{I}\right)^{-1}\boldsymbol{\Phi}^T\boldsymbol{y} = \boldsymbol{w}.$$

Now there are at least two ways to derive Kernel Ridge Regression from this. The first one is replacing weights by the weighted sum of training data. Let us take the loss

$$L(\boldsymbol{w}) = \frac{1}{2}\left(\boldsymbol{y}-\boldsymbol{\Phi}\boldsymbol{w}\right)^T\left(\boldsymbol{y}-\boldsymbol{\Phi}\boldsymbol{w}\right) + \frac{\lambda}{2}\boldsymbol{w}^T\boldsymbol{w}$$

and replace $\boldsymbol{w} = \boldsymbol{\Phi}^T\boldsymbol{\alpha}$, which results in

$$L(\boldsymbol{\alpha}) = \frac{1}{2}\left(\boldsymbol{y}-\boldsymbol{\Phi}\boldsymbol{\Phi}^T\boldsymbol{\alpha}\right)^T\left(\boldsymbol{y}-\boldsymbol{\Phi}\boldsymbol{\Phi}^T\boldsymbol{\alpha}\right) + \frac{\lambda}{2}\boldsymbol{\alpha}^T\boldsymbol{\Phi}\boldsymbol{\Phi}^T\boldsymbol{\alpha},$$

where $\boldsymbol{\Phi}\boldsymbol{\Phi}^T = \boldsymbol{K} \in \mathbb{R}^{N\times N}$ is a Gram matrix, that is, it contains inner products of all inputs of the dataset. We can rewrite this to [Bis06, page 293, Equation 6.5]

$$L(\boldsymbol{\alpha}) = \frac{1}{2}\boldsymbol{\alpha}^T\boldsymbol{K}\boldsymbol{K}\boldsymbol{\alpha} - \boldsymbol{\alpha}^T\boldsymbol{K}\boldsymbol{y} + \frac{1}{2}\boldsymbol{y}^T\boldsymbol{y} + \frac{\lambda}{2}\boldsymbol{\alpha}^T\boldsymbol{K}\boldsymbol{\alpha}.$$

The optimum is

$$\boldsymbol{\alpha} = \left(\boldsymbol{K} + \lambda\boldsymbol{I}\right)^{-1}\boldsymbol{y}.$$

We can make predictions for a new feature vector $\boldsymbol{\phi} = \phi(\boldsymbol{x})$ with

$$y = \boldsymbol{w}^T\boldsymbol{\phi} = \boldsymbol{\alpha}^T\boldsymbol{\Phi}\boldsymbol{\phi},$$

where we can define $\boldsymbol{k} = \boldsymbol{\Phi}\boldsymbol{\phi}$, hence,

$$y = \boldsymbol{\alpha}^T\boldsymbol{k}.$$

Note that $\boldsymbol{k}_n = \boldsymbol{\Phi}_n\boldsymbol{\phi}$, which is an inner product. Hence, the solution $\boldsymbol{\alpha}$ and the prediction y can be expressed purely by inner products of the feature vectors. Hence, we can use the kernel trick and replace those inner products with kernels. This allows us to approximate

nonlinear functions without actually transforming data with a nonlinear function $\boldsymbol{\phi}$. A typical example of a kernel is the RBF kernel $k(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\gamma||\boldsymbol{x} - \boldsymbol{x}'||^2\right)$ with the hyperparameter $\gamma > 0$.

The alternative solution is based on the formula [Bis06, Appendix C, Equation C.5]

$$(\boldsymbol{P}^{-1} + \boldsymbol{B}^T \boldsymbol{R}^{-1} \boldsymbol{B})^{-1} \boldsymbol{B}^T \boldsymbol{R}^{-1} = \boldsymbol{P}\boldsymbol{B}^T (\boldsymbol{B}\boldsymbol{P}\boldsymbol{B}^T + \boldsymbol{R})^{-1}.$$

To go directly from the solution of ridge regression to kernel ridge regression. Let us assign the variables $\boldsymbol{P}^{-1} = \lambda \boldsymbol{I}$, $\boldsymbol{B} = \boldsymbol{\Phi}$ and $\boldsymbol{R} = \boldsymbol{I}$. Using the formula, we can transform the solution of ridge regression:

$$\begin{aligned}
\boldsymbol{w} &= \left[\left(\boldsymbol{\Phi}^T \boldsymbol{\Phi} + \lambda \boldsymbol{I}\right)^{-1} \boldsymbol{\Phi}^T\right] \boldsymbol{y} \\
&= \left[\lambda^{-1} \boldsymbol{\Phi}^T \left(\lambda^{-1} \boldsymbol{\Phi}\boldsymbol{\Phi}^T + \boldsymbol{I}\right)^{-1}\right] \boldsymbol{y} \\
&= \left[\boldsymbol{\Phi}^T \left(\boldsymbol{\Phi}\boldsymbol{\Phi}^T + \lambda \boldsymbol{I}\right)^{-1}\right] \boldsymbol{y} \\
&= \boldsymbol{\Phi}^T \left(\boldsymbol{K} + \lambda \boldsymbol{I}\right)^{-1} \boldsymbol{y}
\end{aligned}$$

We can make new predictions for a feature vector $\boldsymbol{\phi}$ with

$$y = \boldsymbol{w}^T \boldsymbol{\phi} = \left(\boldsymbol{\Phi}^T \left(\boldsymbol{K} + \lambda \boldsymbol{I}\right)^{-1} \boldsymbol{y}\right)^T \boldsymbol{\phi} = \left(\left(\boldsymbol{K} + \lambda \boldsymbol{I}\right)^{-1} \boldsymbol{y}\right)^T \boldsymbol{\Phi}\boldsymbol{\phi}$$

This is exactly the same solution because we can define

$$\boldsymbol{\alpha} = \left(\boldsymbol{K} + \lambda \boldsymbol{I}\right)^{-1} \boldsymbol{y},$$

hence, the prediction reduces to

$$y = \boldsymbol{w}^T \boldsymbol{\phi} = \left(\left(\boldsymbol{K} + \lambda \boldsymbol{I}\right)^{-1} \boldsymbol{y}\right)^T \boldsymbol{\Phi}\boldsymbol{\phi} = \boldsymbol{\alpha}^T \boldsymbol{\Phi}\boldsymbol{\phi} = \boldsymbol{\alpha}^T \boldsymbol{k}.$$

Now that we extended linear regression to kernel ridge regression we know how to make the same extension from weighted linear regression or RWR [PS07] to CrKR.

RWR is essentially linear regression with weighted samples, that is, assuming that we have multiple output dimensions the weight matrix $\boldsymbol{W} \in \mathbb{R}^{D' \times F}$ is

$$\boldsymbol{W} = (\boldsymbol{\Phi}^T \boldsymbol{D} \boldsymbol{\Phi} + \lambda \boldsymbol{I})^{-1} \boldsymbol{\Phi}^T \boldsymbol{D} \boldsymbol{Y},$$

where $\boldsymbol{D}$ is a diagonal matrix that contains a weight per sample. $\boldsymbol{Y} \in \mathbb{R}^{N \times F}$ now is a matrix that contains one output vector per row. $\lambda = 0$ in the original paper [PS07], but we already introduce regularization here to save one step. RWR has been extended to a form of kernel ridge regression with the second method that we used to derive kernel ridge regression from ridge regression by Kober et al. [Kob+12] to obtain a new contextual policy search algorithm that is called CrKR. From reward-weighted regression with one output dimension,

$$\boldsymbol{w} = (\boldsymbol{\Phi}^T \boldsymbol{D} \boldsymbol{\Phi} + \lambda \boldsymbol{I})^{-1} \boldsymbol{\Phi}^T \boldsymbol{D} \boldsymbol{y},$$

we can go to cost-regularized kernel regression

$$\begin{aligned} \boldsymbol{w} &= \left[(\boldsymbol{\Phi}^T \boldsymbol{D} \boldsymbol{\Phi} + \lambda I)^{-1} \boldsymbol{\Phi}^T \boldsymbol{D}\right] \boldsymbol{y} \\ &= \left[\lambda^{-1} \boldsymbol{\Phi}^T (\boldsymbol{\Phi} \lambda^{-1} \boldsymbol{\Phi}^T + \boldsymbol{D}^{-1})^{-1}\right] \boldsymbol{y} \\ &= \left[\boldsymbol{\Phi}^T (\boldsymbol{\Phi} \boldsymbol{\Phi}^T + \lambda \boldsymbol{D}^{-1})^{-1}\right] \boldsymbol{y} \end{aligned}$$

Note that we have to learn a separate model for each output dimension. Predictions can be made with

$$y = \boldsymbol{w}^T \boldsymbol{\phi} = \left(\boldsymbol{\Phi}^T \left(\boldsymbol{K} + \lambda \boldsymbol{D}^{-1}\right)^{-1} \boldsymbol{y}\right)^T \boldsymbol{\phi} = \left(\left(\boldsymbol{K} + \lambda \boldsymbol{D}^{-1}\right)^{-1} \boldsymbol{y}\right)^T \boldsymbol{\Phi} \boldsymbol{\phi},$$

where $\boldsymbol{\alpha} = (\boldsymbol{K} + \lambda \boldsymbol{D}^{-1})^{-1} \boldsymbol{y}$ can be precomputed, hence,

$$y = \boldsymbol{\alpha}^T \boldsymbol{\Phi} \boldsymbol{\phi} = \boldsymbol{\alpha}^T \boldsymbol{k}.$$

Similar to GPR [RW05], we can also compute a standard deviation

$$\sigma = \sqrt{k(\boldsymbol{x}, \boldsymbol{x}) + \lambda - \boldsymbol{k}^T \left(\boldsymbol{K} + \lambda \boldsymbol{D}^{-1}\right)^{-1} \boldsymbol{k}}$$

of the prediction,[1] where $k(\boldsymbol{x}, \boldsymbol{x})$ is the kernel of the sample with itself and $\boldsymbol{k} = \boldsymbol{\Phi} \boldsymbol{\phi}$ is a vector that contains the kernels between the sample and the whole training set.

[1] Note that we cannot directly derive this form of uncertainty estimate from the maximum likelihood as defined in Section 2.3.3. Refer to Kober et al. [Kob+12] and Rasmussen and Williams [RW05] for details.

Appendix E.

Preliminary Experiments with Active Contextual Policy Search

This appendix contains preliminary experiments that were conducted with active contextual policy search but were not relevant enough to be included in Section 4.1. It is joint work with Jan Hendrik Metzen, who wrote Section E.2.

This appendix was published originally as [FM14] and has been revised.

E.1. Model of the Contextual Learning Problem

In this section, we show how contextual policy search algorithms can benefit from active context selection by means of a simple artificial model of the contextual learning problem. The model abstracts away the contextual policy search which is possible because our approach treats it as a black box (see Figure 4.1).

We assume that the context space $\mathcal{S} = \{0, 1, \ldots, 9\}$ is discrete and associated to each context $\boldsymbol{s}$ is a hidden value $l_{\boldsymbol{s}} \in [0, 100]$ that indicates the agent's competence in $\boldsymbol{s}$, that is, how well $\boldsymbol{s}$ has been learned. Large values of $l_{\boldsymbol{s}}$ simulate that the current policy parameters in context $\boldsymbol{s}$ are close to the optimal policy parameters $\boldsymbol{\theta}^*(\boldsymbol{s})$. The true return in context $\boldsymbol{s}$, which is given by

$$R(\boldsymbol{s}) = (1 + \exp(-0.1 l_{\boldsymbol{s}} + 4))^{-1} + b_{\boldsymbol{s}},$$

depends directly on $l_{\boldsymbol{s}}$. $R(\boldsymbol{s})$ corresponds to a scaled and shifted logistic function so that $R(\boldsymbol{s}) \approx b_{\boldsymbol{s}}$ if $l_{\boldsymbol{s}} \approx 0$ and $R(\boldsymbol{s}) \approx 1 + b_{\boldsymbol{s}}$ if $l_{\boldsymbol{s}} \geq 100$. In real learning problems, different tasks often have a different maximum return. To simulate this, we artificially create a true return baseline $b_{\boldsymbol{s}}$ for each context. The baseline is randomly sampled from a normal distribution with zero mean and standard deviation σ_b. The true return is not observed directly. Instead, we add Gaussian noise with zero mean and standard deviation σ_r to simulate trial and error of a learning agent.

We assume that each context has an intrinsic complexity which controls how much the agent learns in a single episode in this context. This complexity can change abruptly between neighboring contexts even in continuous domains. This model is motivated for instance by reaching tasks, in which it might happen that a slight modification of the goal position requires that the agent needs to avoid an obstacle that blocks the direct path. As a result, the slightly different context corresponding to the blocked goal would have a significantly worse expected learning progress than its neighbors and its solution

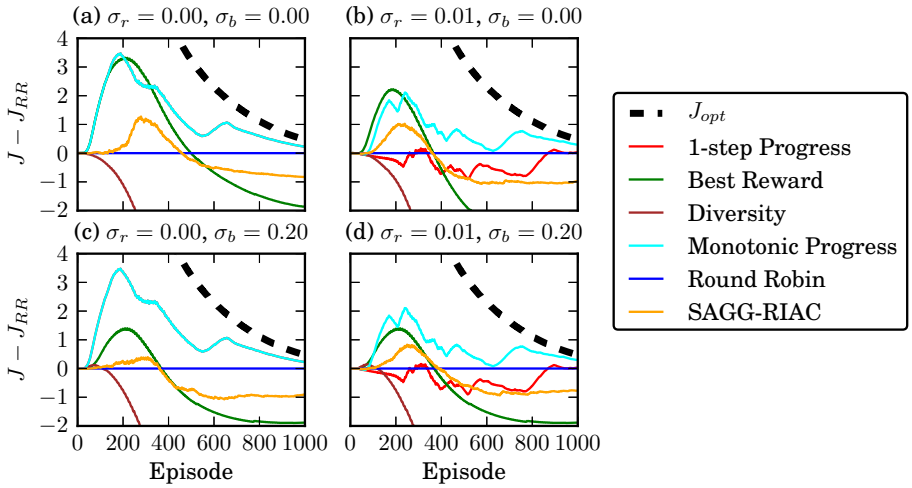


Figure E.1.: Relative learning curves of several active context selection methods. The performance of the round robin context selection baseline (J_{RR}) is subtracted from all learning curves. The values of the standard deviation for return measurements σ_r and of the standard deviation for the baseline σ_b have been varied. After 1000 episodes the monotonic progress heuristic has approximately reached the upper bound of the performance J_{opt}.

cannot be transferred well. We model this behavior of learning progress and skill transfer by assigning a different learning progress factor $w_{\boldsymbol{s}} \in \{0.1^2, 0.2^2, \ldots, 1^2\}$ to each context randomly, where large $w_{\boldsymbol{s}}$ corresponds to a larger improvement of competence after one additional episode in $\boldsymbol{s}$.

Each time, the return of a context $\boldsymbol{s}_t$ will be queried, the values $l_{\boldsymbol{s}}$ of each contexts $\boldsymbol{s}$ will be updated to simulate the learning progress according to the update rule

$$l_{\boldsymbol{s}}^{(t)} = l_{\boldsymbol{s}}^{(t-1)} + w_{\boldsymbol{s}_t} \cdot 0.5^{|\boldsymbol{s}-\boldsymbol{s}_t|},$$

which models that experience obtained in one context generalizes to other contexts based on their similarity (measured here using the euclidean distance). Contexts which are easier learnable (large $w_{\boldsymbol{s}}$) lead to higher overall learning progress at the beginning. However, it does not make sense to focus at the context with maximum w_s indefinitely because $R(s)$ saturates once $l_s \geq 100$. Hence, the estimate of the learning progress has to be adaptive. Since we only have a discrete set of contexts, we can exactly compute $J = \sum_{\boldsymbol{s}} R(\boldsymbol{s})$ and the upper bound of J is $J_{opt} = 10 + \sum_{\boldsymbol{s}} b_{\boldsymbol{s}}$.

In addition to our proposed methods, we examine the context generation and selection method from SAGG-RIAC and context selection in a fixed order (round robin). In order to show different properties of the active context selection methods, we display the number of episodes versus the relative J in comparison to round robin selection in Figure E.1. We have used the parameters $\gamma = 0.95$ and $\xi = 10^{-8}$ in all heuristics that are based on D-UCB. For the diversity and the best-reward heuristics we have used $B = 1$ and for

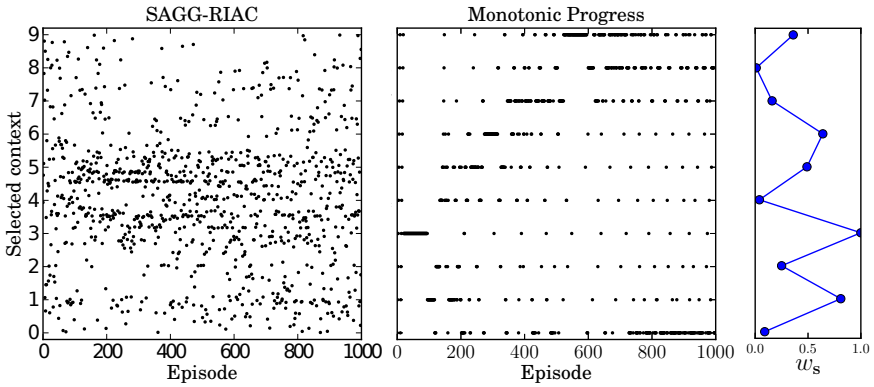


Figure E.2.: Selected contexts per episode with SAGG-RIAC and monotonic progress heuristic and D-UCB (with $\sigma_r = 0$). The learning progress factor w_s of each context is displayed on the right side.

all others $B = 0.25$. For SAGG-RIAC, we set the maximum size of samples per region before we split the region to 8 and the window size which is used to compute the interest value is 10.

We can see that despite different learning progress factors, round robin context selection is a good baseline. The best-reward heuristic that focuses on the best learnable context is a good heuristic at the beginning. However, when the best context approaches the optimum return, the learning progress decreases and approaches zero. At this time it would be better to switch to a context in which greater learning progress can be achieved. This will be even more significant for larger context spaces because the transferability of knowledge decreases with the size of the context space and the complexity of the different contexts. In addition, an artificial baseline for each return (see Figure E.1 (c) and (d)) leads to severe degradation because the context with maximum $R(s)$ is not necessarily the context with maximum learning progress. The diversity heuristic does not work well either. This is because the differences of the expected learning progress are too large between contexts and it will select the worst learnable context.

The 1-step progress heuristic and monotonic progress heuristic essentially focus on the same context as the best-reward heuristic at the beginning. But they switch to other contexts with greater learning progress when the learning progress in the best learnable context decreases. Moreover, the 1-step progress and monotonic progress heuristics are invariant under different baselines. The heuristics behave identically when the return is noise-free, that is $\sigma_r = 0$. If there is noise (which simulates the exploration of the agent), the monotonic progress heuristic is better (see Figure E.1 (b) and (d)).

A context selection method that differs from all others is the context generation and selection method from SAGG-RIAC. It is designed for continuous context spaces. However, it has a crucial disadvantage in our model of the learning progress: we assume that the expected learning progress of neighboring contexts can change abruptly. This is a problem for SAGG-RIAC because it focuses on *regions* of the context space that

have a high return derivative. Among these are not only regions with a high learning progress but also regions with abruptly changing learning progress. In Figure E.2 we can see which contexts have been selected by SAGG-RIAC during the simulated learning process: it focuses most of the time on the region around the contexts 3, 4 and 5 because of the greatly varying learning progress factor $w_{\boldsymbol{s}}$ in this region, which results in a high competence derivative. Therefore, a significant part of the explored contexts are not informative because the context 4 has a low learning progress.

The monotonic progress heuristic, in contrast, selects most of the time tasks with a high learning progress as we can see in Figure E.2. At the beginning, it focuses on the context 3 which has the highest $w_{\boldsymbol{s}}$. After some time, when the learning progress in this context saturates, it concentrates on other contexts. At the end it concentrates on the contexts that have not been learned perfectly yet even though they have a low intrinsic learning progress factor.

E.2. Contextual Function Optimization

In this section, we evaluate the proposed approach on an artificial test problem, compare it to reasonable baseline methods, and analyze the effect of different intrinsic reward heuristics. The test problem is chosen such that some contexts are harder in the sense that the parameters $\boldsymbol{\theta}$ need to be chosen more precisely to reach the same level of return. By focusing on learning primarily the parameters $\boldsymbol{\theta}$ for these contexts, active context selection should be able to outperform a uniform random context selection.

E.2.1. Problem Domain

The context is denoted by $\boldsymbol{s} \in S = [-1, 1]^{n_s}$. We use the objective function

$$f(\boldsymbol{\theta}, \boldsymbol{s}) = -||A\boldsymbol{\theta} - \boldsymbol{s}||_2 \cdot ||s||_2^2 + \sum_{i=0}^{n_s-1} \boldsymbol{s}_i,$$

where $\boldsymbol{\theta} \in \mathbb{R}^{n_\theta}$ denotes the low-level parameters and the matrix $A \in [0, 1]^{n_\theta \times n_s}$ is chosen uniform randomly such that it has rank n_s ($n_\theta > n_s$). The objective function consists of three terms: the *parameter error* $-||A\boldsymbol{\theta}-\boldsymbol{s}||_2$ which can be influenced by the agent's choice of $\boldsymbol{\theta}$, the *context complexity* $||s||_2^2$, which controls how strongly the agent's parameter error deteriorates the task performance, and the *baseline* $\sum_{i=0}^{n_s-1} \boldsymbol{s}_i$, which controls the maximum value in a context. Since A has rank n_s, $\boldsymbol{\theta}$ can always be chosen such that the parameter error becomes 0 and thus the optimal value $f^*(\boldsymbol{s})$ is equal to the baseline $\sum_{i=0}^{n_s-1} \boldsymbol{s}_i$. However, if the agent chooses $\boldsymbol{\theta}$ suboptimally, the same parameter error has different effects on the value of f in different contexts: in contexts with high context complexity, the value of f will be considerably smaller than f^*, while the difference will be less pronounced in contexts with lower context complexity. Most extremely, for $\boldsymbol{s} = \boldsymbol{0}$, the choice of $\boldsymbol{\theta}$ is arbitrary since f will always be equal to f^*. Thus, an agent should focus on learning $\boldsymbol{\theta}$ in the contexts with high complexity if its objective is to minimize $|f(\boldsymbol{\theta}, \boldsymbol{s}) - f^*(\boldsymbol{s})|$.

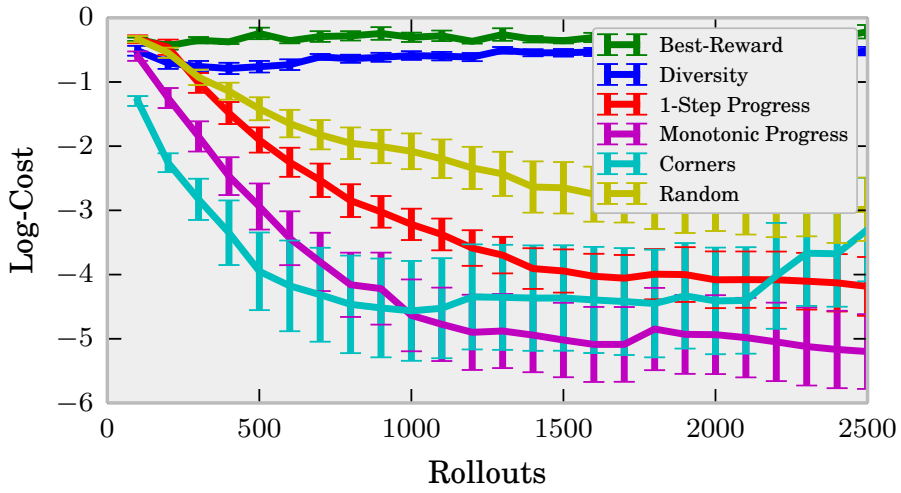


Figure E.3.: Learning curves of different task selection heuristics. A contextual upper-level policy has been learned using C-REPS for different active task-selection strategies. The logarithm of the cost $|f(\boldsymbol{\theta}, \boldsymbol{s}) - f^*(\boldsymbol{s})|$ averaged over 100 test contexts is used as performance measure. Shown are mean and standard error of the mean for 20 runs of 2500 episodes. (Illustrated by Jan Hendrik Metzen in Fabisch and Metzen [FM14].)

E.2.2. Comparison of Task Selection Heuristics

In a first experiment, we compare task selection with D-UCB for different intrinsic reward heuristics to two baseline methods. In this experiment, training takes place on 25 contexts placed on an equidistant grid over a context space with $n_s = 2$ dimensions; that is, the set of contexts that will be used for training is $S_{\text{train}} = [-1, -\frac{1}{2}, 0, \frac{1}{2}, 1]^2$. The objective of learning, however, is to generalize π_ω over the entire context space $\mathcal{S}$, that is, to choose $\pi_\omega(\boldsymbol{\theta}|\boldsymbol{s})$ such that $f(\boldsymbol{\theta}, \boldsymbol{s})$ is maximized. As baseline, we use a *Random* task-selection heuristic, which selects uniform randomly among the training contexts. Moreover, we use a *Corner* task-selection heuristic, which selects the four contexts, where the context complexity is maximal, that is $\boldsymbol{s} = (\pm 1, \pm 1)$, in a round-robin fashion.

For the D-UCB, we have used $B = 1.0$, $\gamma = 0.99$, and $\xi = 10^{-8}$. The external return $R(\boldsymbol{s}, \boldsymbol{\theta})$, based on which the intrinsic reward r_β is computed, is set to $R(\boldsymbol{s}, \boldsymbol{\theta}) = f(\boldsymbol{\theta}, \boldsymbol{s})$ where $\boldsymbol{\theta} = \pi_\omega(\boldsymbol{s})$ is sampled from the upper-level policy for the given context $\boldsymbol{s}$. Note that the returns in $\boldsymbol{s}$ have high variance because of the agent's explorative behavior and are also non-stationary since they depend on the current upper-level policy π_ω. Contextual policy search was conducted with C-REPS with $\epsilon = 2.0$, $N = 50$, and performing an update every 25 episodes. The evaluation criterion is the expected value of $|f(\pi_\omega(\boldsymbol{\theta}|\boldsymbol{s}), \boldsymbol{s}) - f^*(\boldsymbol{s})|$ of the learned contextual policy π_ω over the context space $\mathcal{S}$, where exploration of π_ω is disabled. We approximate this quantity by computing the average return of π_ω on 100 test contexts sampled uniform randomly from $\mathcal{S}$.

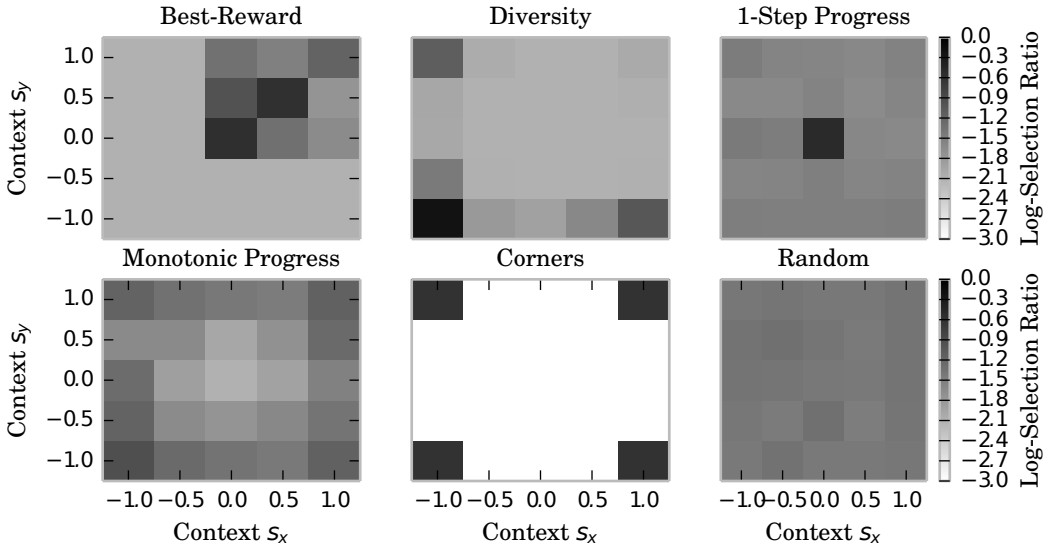


Figure E.4.: Task preferences of different task-selection strategies during the first 250 episodes of training. Shown is the logarithm of the mean selection ratio. (Illustrated by Jan Hendrik Metzen in Fabisch and Metzen [FM14].)

Figure E.3 shows the learning curves for different task-selection strategies. Figure E.4 shows which contexts (tasks) are selected by the different strategies during the first 250 episodes of training. D-UCB with the *Best-Reward* and the *Diversity* intrinsic reward performs significantly worse than a uniform random task selection. The reason for this is that *Best-Reward* focuses mostly on tasks where the baseline term is large (upper right area in Figure E.4) or where the task complexity is small (central area). Conversely, *Diversity* focuses on areas where the baseline term is small. Both strategies are too imbalanced if the baseline term's contribution is not negligible.

D-UCB with the *1-step Progress* intrinsic reward performs equally bad during the first 250 episodes. The reason for the low initial progress is that the *1-step Progress* intrinsic reward not only rewards progress but also penalizes regression. However, regression is inevitable during the initial explorative phase. Because of this, this intrinsic reward heuristic focuses initially on contexts with small context complexity where the parameter error and thus the explorative behavior have only a small effect on the actual return. After the initial explorative phase, this intrinsic reward gets more informative and the corresponding active task selection outperforms uniform random task selection in the long run.

D-UCB with the *Monotonic Progress* intrinsic reward performs considerably better than both D-UCB with the other heuristics and uniform random task selection. The reason for this is that it initially favors complex contexts (the outer areas in Figure E.4 with $||s||_2 \gg 0$), where the potential return improvement is large, without any preference for tasks with small or large baseline value. This task-selection strategy works well and

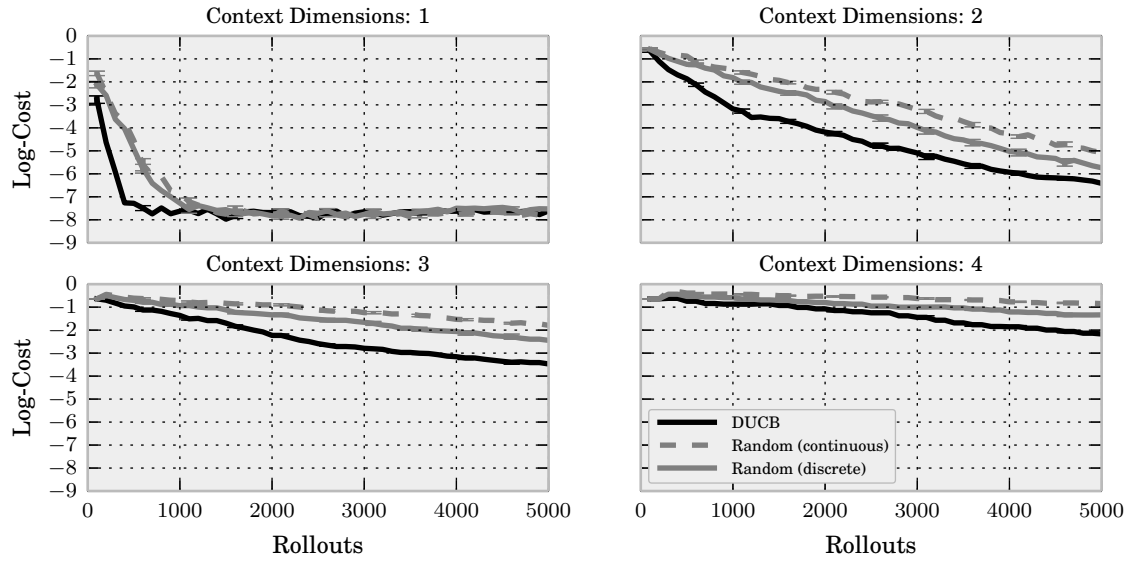


Figure E.5.: Learning curves for different context dimensionality. Shown are mean and standard error of the mean for 20 runs of 5000 episodes. (Illustrated by Jan Hendrik Metzen in Fabisch and Metzen [FM14].)

results in a large and stable learning progress. Based on this, we have tested a second baseline denoted as *Corners*, which selects the four contexts with maximum context complexity in a round-robin fashion. While this resulted in a rapid learning progress initially, it is slightly suboptimal and unstable in the long run since only four of the tasks are ever sampled. Conversely, the *Monotonic Progress* intrinsic reward leads to a more balanced selection of tasks when converging and is thus favorable in the long run. In summary, D-UCB with the *Monotonic Progress* intrinsic reward selects tasks in a way which increased C-REPS' learning progress considerably and proved to be stable at the same time.

E.2.3. Dimensionality of the Context Space

In a second experiment, we compare the performance of D-UCB to uniform random task selection for different dimensionality of the context space. A discrete set $\mathcal{S}_{\text{train}}$ of 25 contexts for training has been generated by selecting the k-the context $\boldsymbol{s}_k$ uniform random from $\mathbb{R}^{n_s}$ under the constraint $||\boldsymbol{s}_k||_2 = k/25$. The set of test contexts has been generated in the same way but with an other random seed. D-UCB has been combined with the *Monotonic Progress* heuristic. The D-UCB parameters have been set to $B = 1.0$, $\gamma = 0.99$, and $\xi = 10^{-8}$, and the C-REPS parameters to $\epsilon = 1.0$ and $N = 25n_s^2$, and an update was performed every $13n_s^2$ episodes. As baseline, two different random context selection strategies have been tested: *Random (discrete)* chooses tasks uniform randomly

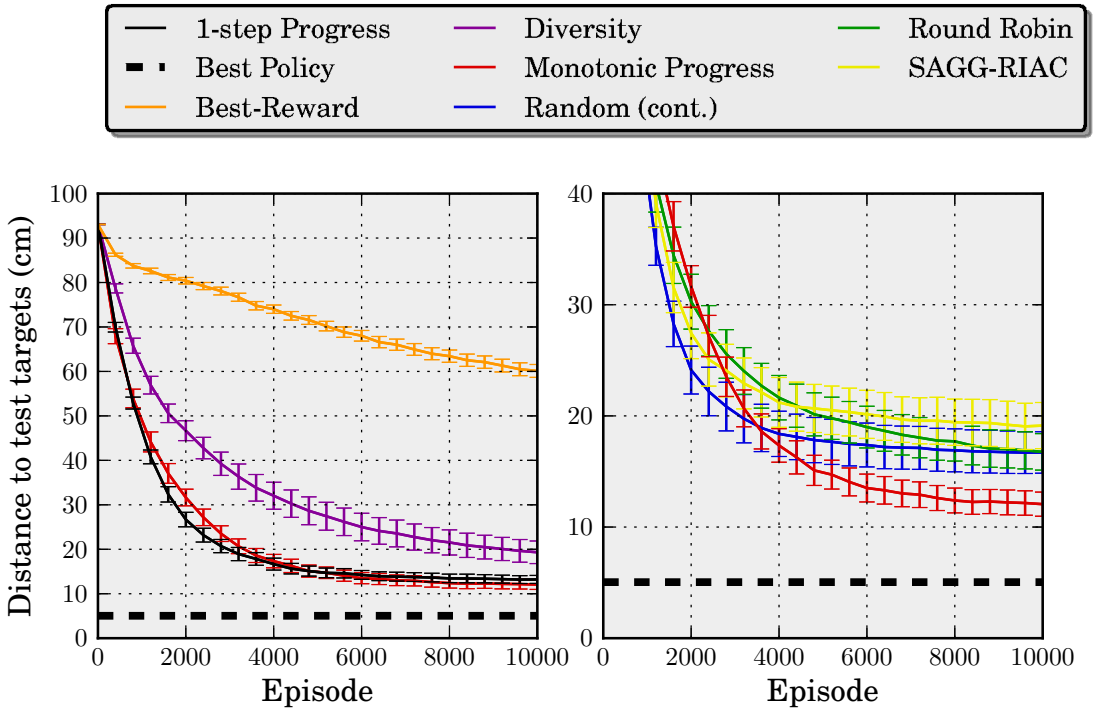


Figure E.6.: Learning curves of Figure 4.3 including SAGG-RIAC.

from $\mathcal{S}_{\text{train}}$ while *Random (continuous)* chooses tasks uniform randomly from $\mathbb{R}^{n_s}$ under the constraint $||\boldsymbol{s}_k||_2 \leq 1$.

Figure E.5 shows the learning curves for $n_s \in \{1, 2, 3, 4\}$. For any value of n_s, D-UCB outperforms random task selection in the initial learning phase (not the final performance). We can further see that selecting a finite, discrete set of training contexts from the continuous context space does not necessarily impair performance; conversely, *Random (discrete)* performs slightly better than *Random (continuous)*. While the general learning progress decreases considerably for higher dimensionality n_s, this is not directly an issue of the task-selection strategy (since D-UCB still outperforms random task selection) but rather of the underlying contextual policy search method. Thus, the results show that active context selection with D-UCB works satisfyingly while higher dimensional context spaces remain a general challenge for contextual policy search.

E.3. SAGG-RIAC for Ball Throwing

The evaluation in Section 4.1.3 does not contain a comparison to SAGG-RIAC, which has been done though in response to the review but was not included in the final paper. SAGG-RIAC focuses on regions with high reward derivatives. Figure E.6 shows that this is not beneficial in our experiment with a static environment.

Appendix F.

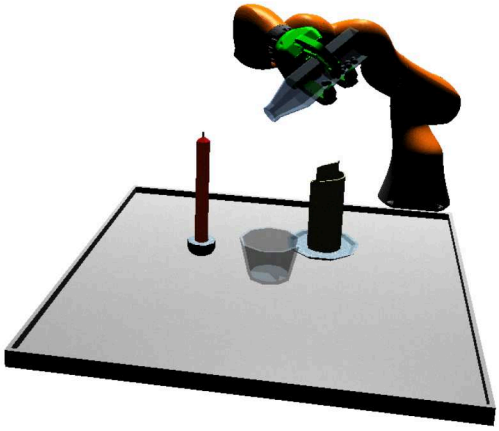
Descriptions of Robots

F.1. COMPI

The robotic arm COMPI has been developed at the Robotics Innovation Center of the DFKI. It is presented in Bargsten and de Gea Fernández [Bd15]. Technical specifications can be found at the corresponding website [Bar20]. It has a maximum arm length of 94 cm, a maximum payload of 2 kg, a weight of 4.75 kg, and six DOF.

We control joint positions at 1 kHz in our experiments. For throwing motions it was particularly useful that joint trajectories could be executed with high repeatability because of excellent controller configuration that also considers the spring between the second and third joint. Furthermore, COMPI is able to exhibit high accelerations because of its low weight in comparison to industrial robot arms. In our experiments we attached a scoop to the arm that can be used to throw a ball.

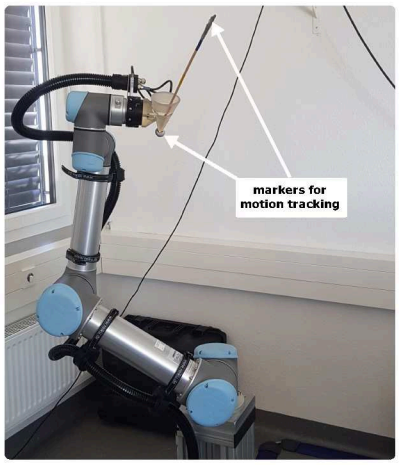
F.2. Kuka iiwa 7/14



Both variants of the iiwa, Kuka iiwa 14 R820 and Kuka iiwa 7 R800 have seven DOF. We use the smaller version with a maximum payload of 7 kg payload only in simulation and the larger version with 14 kg payload in simulated and real experiments. More details can be found at the corresponding website [Kuk20].

We can control either joints or end-effector poses with a frequency of 200 Hz. In our experiments we either only control the pose of the end-effector or we attach a gripper with three fingers if we want to grasp or hold something.

F.3. Universal Robot UR5/10



The UR5 and UR10 of Universal Robots can carry 5 kg and 10 kg of payload respectively. The robot arms have six DOF and can be controlled in various modes at 125 Hz. Further technical details can be found at the corresponding website [Uni20]. Although

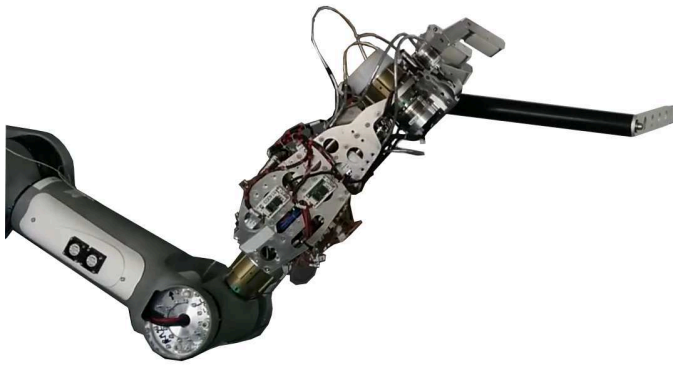
in this thesis it has been used for imitation learning from motion capture data, it also provides a compliant freedrive mode that can be used for kinesthetic teaching.

We send joint position commands in our experiments. For grasping we use a gripper with two fingers and one DOF. In the throwing experiments we use a specific cone that holds the stick.

F.4. Mitubishi PA-10

The Mitsubishi PA-10 7C has seven DOF. Further technical details can be found at DFKI's website [Gea20]. In this thesis the robot is only used in simulation.

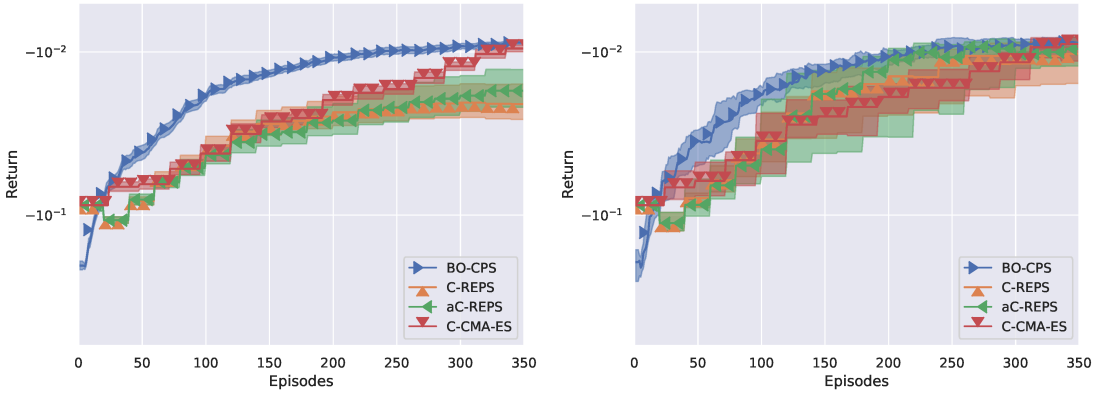
F.5. Mantis' Arm



We used an arm of DFKI's robot Mantis for learning how to pull a lever. The robot Mantis is presented by Bartsch et al. [Bar+16] and at DFKI's website [Bri20]. The arm was detached from the rest of the robot and mounted at a fixed structure for this purpose. The stand-alone arm has six DOF.

Appendix G.

Detailed Evaluation of Variational Trajectory Autoencoder



(a) Learning curves: mean and standard error.

(b) Learning curves: 25-, 50-, and 75-percentiles.

Figure G.1.: Comparison of all contextual policy search algorithms on the reaching problem. Note that the median performances are more similar than mean performances.

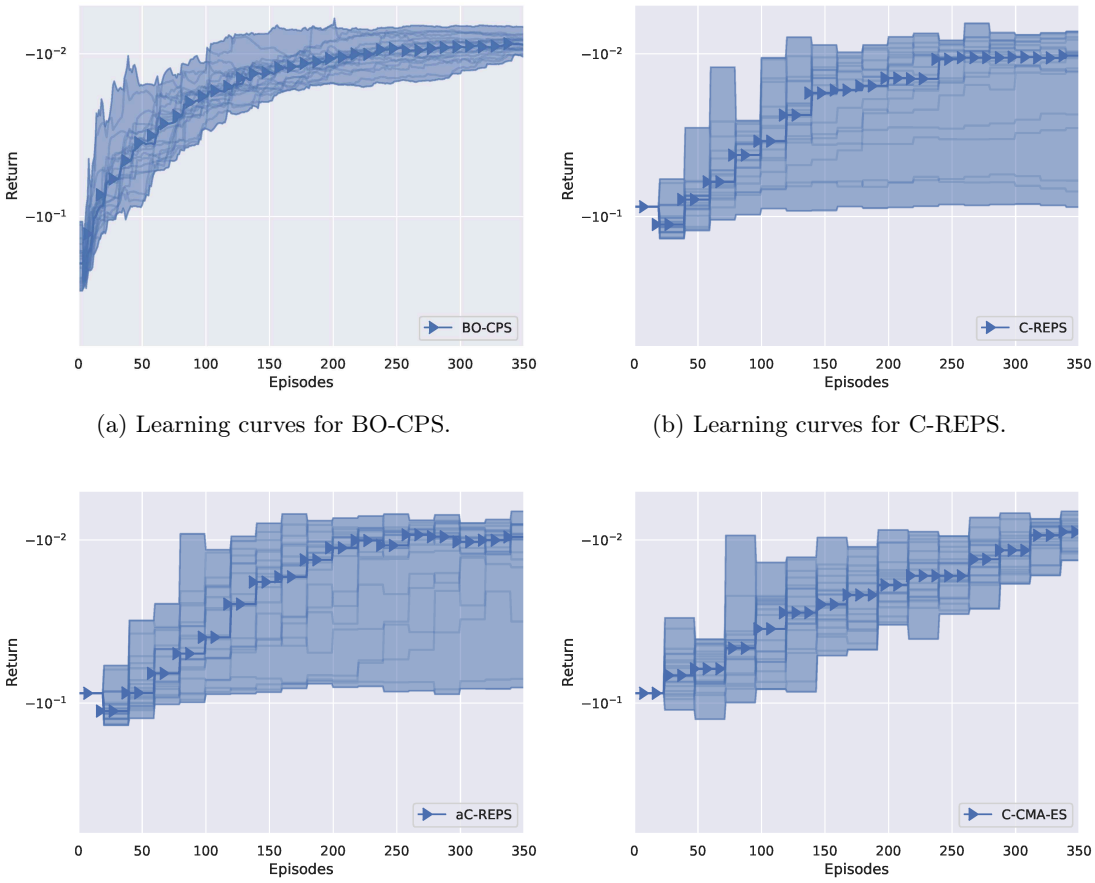
This is a detailed analysis of the results from Section 4.5. We analyze the convergence behavior of all contextual policy search algorithms by learning for more than 250 episodes. Figure G.1 shows the learning curves for all algorithms. Different between the two plots are the statistics: mean and standard error on the left and percentiles on the right.

Although the differences in regard to percentiles between BO-CPS and the other algorithms is negligible, it is larger with respect to the mean. After 350 episodes, however, C-CMA-ES is able to reach a performance similar to the one of BO-CPS. The differences between the algorithms that we see here are not as striking as in previous evaluations because all of them benefit from the low dimensionality of the search space. C-CMA-ES, an algorithm that inherits many properties from the well-tuned black-box optimizer CMA-ES, benefits more than C-REPS, which has previously mostly been evaluated for problems with 20 or more dimensions, in which parameters or metaparameters of move-

ment primitives had to be learned. There is only a slight advantage of aC-REPS over C-REPS.

The difference that we noticed in G.1 is best explained by Figure G.2, which shows all 20 individual learning curves per algorithm. BO-CPS shows the most consistent improvement over all experiments and continuous improvement. C-REPS and aC-REPS work well in most cases, but there are some outliers that converge too early, which can be noticed in the mean but not in most percentiles, for instance, not in the median.

If we are only able to learn once on a robotic system, we have to make sure that we select the most reliable algorithm. This is BO-CPS in this case, closely followed by C-CMA-ES.



(a) Learning curves for BO-CPS.

(b) Learning curves for C-REPS.

(c) Learning curves for aC-REPS.

(d) Learning curves for C-CMA-ES.

Figure G.2.: Learning curves per individual run. Each plot shows all learning curves for all experiments with one algorithm (thin lines). The area marks the interval between the best and the worst returns, and the median is indicated by the thick line.

Glossary

A3C Asynchronous Advantage Actor Critic. 59–61

aC-REPS active training set selection C-REPS. 118–121, 145, 148, 149, 242

ACER Actor Critic with Experience Replay. 59

ACM-ES CMA-ES with a ranking SVM as surrogate model. 123, 124, 127, 170, 180

aCMA-ES active CMA-ES. 123

ALE Arcade Learning Environment. 52, 60, 61

BFGS Broyden-Fletcher-Goldfarb-Shanno. 39

BO Bayesian Optimization. 39, 40, 50, 165, 170

BO-CPS Bayesian Optimization for Contextual Policy Search. 50, 80, 130–136, 141, 143, 145–150, 165, 170, 177, 179–181, 241, 242

C-ACM-ES Contextual ACM-ES. 124, 126–128, 130, 148

C-CMA-ES Contextual Covariance Matrix Adaptation Evolution Strategies. 50, 69, 74, 76–78, 80, 123, 125, 127–130, 145, 149, 150, 170, 173, 177, 241, 242

C-MORE Contextual Model-based Relative Entropy Stochastic Search. 50, 80

C-REPS Contextual Relative Entropy Policy Search. 50, 68, 73–76, 78–80, 104, 109, 111, 117–121, 123, 127, 129, 132, 133, 143, 145, 149, 150, 165, 170, 173, 177, 220, 241, 242

CEM Cross-Entropy Method. 38, 56, 64, 221

CMA-ES Covariance Matrix Adaptation Evolution Strategies. 38–40, 44, 47, 48, 50, 56, 76, 85, 93, 94, 96, 123, 138, 158, 164, 170, 180, 241

CNN Convolutional Neural Network. 62, 64

COCO Comparing Continuous Optimisers. 76, 79

CrKR Cost-Regularized Kernel Regression. 49, 50, 68, 72, 74, 225, 227

D-UCB Discounted Upper Confidence Bound. 101, 105, 106, 109–114, 230, 231, 233–236

DAgger dataset aggregation. 32, 221

DDPG Deep Deterministic Policy Gradients. 55–57, 62–64, 181

DIRECT Dividing Rectangles. 39, 132, 145

DMP Dynamical Movement Primitive. 33–36, 44, 46, 49, 50, 66, 67, 93–99, 108, 109, 120, 129, 132, 133, 136, 138, 149, 157, 158, 160, 164, 165, 170, 179

DOF degrees of freedom. 9, 19, 44, 46, 62, 64, 94, 193, 197, 202, 204, 205, 207, 219, 237–240

DPG Deterministic Policy Gradients. 55

DQN Deep Q Networks. 52, 55, 59, 66, 181

DTW dynamic time warping. 91

EI expected improvement. 131

ES Evolution Strategy. 37, 38, 40, 48, 65

GAE Generalized Advantage Estimation. 57, 58

GECCO Genetic and Evolutionary Computation Conference. 180

GP-UCB Gaussian process Upper Confidence Bound. 131, 133, 134, 136, 145

GPR Gaussian process regression. 130, 131, 133, 181, 220, 228

GPS Guided Policy Search. 62, 66, 221

HER Hindsight Experience Replay. 55, 56, 63, 181

iid independent and identically distributed. 79, 105, 125

IK inverse kinematics. 86, 93–97, 99

IL imitation learning. 154, 156, 160–162, 164, 178

iLQG iterative linear-quadratic-Gaussian regulator. 62

IMPALA Importance Weighted Actor-Learner Architecture. 59–61

IMU inertial measurement unit. 208

KL Kullback-Leibler. 39, 46, 47, 56, 61–63, 68, 224

L-BFGS Limited-memory BFGS. 39, 85

L-BFGS-B L-BFGS for Bound Constrained Optimization. 39, 47, 132, 145

LiDAR Light Detection and Ranging. 209

LQR linear-quadratic regulator. 205

MABP Multi-armed bandit problem. 105, 106

MAML Model-Agnostic Meta Learning. 219

MAP maximum a posteriori. 72

MCTS Monte Carlo tree search. 220

MDP Markov decision process. 41, 66

MORE Model-based Relative Entropy Stochastic Search. 47, 50

MPC model predictive control. 16, 221

MSBE mean squared Bellman error. 42, 51

NAC Natural Actor-Critic. 39, 44, 46, 47

NAF Normalized Advantage Function. 52

NES Natural Evolution Strategies. 38–40, 44, 47, 170

NFQ Neural Fitted Q Iteration. 51

PCA principal component analysis. 138, 142, 143

PER prioritized experience replay. 52

PETS probabilistic ensemble with trajectory sampling. 221

PI probability of improvement. 131

PI^2 Policy Improvements with Path Integrals. 48

PID proportional-integral-derivative. 29, 205

PILCO Probabilistic Inference for Learning Control. 220, 221

PoWER Policy Learning by Weighting Exploration with the Returns. 45–48

PPO Proximal Policy Optimization. 58, 62–64, 221

ProMP Probabilistic movement primitives. 35, 138, 170

PUBSVE positive upper boundary support vector estimation. 21, 115–117, 121, 150, 182

ranking SVM ranking support vector machine. 123, 124, 126, 149

RBF radial basis function. 71, 74, 115, 119, 124, 133, 227

REPS Relative Entropy Policy Search. 46, 47, 50, 56, 63, 93, 120, 158, 164, 165, 170

RL reinforcement learning. 40, 49–51, 59, 63, 66, 94, 148, 154, 155, 164, 168, 178, 179, 181, 183, 219, 220

RWR Reward-Weighted Regression. 45, 46, 49, 56, 68, 72, 225, 227

SAC Soft Actor-Critic. 61–63, 221

SAGG-RIAC self-adaptive goal generation—robust intelligent adaptive curiosity. 103, 108, 230–232, 236

SARSA State Action Reward State Action. 42

SDMP Stylistic Dynamic Movement Primitives. 35, 138

SEDS Stable Estimator of Dynamical Systems. 36, 37

SVD singular value decomposition. 138

SVR support vector regression. 116, 117

TD temporal-difference. 41–43

TD3 Twin Delayed Deep DPG. 55, 62

TP-GMM Task-Parameterized Gaussian Mixture Model. 36

TPU Tensor Processing Unit. 60

TRPO Trust Region Policy Optimization. 56–59, 63, 64, 219, 221

UCB Upper Confidence Bound. 39, 105, 106

VAE variational autoencoder. 137–140, 142, 143, 181

VIP Variational Inference for Policy Search. 46, 47, 49, 68, 73, 74

vMCI velocity-based Multiple Change-point Inference. 156

VTAE Variational Trajectory Autoencoder. 137, 141–144, 148, 149, 179

ZMP zero moment point. 186

List of Figures

List of Tables

Bibliography

[Aba+15] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015. URL: https://www.tensorflow.org (visited on 05/28/2020).

[Abd+15] Abbas Abdolmaleki, Rudolf Lioutikov, Jan Peters, Nuno Lau, Luís Paulo Reis, and Gerhard Neumann. "Model-Based Relative Entropy Stochastic Search". In: *Advances in Neural Information Processing Systems*. Ed. by C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett. Curran Associates, Inc., 2015, pp. 3537–3545. URL: http://papers.nips.cc/paper/5672-model-based-relative-entropy-stochastic-search.pdf.

[Abd+17a] Abbas Abdolmaleki, Bob Price, Nuno Lau, Luís Paulo Reis, and Gerhard Neumann. "Contextual Covariance Matrix Adaptation Evolutionary Strategies". In: *International Joint Conference on Artificial Intelligence (IJCAI)*. Ed. by Carles Sierra. 2017, pp. 1378–1385. DOI: 10.24963/ijcai.2017/191.

[Abd+17b] Abbas Abdolmaleki, David Simões, Nuno Lau, Luís Paulo Reis, and Gerhard Neumann. "Learning a Humanoid Kick with Controlled Distance". In: *RoboCup: Robot Soccer World Cup*. Ed. by Sven Behnke, Raymond Sheh, Sanem Sariel, and Daniel D. Lee. Springer, 2017, pp. 45–57. ISBN: 978-3-319-68792-6.

[Abd+19] Abbas Abdolmaleki, David Simões, Nuno Lau, Luís Paulo Reis, and Gerhard Neumann. "Contextual Direct Policy Search with Regularized Covariance Matrix Estimation". In: *Journal of Intelligent and Robotic Systems* 96 (2 2019), pp. 141–157. DOI: 10.1007/s10846-018-0968-4.

[Abe+19] Kfir Aberman, Rundi Wu, Dani Lischinski, Baoquan Chen, and Daniel Cohen-Or. "Learning Character-Agnostic Motion for Motion Retargeting in 2D". In: *ACM Transactions on Graphics* 38.4 (2019). ISSN: 0730-0301. DOI: 10.1145/3306346.3322999.

[ACC14] Tohid Alizadeh, Sylvain Calinon, and Darwin G. Caldwell. "Learning from demonstrations with partially observable task parameters". In: *IEEE International Conference on Robotics and Automation (ICRA)*. Ed. by Ning Xi and William R. Hamel. 2014, pp. 3309–3314. DOI: 10.1109/ICRA.2014.6907335.

[Adi+08] Esther Adi-Japha, Avi Karni, Ariel Parnes, Iris Loewenschuss, and Eli Vakil. "A Shift in Task Routines During the Learning of a Motor Skill: Group-Averaged Data May Mask Critical Phases in the Individuals' Acquisition of Skilled Performance". In: *Journal of Experimental Psychology: Learning, Memory, and Cognition* 24 (2008), pp. 1544–1551.

[Agr95] Rajeev Agrawal. "Sample Mean Based Index Policies with O(log n) Regret for the Multi-Armed Bandit Problem". English. In: *Advances in Applied Probability* 27.4 (1995), pp. 1054–1078.

[Agu+16] Jeffrey Aguilar, Tingnan Zhang, Feifei Qian, Mark Kingsbury, Benjamin McInroe, Nicole Mazouchova, Chen Li, Ryan Maladen, Chaohui Gong, Matt Travers, Ross L Hatton, Howie Choset, Paul B Umbanhowar, and Daniel I Goldman. "A review on locomotion robophysics: the study of movement at the intersection of robotics, soft matter and dynamical systems". In: *Reports on Progress in Physics* 79.11 (2016). DOI: 10.1088/0034-4885/79/11/110001.

[Aki+10] Youhei Akimoto, Yuichi Nagata, Isao Ono, and Shigenobu Kobayashi. "Bidirectional Relation between CMA Evolution Strategies and Natural Evolution Strategies". In: *Parallel Problem Solving from Nature (PPSN)*. Ed. by Robert Schaefer, Carlos Cotta, Joanna Kołodziej, and Günter Rudolph. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 154–163. ISBN: 978-3-642-15844-5.

[Ale+18] Alexander Alemi, Ben Poole, Ian Fischer, Joshua Dillon, Rif A. Saurous, and Kevin Murphy. "Fixing a Broken ELBO". In: *International Conference on Machine Learning (ICML)*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, 2018, pp. 159–168.

[Amo+12] Heni Ben Amor, Oliver Kroemer, Ulrich Hillenbrand, Gerhard Neumann, and Jan Peters. "Generalization of human grasping for multi-fingered robot hands". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Ed. by Anibal T. de Almeida, Urbano Nunes, and Eugenio Guglielmelli. 2012, pp. 2043–2050. ISBN: 978-1-4673-1737-5.

[Amo+14] Heni Ben Amor, Gerhard Neumann, Sanket Kamthe, Oliver Kroemer, and Jan Peters. "Interaction primitives for human-robot cooperation tasks". In: *IEEE International Conference on Robotics and Automation (ICRA)*. Ed. by Ning Xi and William R. Hamel. 2014, pp. 2831–2837. DOI: 10.1109/ICRA.2014.6907265.

[Amo+16] Dario Amodei, Chris Olah, Jacob Steinhardt, Paul F. Christiano, John Schulman, and Dan Mané. "Concrete Problems in AI Safety". In: *CoRR* abs/1606.06565 (2016). arXiv: 1606.06565.

[AMS97] Christopher G. Atkeson, Andrew W. Moore, and Stefan Schaal. "Locally Weighted Learning for Control". In: *Artificial Intelligence Review* 11.1 (1997), pp. 75–113. ISSN: 1573-7462. DOI: 10.1023/A:1006511328852.

[And+17] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. "Hindsight Experience Replay". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Curran Associates, Inc., 2017, pp. 5048–5058. URL: http://papers.nips.cc/paper/7090-hindsight-experience-replay.pdf.

[Arg+09] Brenna D. Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. "A survey of robot learning from demonstration". In: *Robotics and Autonomous Systems* 57.5 (2009), pp. 469–483.

[Ark98] Ronald C. Arkin. *Behavior-based Robotics*. 1st ed. Cambridge, MA, USA: MIT Press, 1998. ISBN: 0262011654.

[Aru+17] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. "Deep Reinforcement Learning: A Brief Survey". In: *IEEE Signal Processing Magazine* 34.6 (2017), pp. 26–38. ISSN: 1053-5888. DOI: 10.1109/MSP.2017.2743240.

[Asa+96] Minoru Asada, Shoichi Noda, Sukoya Tawaratsumida, and Koh Hosoda. "Purposive Behavior Acquisition for a Real Robot by Vision-Based Reinforcement Learning". In: *Machine Learning* 23.2 (1996), pp. 279–303. ISSN: 1573-0565. DOI: 10.1023/A:1018237008823.

[Ash+15] Jayen Ashar, Jaiden Ashmore, Brad Hall, Sean Harris, Bernhard Hengst, Roger Liu, Zijie Mei (Jacky), Maurice Pagnucco, Ritwik Roy, Claude Sammut, Oleg Sushkov, Belinda Teh, and Luke Tsekouras. "RoboCup SPL 2014 Champion Team Paper". In: *RoboCup: Robot Soccer World Cup*. Ed. by Reinaldo A. C. Bianchi, H. Levent Akin, Subramanian Ramamoorthy, and Komei Sugiura. Cham: Springer International Publishing, 2015, pp. 70–81. ISBN: 978-3-319-18615-3.

[BA15] Patrick Beeson and Barrett Ames. "TRAC-IK: An open-source library for improved solving of generic inverse kinematics". In: *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*. 2015, pp. 928–935.

[Bad+20] Adrià Puigdomènech Badia, Bilal Piot, Steven Kapturowski, Pablo Sprechmann, Alex Vitvitskyi, Daniel Guo, and Charles Blundell. "Agent57: Outperforming the Atari Human Benchmark". In: *CoRR* abs/2003.13350 (2020). arXiv: 2003.13350.

[Bai95] Leemon Baird. "Residual Algorithms: Reinforcement Learning with Function Approximation". In: *International Conference on Machine Learning (ICML)*. Ed. by Armand Prieditis and Stuart Russell. San Francisco (CA): Morgan Kaufmann, 1995, pp. 30–37. ISBN: 978-1-55860-377-6. DOI: 10.1016/B978-1-55860-377-6.50013-X.

[Bak+19] Bowen Baker, Ingmar Kanitscheider, Todor Markov, Yi Wu, Glenn Powell, Bob McGrew, and Igor Mordatch. "Emergent Tool Use From Multi-Agent Autocurricula". In: *CoRR* (2019). arXiv: 1909.07528 [cs.LG].

[Bar+13] Samuel Barrett, Katie Genter, Yuchen He, Todd Hester, Piyush Khandelwal, Jacob Menashe, and Peter Stone. "UT Austin Villa 2012: Standard Platform League World Champions". In: *RoboCup: Robot Soccer World Cup*. Ed. by Xiaoping Chen, Peter Stone, Luis Enrique Sucar, and Tijn Van der Zant. Lecture Notes in Artificial Intelligence. Berlin: Springer Verlag, 2013.

[Bar+16] Sebastian Bartsch, Marc Manz, Peter Kampmann, Alexander Dettmann, Hendrik Hanff, Malte Langosz, Kai von Szadkowski, Jens Hilljegerdes, Marc Simnofske, Philipp Kloss, Manuel Meder, and Frank Kirchner. "Development and Control of the Multi-Legged Robot Mantis". In: *International Symposium on Robotics (ISR)*. 2016, pp. 379–386. ISBN: 978-3-8007-4231-8.

[Bar20] Vinzenz Bargsten. *COMPI: Compliant Robot Arm*. 2020. URL: https://robotik.dfki-bremen.de/en/research/robot-systems/compi.html (visited on 05/06/2020).

[Bay+18] Atilim Gunes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. "Automatic Differentiation in Machine Learning: a Survey". In: *Journal of Machine Learning Research* 18.153 (2018), pp. 1–43. URL: http://jmlr.org/papers/v18/17-468.html.

[BB96] Steven J. Bradtke and Andrew G. Barto. "Linear Least-Squares Algorithms for Temporal Difference Learning". In: *Machine Learning* 22 (1996), pp. 33–57. DOI: 10.1007/BF00114723.

[BC12] Sébastien Bubeck and Nicolò Cesa-Bianchi. "Regret Analysis of Stochastic and Nonstochastic Multi-armed Bandit Problems". In: *Foundations and Trends in Machine Learning* 5.1 (2012), pp. 1–122.

[BCd10] Eric Brochu, Vlad M. Cora, and Nando de Freitas. "A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning". In: *CoRR* abs/1012.2599 (2010). arXiv: 1012.2599.

[BCL16] Daniel Berio, Sylvain Calinon, and Frederic Fol Leymarie. "Learning dynamic graffiti strokes with a compliant robot". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Ed. by Il Hong Suh and Dong-Soo Kwon. 2016, pp. 3981–3986. DOI: 10.1109/IROS.2016.7759586.

[Bd15] Vinzenz Bargsten and José de Gea Fernández. "COMPI: Development of a 6-DOF Compliant Robot Arm for Human-Robot Cooperation". In: *International Workshop on Human-Friendly Robotics*. Munich, Germany, 2015.

[BDM17] Marc G. Bellemare, Will Dabney, and Rémi Munos. "A Distributional Perspective on Reinforcement Learning". In: *International Conference on Machine Learning (ICML)*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. International Convention Centre, Sydney, Australia: PMLR, 2017, pp. 449–458.

[Bel+13] Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. "The Arcade Learning Environment: An Evaluation Platform for General Agents". In: *Journal of Artificial Intelligence Research* 47.1 (2013), pp. 253–279. ISSN: 1076-9757.

[Ben+09] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. "Curriculum Learning". In: *International Conference on Machine Learning (ICML)*. Ed. by Leon Bottou and Michael Littman. 2009, pp. 41–48.

[Ber+10] Jur van den Berg, Stephen Miller, Daniel Duckworth, Humphrey Hu, Andrew Wan, Xiao-Yu Fu, Ken Goldberg, and Pieter Abbeel. "Superhuman performance of surgical tasks by robots using iterative learning from human-guided demonstrations". In: *IEEE International Conference on Robotics and Automation (ICRA)*. Ed. by Wesley Snyder and Vijay Kumar. 2010, pp. 2074–2081. DOI: 10.1109/ROBOT.2010.5509621.

[Ber+12] Erik Berger, David Vogt, Christian Poenisch, Heni Ben Amor, and Bernhard Jung. "Cooperative Human-Robot Manipulation Tasks". In: *Beyond Robot Grasping - Modern Approaches for Learning Dynamic Manipulation, IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Ed. by Heni Ben Amor, Ashutosh Saxena, Oliver Kroemer, and Jan Peters. 2012. URL: https://www.ias.informatik.tu-darmstadt.de/uploads/Research/IROS2012/iros2.pdf.

[Ber+20] Quentin Berthet, Mathieu Blondel, Olivier Teboul, Marco Cuturi, Jean-Philippe Vert, and Francis Bach. "Learning with Differentiable Perturbed Optimizers". In: *CoRR* (2020). arXiv: 2002.08676 [cs.LG].

[Bez+17] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B. Shah. "Julia: A Fresh Approach to Numerical Computing". In: *SIAM Review* 59.1 (2017), pp. 65–98. DOI: 10.1137/141000671.

[BF97] Hamid Benbrahim and Judy A. Franklin. "Biped dynamic walking using reinforcement learning". In: *Robotics and Autonomous Systems* 22 (1997), pp. 283–302.

[BFB11] Oliver Birbach, Udo Frese, and Berthold Bäuml. "Realtime perception for catching a flying ball with a mobile humanoid". In: *IEEE International Conference on Robotics and Automation (ICRA)*. Ed. by Zexiang Li and Yuan Fang Zheng. 2011, pp. 5955–5962. DOI: 10.1109/ICRA.2011.5980138.

[BGK16] Vinzenz Bargsten, José de Gea Fernández, and Yohannes Kassahun. "Experimental Robot Inverse Dynamics Identification Using Classical and Machine Learning Techniques". In: *International Symposium on Robotics (ISR)*. 2016, pp. 1–6.

[Bil+08] Aude Billard, Sylvain Calinon, Rüdiger Dillmann, and Stefan Schaal. "Robot Programming by Demonstration". In: *Springer Handbook of Robotics*. Ed. by Bruno Siciliano and Oussama Khatib. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 1371–1394. ISBN: 978-3-540-30301-5. DOI: 10.1007/978-3-540-30301-5_60.

[Bis06] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006. ISBN: 0387310738.

[BKP14] Sascha Brandl, Oliver Kroemer, and Jan Peters. "Generalizing pouring actions between objects using warped parameters". In: *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*. Ed. by Carlos Balaguer and Giorgio Metta. 2014, pp. 616–621. DOI: 10.1109/HUMANOIDS.2014.7041426.

[BL17] Arne Böckmann and Tim Laue. "Kick Motions for the NAO Robot Using Dynamic Movement Primitives". In: *RoboCup: Robot Soccer World Cup*. Ed. by Sven Behnke, Raymond Sheh, Sanem Sarıel, and Daniel D. Lee. Cham: Springer International Publishing, 2017, pp. 33–44. ISBN: 978-3-319-68792-6.

[BLE07] Nicole Birdwell, Scott S. Livingston, and Itamar Elhanany. *Reinforcement learning in sonsor-guided AIBO robots*. Tech. rep. University of Tennesse, 2007.

[Blo+20] Mathieu Blondel, Olivier Teboul, Quentin Berthet, and Josip Djolonga. "Fast Differentiable Sorting and Ranking". In: *CoRR* (2020). arXiv: 2002.08871 [stat.ML].

[BM03] Andrew G. Barto and Sridhar Mahadevan. "Recent Advances in Hierarchical Reinforcement Learning". In: *Discrete Event Dynamic Systems* 13 (4 2003), pp. 341–379. DOI: 10.1023/A:1025696116075.

[BO13] Adrien Baranes and Pierre-Yves Oudeyer. "Active learning of inverse models with intrinsically motivated goal exploration in robots". In: *Robotics and Autonomous Systems* 61.1 (2013), pp. 49–73.

[Boh+14] Jeanette Bohg, Antonio Morales, Tamim Asfour, and Danica Kragic. "Data-Driven Grasp Synthesis: A Survey". In: *IEEE Transactions on Robotics* 30.2 (2014), pp. 289–309. ISSN: 1552-3098. DOI: 10.1109/TRO.2013.2289018.

[Boj+16] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. "End to End Learning for Self-Driving Cars". In: *CoRR* abs/1604.07316 (2016). arXiv: 1604.07316.

[Bon15] Bertold Bongardt. "Analytic Approaches for Design and Operation of Haptic Human-Machine Interfaces". PhD thesis. Bremen, Germany: University of Bremen, 2015.

[Bos18] Boston Dynamics. *Atlas: The World's Most Dynamic Humanoid*. 2018. URL: https://www.bostondynamics.com/atlas (visited on 10/06/2018).

[Boz+18] Asil Kaan Bozcuoglu, Gayane Kazhoyan, Yuki Furuta, Simon Stelter, Michael Beetz, Kei Okada, and Masayuki Inaba. "The Exchange of Knowledge using Cloud Robotics". In: *IEEE International Conference on Robotics and Automation (ICRA)*. Ed. by Kevin Lynch. IEEE, 2018.

[Bra+18] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, and Skye Wanderman-Milne. *JAX: composable transformations of Python+NumPy programs*. 2018. URL: http://github.com/google/jax (visited on 05/28/2020).

[Bri20] Wiebke Brinkmann. *MANTIS: Multi-legged Manipulation and Locomotion System*. 2020. URL: https://robotik.dfki-bremen.de/en/research/robot-systems/mantis/ (visited on 05/06/2020).

[Bro+16] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. "OpenAI Gym". In: *CoRR* abs/1606.01540 (2016). arXiv: arXiv:1606.01540.

[Bro70] Charles G. Broyden. "The Convergence of a Class of Double-rank Minimization Algorithms". In: *IMA Journal of Applied Mathematics* 6.1 (1970), pp. 76–90. ISSN: 0272-4960. DOI: 10.1093/imamat/6.1.76.

[Bro86] Rodney Brooks. "A robust layered control system for a mobile robot". In: *IEEE Journal on Robotics and Automation* 2.1 (1986), pp. 14–23. ISSN: 0882-4967. DOI: 10.1109/JRA.1986.1087032.

[BSC04] Andrew G. Barto, Satinder Singh, and Nuttapong Chentanez. "Intrinsically motivated learning of hierarchical collections of skills". In: *International Conference of Developmental Learning*. LaJolla, CA, USA, 2004, pp. 112–119.

[Buc+11] Jonas Buchli, Freek Stulp, Evangelos Theodorou, and Stefan Schaal. "Learning variable impedance control". In: *International Journal of Robotics Research* 30.7 (2011), pp. 820–833. DOI: 10.1177/0278364911402527.

[Byr+95a] Richard H. Byrd, Peihuang Lu, Jorge Nocedal, and Ciyou Zhu. "A Limited-Memory Algorithm for Bound Constrained Optimization". In: *SIAM Journal on Scientific Computing* 16 (1995), pp. 1190–1208.

[Byr+95b] Richard H. Byrd, Peihuang Lu, Jorge Nocedal, and Ciyou Zhu. "A Limited-Memory Algorithm for Bound Constrained Optimization". In: *SIAM Journal on Scientific Computing* 16 (1995), pp. 1190–1208.

[Cac+17] Riccardo Caccavale, Matteo Saveriano, Giuseppe Fontanelli, Fanny Ficuciello, Dongheui Lee, and Alberto Finzi. "Imitation Learning and Attentional Supervision of Dual-Arm Structured Tasks". In: *IEEE International Conference on Development and Learning and on Epigenetic Robotics (ICDL-EPIROB)*. 2017. URL: http://elib.dlr.de/113326/.

[Cac+18] Riccardo Caccavale, Matteo Saveriano, Alberto Finzi, and Dongheui Lee. "Kinesthetic teaching and attentional supervision of structured tasks in human–robot interaction". In: *Autonomous Robots* (2018). ISSN: 1573-7527. DOI: 10.1007/s10514-018-9706-9.

[Cal+15a] Berk Calli, Arjun Singh, Aaron Walsman, Siddhartha Srinivasa, Pieter Abbeel, and Aaron M. Dollar. "The YCB object and Model set: Towards common benchmarks for manipulation research". In: *International Conference on Advanced Robotics (ICAR)*. 2015, pp. 510–517. DOI: 10.1109/ICAR.2015.7251504.

[Cal+15b] Berk Calli, Aaron Walsman, Arjun Singh, Siddhartha Srinivasa, Pieter Abbeel, and Aaron M. Dollar. "Benchmarking in Manipulation Research: Using the Yale-CMU-Berkeley Object and Model Set". In: *IEEE Robotics Automation Magazine* 22.3 (2015), pp. 36–52. ISSN: 1070-9932. DOI: 10.1109/MRA.2015.2448951.

[Cal+16] Roberto Calandra, André Seyfarth, Jan Peters, and Marc Peter Deisenroth. "Bayesian optimization for learning gaits under uncertainty". In: *Annals of Mathematics and Artificial Intelligence* 76.1 (2016), pp. 5–23. ISSN: 1573-7470. DOI: 10.1007/s10472-015-9463-9.

[Cal+17] Berk Calli, Arjun Singh, James Bruce, Aaron Walsman, Kurt Konolige, Siddhartha Srinivasa, Pieter Abbeel, and Aaron M. Dollar. "Yale-CMU-Berkeley dataset for robotic manipulation research". In: *International Journal of Robotics Research* 36.3 (2017), pp. 261–268. ISSN: 0278-3649. DOI: 10.1177/0278364917700714.

[Cal16] Sylvain Calinon. "A tutorial on task-parameterized movement learning and retrieval". In: *Intelligent Service Robotics* 9.1 (2016), pp. 1–29. ISSN: 1861-2784. DOI: 10.1007/s11370-015-0187-9.

[Can+18] Gerard Canal, Emmanuel Pignat, Guillem Alenya, Sylvain Calinon, and Carme Torras. "Joining high-level symbolic planning with low-level motion primitives in adaptive HRI: application to dressing assistance". In: *IEEE International Conference on Robotics and Automation (ICRA)*. Ed. by Kevin Lynch. 2018.

[Car+12] Arnau Carrera, Seyed Reza Ahmadzadeh, Arash Ajoudani, Petar Kormushev, Marc Carreras, and Darwin G. Caldwell. "Towards Autonomous Robotic Valve Turning". In: *Cybernetics and Information Technologies* 12.3 (2012), pp. 17–26. URL: http://kormushev.com/papers/Carrera_CIT-2012.pdf.

[Car+20] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. "End-to-End Object Detection with Transformers". In: *CoRR* (2020). arXiv: 2005.12872 [cs.CV].

[Cas+18] Pablo Samuel Castro, Subhodeep Moitra, Carles Gelada, Saurabh Kumar, and Marc G. Bellemare. "Dopamine: A Research Framework for Deep Reinforcement Learning". In: *CoRR* abs/1812.06110 (2018). arXiv: 1812.06110.

[CD88] William S. Cleveland and Susan J. Devlin. "Locally Weighted Regression: An Approach to Regression Analysis by Local Fitting". In: *Journal of the American Statistical Association* 83.403 (1988), pp. 596–610.

[Cha+19] Ian Char, Youngseog Chung, Willie Neiswanger, Kirthevasan Kandasamy, Andrew Oakleigh Nelson, Mark Boyer, Egemen Kolemen, and Jeff Schneider. "Offline Contextual Bayesian Optimization". In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett. Curran Associates, Inc., 2019, pp. 4627–4638.

[Che+15a] Chenyi Chen, Ari Seff, Alain Kornhauser, and Jianxiong Xiao. "DeepDriving: Learning Affordance for Direct Perception in Autonomous Driving". In: *IEEE International Conference on Computer Vision (ICCV)*. Ed. by Katsushi Ikeuchi, Christoph Schnörr, Josef Sivic, and Rene Vidal. IEEE, 2015, pp. 2722–2730. ISBN: 978-1-4673-8391-2. DOI: 10.1109/ICCV.2015.312.

[Che+15b] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. "MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems". In: *CoRR* abs/1512.01274 (2015). URL: http://arxiv.org/abs/1512.01274.

[Che+17a] Yevgen Chebotar, Karol Hausman, Marvin Zhang, Gaurav Sukhatme, Stefan Schaal, and Sergey Levine. "Combining Model-Based and Model-Free Updates for Trajectory-Centric Reinforcement Learning". In: *International Conference on Machine Learning (ICML)*. Ed. by Doina Precup and Yee Whye Teh. 2017.

[Che+17b] Yevgen Chebotar, Mrinal Kalakrishnan, Ali Yahya, Adrian Li, Stefan Schaal, and Sergey Levine. "Path Integral Guided Policy Search". In: *IEEE International Conference on Robotics and Automation (ICRA)*. Ed. by I-Ming Chen and Yoshihiko Nakamura. 2017, pp. 3381–3388. DOI: 10.1109/ICRA.2017.7989384.

[Chi+17] Mingshan Chi, Yufeng Yao, Yaxin Liu, Yiqian Teng, and Ming Zhong. "Learning motion primitives from demonstration". In: *Advances in Mechanical Engineering* 9.12 (2017). DOI: 10.1177/1687814017737260.

[Chu+18a] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. "Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models". In: *Advances in Neural Information Processing Systems*. Ed. by Samy Bengio, Hanna Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett. Curran Associates, Inc., 2018, pp. 4754–4765. URL: http://papers.nips.cc/paper/7725-deep-reinforcement-learning-in-a-handful-of-trials-using-probabilistic-dynamics-models.pdf.

[Chu+18b] Tzu-Kuan Chuang, Ni-Ching Lin, J. Jean Chen, Chen-Hao Hung, Yi-Wei Huang, C. H. Teng, Haikun Huang, Lap-Fai Yu, Laura Giarré, and Hsueh-Cheng Wang. "Deep Trail-Following Robotic Guide Dog in Pedestrian Environments for People Who Are Blind and Visually Impaired - Learning from Virtual and Real Worlds". In: *IEEE International Conference on Robotics and Automation (ICRA)*. Ed. by Kevin Lynch. IEEE, 2018, pp. 5849–5855.

[CLH18] Patryk Chrabaszcz, Ilya Loshchilov, and Frank Hutter. "Back to Basics: Benchmarking Canonical Evolution Strategies for Playing Atari". In: *International Joint Conference on Artificial Intelligence (IJCAI)*. Ed. by Jérôme Lang. 2018, pp. 1419–1426. DOI: 10.24963/ijcai.2018/197.

[CLK11] Shengyong Chen, Youfu Li, and Ngai Ming Kwok. "Active vision in robotic systems: A survey of recent developments". In: *International Journal of Robotics Research* 30.11 (2011), pp. 1343–1377. ISSN: 0278-3649. DOI: 10.1177/0278364911410755.

[Coc+06] Alexandru Cocora, Kristian Kersting, Christian Plagemann, Wolfram Burgard, and Luc De Raedt. "Learning Relational Navigation Policies". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Ed. by Yunhui Liu and Ning Xi. 2006, pp. 2792–2797. DOI: 10.1109/IROS.2006.282061.

[Cor+18] Enric Corona, Guillem Alenyà, Antonio Gabas, and Carme Torras. "Active garment recognition and target grasping point detection using deep learning". In: *Pattern Recognition* 74 (2018), pp. 629–641. ISSN: 0031-3203. DOI: 10.1016/j.patcog.2017.09.042.

[Cou15] Erwin Coumans. "Bullet Physics Simulation". In: *International Conference on Computer Graphics and Interactive Techniques Courses*. Ed. by Glenn Goldman. SIGGRAPH '15. Los Angeles, California: Association for Computing Machinery, 2015. ISBN: 978-1-4503-3634-5. DOI: 10.1145/2776880.2792704.

[CP07] Karla Conn and Richard Alan Peters. “Reinforcement Learning with a Supervisor for a Mobile Robot in a Real-world Environment”. In: *International Symposium on Computational Intelligence in Robotics and Automation*. 2007, pp. 73–78. DOI: 10.1109/CIRA.2007.382878.

[CSO18] Cédric Colas, Olivier Sigaud, and Pierre-Yves Oudeyer. “How Many Random Seeds? Statistical Power Analysis in Deep Reinforcement Learning Experiments”. In: *CoRR* abs/1806.08295 (2018). arXiv: 1806.08295.

[CSZ06] Olivier Chapelle, Bernhard Schölkopf, and Alexander Zien. *Semi-Supervised Learning*. MIT Press, 2006. ISBN: 978-0-262-03358-9.

[CT18] Adrià Colomé and Carme Torras. “Dimensionality Reduction for Dynamic Movement Primitives and Application to Bimanual Manipulation of Clothes”. In: *IEEE Transactions on Robotics* 34.3 (2018), pp. 602–615. ISSN: 1552-3098. DOI: 10.1109/TRO.2018.2808924.

[Cul+15] Antoine Cully, Jeff Clune, Danesh Tarapore, and Jean-Baptiste Mouret. “Robots that can adapt like animals”. In: *Nature* 521.7553 (2015), pp. 503–507. DOI: 10.1038/nature14422.

[da +14] Bruno Castro da Silva, Gianluca Baldassarre, George Konidaris, and Andrew G. Barto. “Learning parameterized motor skills on a humanoid robot”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. Ed. by Ning Xi and William R. Hamel. 2014, pp. 5239–5244. DOI: 10.1109/ICRA.2014.6907629.

[Dan+13] Christian Daniel, Gerhard Neumann, Oliver Kroemer, and Jan Peters. “Learning sequential motor tasks”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. Ed. by Lynne E. Parker. 2013, pp. 2626–2632. DOI: 10.1109/ICRA.2013.6630937.

[Dan+15] Christian Daniel, Oliver Kroemer, Malte Viering, Jan Metz, and Jan Peters. “Active Reward Learning with a Novel Acquisition Function”. In: *Autonomous Robots* 39.3 (2015), pp. 389–405.

[de +17] José de Gea Fernández, Dennis Mronga, Martin Günther, Tobias Knobloch, Malte Wirkus, Martin Schröer, Mathias Trampler, Stefan Stiene, Elsa Kirchner, Vinzenz Bargsten, Timo Bänziger, Johannes Teiwes, Thomas Krüger, and Frank **Kirchner**. “Multimodal sensor-based whole-body control for human-robot collaboration in industrial settings”. In: *Robotics and Autonomous Systems* 94 (2017), pp. 102–119. ISSN: 0921-8890. DOI: 10.1016/j.robot.2017.04.007.

[Deg+19] Jonas Degrave, Michiel Hermans, Joni Dambre, and Francis wyffels. “A Differentiable Physics Engine for Deep Learning in Robotics”. In: *Frontiers in Neurorobotics* 13 (2019), p. 6. ISSN: 1662-5218. DOI: 10.3389/fnbot.2019.00006.

[Del20] Brian Delhaisse. *PyRoboLearn: a Python framework for Robot Learning*. 2020. URL: https://github.com/robotlearn/pyrobolearn (visited on 06/05/2020).

[Den+09] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. "ImageNet: A Large-Scale Hierarchical Image Database". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Ed. by Irfan Essa, Sin Bing Kang, and Marc Pollefeys. 2009, pp. 248–255.

[Det+14] Alexander Dettmann, Malte Langosz, Kai Alexander von Szadkowski, and Sebastian Bartsch. "Towards Lifelong Learning of Optimal Control for Kinematically Complex Robots". In: *Workshop on Modelling, Estimation, Perception and Control of All Terrain Mobile Robots, IEEE International Conference on Robotics and Automation (ICRA)*. Ed. by Philippe Martinet, Kasuya Yoshida, and Marcel Bergerman. Hong Kong, China: IEEE, 2014. URL: http://wmepc14.irccyn.ec-nantes.fr/material/paper/paper-Dettmann.pdf.

[Dev+19] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Ed. by Jill Burstein, Christy Doran, and Thamar Solorio. Minneapolis, Minnesota: Association for Computational Linguistics, 2019, pp. 4171–4186. DOI: 10.18653/v1/N19-1423.

[DFK16a] DFKI GmbH, Robotics Innovation Center. *BESMAN — Dritte Demonstration — Robotersystem MANTIS*. 2016. URL: https://robotik.dfki-bremen.de/de/mediathek/videoarchiv/besman-dritte-demo.html (visited on 05/28/2020).

[DFK16b] DFKI GmbH, Robotics Innovation Center. *BESMAN — Zweite Demonstration — KUKA LBR iiwa*. 2016. URL: https://robotik.dfki-bremen.de/de/mediathek/videoarchiv/besman-zweite-demo.html (visited on 05/28/2020).

[DFK16c] DFKI GmbH, Robotics Innovation Center. *BesMan: Behaviors for Mobile Manipulation*. 2016. URL: https://robotik.dfki-bremen.de/de/forschung/projekte/besman-1.html (visited on 05/28/2020).

[DFK16d] DFKI GmbH, Robotics Innovation Center. *LIMES: Lernen intelligenter Bewegungen kinematisch komplexer Laufroboter für die Exploration im Weltraum*. 2016. URL: https://robotik.dfki-bremen.de/de/forschung/projekte/limes.html (visited on 05/28/2020).

[DFR15] Marc Peter Deisenroth, Dieter Fox, and Carl Edward Rasmussen. "Gaussian Processes for Data-Efficient Learning in Robotics and Control". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 37.2 (2015), pp. 408–423. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2013.218.

[DGK15] R. Key Dismukes, Timothy E. Goldsmith, and Janeen A. Kochan. *Effects of Acute Stress on Aircrew Performance: Literature Review and Analysis of Operational Aspects*. Tech. rep. TM-2015-218930. Moffett Field, CA: NASA Ames Research Center, 2015.

[DH97] Peter Dayan and Geoffrey E. Hinton. "Using Expectation-Maximization for Reinforcement Learning". In: *Neural Computation* 9.2 (1997), pp. 271–278. DOI: 10.1162/neco.1997.9.2.271.

[Dha+17] Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov. *OpenAI Baselines*. 2017. URL: https://github.com/openai/baselines (visited on 05/07/2020).

[dKB14] Bruno Castro da Silva, George Konidaris, and Andrew G. Barto. "Active Learning of Parameterized Skills". In: *International Conference on Machine Learning (ICML)*. Ed. by Eric P. Xing and Tony Jebara. Vol. 32. Proceedings of Machine Learning Research 2. Bejing, China: PMLR, 2014, pp. 1737–1745. URL: http://proceedings.mlr.press/v32/silva14.html.

[DNP12a] Christian Daniel, Gerhard Neumann, and Jan Peters. "Hierarchical Relative Entropy Policy Search". In: *International Conference on Artificial Intelligence and Statistics (AISTATS)*. 2012, pp. 273–281.

[DNP12b] Christian Daniel, Gerhard Neumann, and Jan Peters. "Learning Concurrent Motor Skills in Versatile Solution Spaces". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Ed. by Anibal T. de Almeida, Urbano Nunes, and Eugenio Guglielmelli. 2012, pp. 3591–3597.

[DNP13] Marc Peter Deisenroth, Gerhard Neumann, and Jan Peters. "A Survey on Policy Search for Robotics". In: *Foundations and Trends in Robotics* 2.1–2 (2013), pp. 328–373.

[Doe+17] Andreas Doerr, Duy Nguyen-Tuong, Alonso Marco, Stefan Schaal, and Sebastian Trimpe. "Model-Based Policy Search for Automatic Tuning of Multivariate PID Controllers". In: *IEEE International Conference on Robotics and Automation (ICRA)*. Ed. by I-Ming Chen and Yoshihiko Nakamura. 2017, pp. 5295–5301.

[DR11] Marc Peter Deisenroth and Carl Edward Rasmussen. "PILCO: A Model-Based and Data-Efficient Approach to Policy Search". In: *International Conference on Machine Learning (ICML)*. Ed. by Lise Getoor and Tobias Scheffer. Omnipress, 2011, pp. 465–472.

[DRF11] Marc Peter Deisenroth, Carl Edward Rasmussen, and Dieter Fox. "Learning to Control a Low-Cost Manipulator using Data-Efficient Reinforcement Learning". In: *Robotics: Science and Systems (RSS)*. Ed. by Hugh Durrant-Whyte, Nicholas Roy, and Pieter Abbeel. 2011. DOI: 10.15607/RSS.2011.VII.008.

[Dru+97] Harris Drucker, Christopher J. C. Burges, Linda Kaufman, Alex J. Smola, and Vladimir Vapnik. "Support Vector Regression Machines". In: *Advances in Neural Information Processing Systems*. Ed. by M. C. Mozer, M. I. Jordan, and T. Petsche. MIT Press, 1997, pp. 155–161. URL: http://papers.nips.cc/paper/1238-support-vector-regression-machines.pdf.

[dSB03] Andrea d'Avella, Philippe Saltiel, and Emilio Bizzi. "Combinations of muscle synergies in the construction of a natural motor behavior". In: *Nature Neuroscience* 6.3 (2003), pp. 300–308. DOI: 10.1038/nn1010.

[Dua+16] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. "Benchmarking Deep Reinforcement Learning for Continuous Control". In: *International Conference on Machine Learning (ICML)*. Ed. by Maria Florina Balcan and Kilian Q. Weinberger. Vol. 48. Proceedings of Machine Learning Research. New York, New York, USA: PMLR, 2016, pp. 1329–1338.

[Dua+17] Yan Duan, Marcin Andrychowicz, Bradly Stadie, Jonathan Ho, Jonas Schneider, Ilya Sutskever, Pieter Abbeel, and Wojciech Zaremba. "One-Shot Imitation Learning". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Curran Associates, Inc., 2017, pp. 1087–1098. URL: http://papers.nips.cc/paper/6709-one-shot-imitation-learning.pdf.

[DWS12] Thomas Degris, Martha White, and Richard S. Sutton. "Off-Policy Actor-Critic". In: *International Conference on Machine Learning (ICML)*. Ed. by John Langford and Joelle Pineau. 2012. URL: http://arxiv.org/abs/1205.4839.

[Ell+12] Lars-Peter Ellekilde, Bojan Nemec, Danny Liljekrans, Thiusius Rajeeth Savarimuthu, Dirk Kraft, Fares J. Abu-Dakka, Aleš Ude, and Norbert Krüger. "Robust peg-in-hole manipulation motivated by a human teleoperating strategy". In: *Beyond Robot Grasping - Modern Approaches for Learning Dynamic Manipulation, IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Ed. by Heni Ben Amor, Ashutosh Saxena, Oliver Kroemer, and Jan Peters. 2012. URL: https://www.ias.informatik.tu-darmstadt.de/uploads/Research/IROS2012/iros14.pdf.

[Eng+20a] Jesse Engel, Lamtharn (Hanoi) Hantrakul, Chenjie Gu, and Adam Roberts. "DDSP: Differentiable Digital Signal Processing". In: *International Conference on Learning Representations (ICLR)*. Ed. by Dawn Song, Kyunghyun Cho, and Martha White. 2020. URL: https://openreview.net/forum?id=B1x1ma4tDr.

[Eng+20b] Logan Engstrom, Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Firdaus Janoos, Larry Rudolph, and Aleksander Madry. "Implementation Matters in Deep RL: A Case Study on PPO and TRPO". In: *International Conference on Learning Representations (ICLR)*. Ed. by Dawn Song, Kyunghyun Cho, and Martha White. 2020. URL: https://openreview.net/forum?id=r1etN1rtPB.

[Eri+18] Zackory Erickson, Henry M. Clever, Greg Turk, C. Karen Liu, and Charles C. Kemp. "Deep Haptic Model Predictive Control for Robot-Assisted Dressing". In: *IEEE International Conference on Robotics and Automation (ICRA)*. Ed. by Kevin Lynch. IEEE, 2018, pp. 4437–4444.

[Esp+18] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. "IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures". In: *International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. Stockholmsmässan, Stockholm Sweden: PMLR, 2018, pp. 1407–1416.

[ET18] Peter Englert and Marc Toussaint. "Learning manipulation skills from a single demonstration". In: *International Journal of Robotics Research* 37.1 (2018), pp. 137–154. DOI: 10.1177/0278364917743795.

[Evr+09] P. Evrard, E. Gribovskaya, S. Calinon, A. Billard, and A. Kheddar. "Teaching physical collaborative tasks: object-lifting case study with a humanoid". In: *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*. 2009, pp. 399–404. DOI: 10.1109/ICHR.2009.5379513.

[Ewe+15] Marco Ewerton, Gerhard Neumann, Rudolf Lioutikov, Heni Ben Amor, Jan Peters, and Guilherme Maeda. "Learning multiple collaborative tasks with a mixture of Interaction Primitives". In: *IEEE International Conference on Robotics and Automation (ICRA)*. Ed. by Lynne Parker and Nancy Amato. 2015, pp. 1535–1542. DOI: 10.1109/ICRA.2015.7139393.

[Fab+13] Alexander Fabisch, Yohannes Kassahun, Hendrik Wöhrle, and Frank Kirchner. "Learning in compressed space". In: *Neural Networks* 42 (2013), pp. 83–93. ISSN: 0893-6080. DOI: 10.1016/j.neunet.2013.01.020.

[Fab+15] Alexander Fabisch, Jan Hendrik Metzen, Mario Michael Krell, and Frank Kirchner. "Accounting for Task-Difficulty in Active Multi-Task Robot Control Learning". In: *KI – Künstliche Intelligenz* 29.4 (2015), pp. 369–377. ISSN: 1610-1987. DOI: 10.1007/s13218-015-0363-2.

[Fab+20] Alexander Fabisch, Christoph Petzoldt, Marc Otto, and Frank Kirchner. "A Survey of Behavior Learning Applications in Robotics—State of the Art and Perspectives". In: *International Journal of Robotics Research* (2020). Submitted.

[Fab19a] Alexander Fabisch. "Empirical Evaluation of Contextual Policy Search with a Comparison-based Surrogate Model and Active Covariance Matrix Adaptation". In: *Genetic and Evolutionary Computation Conference Companion*. Ed. by Manuel López-Ibáñez. GECCO '19. ACM, 2019, pp. 251–252. ISBN: 978-1-4503-6748-6. DOI: 10.1145/3319619.3321935.

[Fab19b] Alexander Fabisch. "pytransform3d: 3D Transformations for Python". In: *Journal of Open Source Software* 4.33 (2019), p. 1159. DOI: 10.21105/joss.01159.

[Fab20] Alexander Fabisch. "A Comparison of Policy Search in Joint Space and Cartesian Space for Refinement of Skills". In: *Advances in Service and Industrial Robotics*. Ed. by Karsten Berns and Daniel Görges. Springer, 2020, pp. 301–309. ISBN: 978-3-030-19648-6. DOI: 10.1007/978-3-030-19648-6_35.

[Fai+10] Aldo Faisal, Dietrich Stout, Jan Apel, and Bruce Bradley. "The Manipulative Complexity of Lower Paleolithic Stone Toolmaking". In: *PLOS ONE* 5.11 (2010), pp. 1–11. DOI: 10.1371/journal.pone.0013718.

[Fal+15] Maurice Fallon, Scott Kuindersma, Sisir Karumanchi, Matthew Antone, Toby Schneider, Hongkai Dai, Claudia Pérez D'Arpino, Robin Deits, Matt DiCicco, Dehann Fourie, Twan Koolen, Pat Marion, Michael Posa, Andrés Valenzuela, Kuan-Ting Yu, Julie Shah, Karl Iagnemma, Russ Tedrake, and Seth Teller. "An Architecture for Online Affordance-based Perception and Whole-body Planning". In: *Journal of Field Robotics* 32.2 (2015), pp. 229–254. ISSN: 1556-4959. DOI: 10.1002/rob.21546.

[FAL17] Chelsea Finn, Pieter Abbeel, and Sergey Levine. "Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks". In: *International Conference on Machine Learning (ICML)*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. International Convention Centre, Sydney, Australia: PMLR, 2017, pp. 1126–1135.

[FFC18] Fanny Ficuciello, Pietro Falco, and Sylvain Calinon. "A Brief Survey on the Role of Dimensionality Reduction in Manipulation Learning and Control". In: *IEEE Robotics and Automation Letters (RA-L)* 3.3 (2018), pp. 2608–2615. DOI: 10.1109/LRA.2018.2818933.

[FG13] Hervé Frezza-Buet and Matthieu Geist. "A C++ Template-based Reinforcement Learning Library: Fitting the Code to the Mathematics". In: *Journal of Machine Learning Research* 14.1 (2013), pp. 625–628. URL: http://www.jmlr.org/papers/v14/frezza-buet13a.html.

[FHM18] Scott Fujimoto, Herke van Hoof, and David Meger. "Addressing Function Approximation Error in Actor-Critic Methods". In: *International Conference on Machine Learning (ICML)*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. Stockholmsmässan, Stockholm Sweden: PMLR, 2018, pp. 1587–1596.

[Fin+17a] Chelsea Finn, Tianhe Yu, Tianhao Zhang, Pieter Abbeel, and Sergey Levine. "One-Shot Visual Imitation Learning via Meta-Learning". In: *Conference on Robot Learning (CoRL)*. Ed. by Sergey Levine, Vincent Vanhoucke, and Ken Goldberg. Vol. 78. Proceedings of Machine Learning Research. PMLR, 2017, pp. 357–368. URL: http://proceedings.mlr.press/v78/finn17a.html.

[Fin+17b] Chelsea Finn, Tianhe Yu, Tianhao Zhang, Pieter Abbeel, and Sergey Levine. "One-Shot Visual Imitation Learning via Meta-Learning". In: *Conference on Robot Learning*. Ed. by Sergey Levine, Vincent Vanhoucke, and Ken Goldberg. Vol. 78. Proceedings of Machine Learning Research. PMLR, 2017, pp. 357–368.

[Fin+19] Steffen Finck, Nikolaus Hansen, Raymond Ros, and Anne Auger. *Real-Parameter Black-Box Optimization Benchmarking 2010: Presentation of the Noiseless Functions*. Tech. rep. 2009/20. Research Center PPE, 2019. URL: https://coco.gforge.inria.fr/downloads/download16.00/bbobdocfunctions.pdf.

[FK20] Alexander Fabisch and Frank Kirchner. "Variational Trajectory Autoencoder for Sample-Efficient Policy Search". In: *Conference on Robot Learning (CoRL)*. Submitted. 2020.

[Fle70] Roger Fletcher. "A new approach to variable metric algorithms". In: *The Computer Journal* 13.3 (1970), pp. 317–322. ISSN: 0010-4620. DOI: 10.1093/comjnl/13.3.317.

[FLK20] Alexander Fabisch, Malte Langosz, and Frank Kirchner. "BOLeRo: Behavior Optimization and Learning for Robots". In: *International Journal of Advanced Robotic Systems* 17 (3 2020). DOI: 10.1177/1729881420913741.

[Flo+18] Carlos Florensa, David Held, Xinyang Geng, and Pieter Abbeel. "Automatic Goal Generation for Reinforcement Learning Agents". In: *International Conference on Machine Learning (ICML)*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. Stockholmsmässan, Stockholm Sweden: PMLR, 2018, pp. 1515–1528.

[FM14] Alexander Fabisch and Jan Hendrik Metzen. "Active Contextual Policy Search". In: *Journal of Machine Learning Research* 15 (2014), pp. 3371–3399. URL: http://jmlr.org/papers/v15/fabisch14a.html.

[FO16] Sébastien Forestier and Pierre-Yves Oudeyer. "Modular active curiosity-driven discovery of tool use". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Ed. by Il Hong Suh and Dong-Soo Kwon. 2016, pp. 3965–3972.

[For+18] Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Matteo Hessel, Ian Osband, Alex Graves, Volodymyr Mnih, Remi Munos, Demis Hassabis, Olivier Pietquin, Charles Blundell, and Shane Legg. "Noisy Networks For Exploration". In: *International Conference on Learning Representations (ICLR)*. Ed. by Iain Murray, Marc'Aurelio Ranzato, and Oriol Vinyals. 2018. URL: https://openreview.net/forum?id=rywHCPkAW.

[FS04] Peggy Fidelman and Peter Stone. "Learning Ball Acquisition on a Physical Robot". In: *International Symposium on Robotics and Automation (ISRA)*. 2004. URL: http://www.cs.utexas.edu/users/ai-lab/?fidelman:isra04.

[FUB16] Nadia Figueroa, Ana Lucia Pais Ureche, and Aude Billard. "Learning complex sequential tasks from demonstration: A pizza dough rolling case study". In: *ACM/IEEE International Conference on Human-Robot Interaction (HRI)*. 2016, pp. 611–612. DOI: 10.1109/HRI.2016.7451881.

[Fuk+19] Lior Fuks, Noor Awad, Frank Hutter, and Marius Lindauer. "An Evolution Strategy with Progressive Episode Lengths for Playing Games". In: *International Joint Conference on Artificial Intelligence (IJCAI)*. Ed. by Sarit Kraus. International Joint Conferences on Artificial Intelligence Organization, 2019, pp. 1234–1240. DOI: 10.24963/ijcai.2019/172.

[Gam+10] Andrej Gams, Tadej Petrič, Leon Zandlajpah, and Aleš Ude. "Optimizing parameters of trajectory representation for movement generalization: robotic throwing". In: *International Workshop on Robotics in Alpe-Adria-Danube Region (RAAD)*. Ed. by Imre J. Rudas, József K. Tar, and Claudiu Pozna. 2010, pp. 161–166. DOI: 10.1109/RAAD.2010.5524592.

[Gam+14] Andrej Gams, Jesse van den Kieboom, Massimo Vespignani, Luc Guyot, Aleš Ude, and Auke Jan Ijspeert. "Rich periodic motor skills on humanoid robots: Riding the pedal racer". In: *IEEE International Conference on Robotics and Automation (ICRA)*. Ed. by Ning Xi and William R. Hamel. 2014, pp. 2326–2332. DOI: 10.1109/ICRA.2014.6907181.

[GB09] Gerd Gigerenzer and Henry Brighton. "Homo Heuristicus: Why Biased Minds Make Better Inferences". In: *Topics in Cognitive Science* 1.1 (2009), pp. 107–143. ISSN: 1756-8765. DOI: 10.1111/j.1756-8765.2008.01006.x.

[GB13] Caglar Gulcehre and Yoshua Bengio. "Knowledge Matters: Importance of Prior Information for Optimization". In: *International Conference on Learning Representations (ICLR)*. Ed. by Aaron Courville, Rob Fergus, and Chris Manning. 2013.

[Gea+17] José de Gea Fernández, Dennis Mronga, Martin Günther, Tobias Knobloch, Malte Wirkus, Martin Schröer, Mathias Trampler, Stefan Stiene, Elsa Kirchner, Vinzenz Bargsten, Timo Bänziger, Johannes Teiwes, Thomas Krüger, and Frank Kirchner. "Multimodal sensor-based whole-body control for human–robot collaboration in industrial settings". In: *Robotics and*

Autonomous Systems 94 (2017), pp. 102–119. ISSN: 0921-8890. DOI: 10.1016/j.robot.2017.04.007.

[Gea20] José de Gea Fernández. *Mitsubishi PA 10-7C*. 2020. URL: https://robotik.dfki-bremen.de/en/research/robot-systems/mitsubishi-pa-10-7c.html (visited on 05/06/2020).

[Geh+14] Christian Gehring, Stelian Coros, Marco Hutter, Michael Bloesch, Péter Fankhauser, Markus A. Hoepflinger, and Roland Siegwart. "Towards automatic discovery of agile gaits for quadrupedal robots". In: *IEEE International Conference on Robotics and Automation (ICRA)*. Ed. by Ning Xi and William R. Hamel. 2014, pp. 4243–4248.

[Ger+15] Alborz Geramifard, Christoph Dann, Robert H. Klein, William Dabney, and Jonathan P. How. "RLPy: A Value-Function-Based Reinforcement Learning Framework for Education and Research". In: *Journal of Machine Learning Research* 16 (2015), pp. 1573–1578. URL: http://jmlr.org/papers/v16/geramifard15a.html.

[GFB94] Vijaykumar Gullapalli, Judy A. Franklin, and Hamid Benbrahim. "Acquiring robot skills via reinforcement learning". In: *IEEE Control Systems* 14.1 (1994), pp. 13–24. ISSN: 1066-033X. DOI: 10.1109/37.257890.

[Gig08] Gerd Gigerenzer. "Why Heuristics Work". In: *Perspectives on Psychological Science* 3.1 (2008), pp. 20–29. DOI: 10.1111/j.1745-6916.2008.00058.x.

[GK16] Lisa Gutzeit and Elsa Andrea Kirchner. "Automatic Detection and Recognition of Human Movement Patterns in Manipulation Tasks". In: *International Conference on Physiological Computing Systems (PHYCS)*. 2016, pp. 54–63.

[Gle98] Michael Gleicher. "Retargetting Motion to New Characters". In: *International Conference on Computer Graphics and Interactive Techniques*. Ed. by Christopher C. Yang and T. M. Murali. SIGGRAPH '98. New York, NY, USA: ACM, 1998, pp. 33–42. ISBN: 0-89791-999-8. DOI: 10.1145/280814.280820.

[GM11] Aurelélien Garivier and Eric Moulines. "On Upper-Confidence Bound Policies for Switching Bandit Problems". In: *International Conference on Algorithmic Learning Theory*. Ed. by Jyrki Kivinen, Csaba Szepesvári, Esko Ukkonen, and Thomas Zeugmann. Springer Berlin Heidelberg, 2011, pp. 174–188. ISBN: 978-3-642-24412-4. DOI: 10.1007/978-3-642-24412-4_16.

[GMB93] Simon F. Giszter, Ferdinando A. Mussa-Ivaldi, and Emilio Bizzi. "Convergent force fields organized in the frog's spinal cord". In: *Journal of Neuroscience* 13 (1993), pp. 467–491.

[Gol70] Donald Goldfarb. "A Family of Variable Metric Updates Derived by Variational Means". In: *Mathematics of Computation* 24.109 (1970), pp. 23–26. DOI: 10.1090/S0025-5718-1970-0258249-6.

[Góm+18] Rafael Gómez-Bombarelli, Jennifer N. Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D. Hirzel, Ryan P. Adams, and Alán Aspuru-Guzik. "Automatic Chemical Design Using a Data-Driven Continuous Representation of Molecules". In: *ACS Central Science* 4 (2 2018), pp. 268–276. DOI: 10.1021/acscentsci.7b00572.

[Goo+14] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. "Generative Adversarial Nets". In: *Advances in Neural Information Processing Systems*. Ed. by Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger. Curran Associates, Inc., 2014, pp. 2672–2680. URL: http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf.

[Got+13] Jacqueline Gottlieb, Pierre-Yves Oudeyer, Manuel Lopes, and Adrien Baranes. "Information-seeking, curiosity, and attention: computational and neural mechanisms". In: *Trends in Cognitive Sciences* 17.11 (2013), pp. 585–93.

[GPG17] Dhiraj Gandhi, Lerrel Pinto, and Abhinav Gupta. "Learning to fly by crashing". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Ed. by Hong Zhang and Richard Vaughan. 2017, pp. 3948–3955.

[GPW06] Tao Geng, Bernd Porr, and Florentin Wörgötter. "Fast biped walking with a reflexive controller and real-time policy searching". In: *Advances in Neural Information Processing Systems*. Ed. by Y. Weiss, B. Schölkopf, and J. C. Platt. MIT Press, 2006, pp. 427–434. URL: http://papers.nips.cc/paper/2769-fast-biped-walking-with-a-reflexive-controller-and-real-time-policy-searching.pdf.

[Gra+16] Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, Adrià Puigdomènech Badia, Karl Moritz Hermann, Yori Zwols, Georg Ostrovski, Adam Cain, Helen King, Christopher Summerfield, Phil Blunsom, Koray Kavukcuoglu, and Demis Hassabis. "Hybrid computing using a neural network with dynamic external memory". In: *Nature* 538.7626 (2016), pp. 471–476. ISSN: 00280836. DOI: 10.1038/nature20101.

[Gra98] Ann M. Graybiel. "The basal ganglia and chunking of action repertoires". In: *Neurobiology of Learning and Memory* 70 (1 1998), pp. 119–136. ISSN: 1074-7427. DOI: 10.1006/nlme.1998.3843.

[Gre+15] Edward Grefenstette, Karl Moritz Hermann, Mustafa Suleyman, and Phil Blunsom. "Learning to Transduce with Unbounded Memory". In: *Advances in Neural Information Processing Systems*. Ed. by C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett. Curran Associates,

Inc., 2015, pp. 1828–1836. URL: http://papers.nips.cc/paper/5648-learning-to-transduce-with-unbounded-memory.pdf.

[GSB10] Kathrin Gräve, Jörg Stückler, and Sven Behnke. "Learning Motion Skills from Expert Demonstrations and Own Experience using Gaussian Process Regression". In: *International Symposium on Robotics (ISR)*. VDE Verlag, 2010, pp. 1–8.

[Gu+16] Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. "Continuous Deep Q-Learning with Model-based Acceleration". In: *International Conference on Machine Learning (ICML)*. Ed. by Maria Florina Balcan and Kilian Q. Weinberger. Vol. 48. Proceedings of Machine Learning Research. New York, New York, USA: PMLR, 2016, pp. 2829–2838.

[Gu+17] Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates". In: *IEEE International Conference on Robotics and Automation (ICRA)*. Ed. by I-Ming Chen and Yoshihiko Nakamura. 2017, pp. 3389–3396. DOI: 10.1109/ICRA.2017.7989385.

[Gua+18] Sergio Guadarrama, Anoop Korattikara, Oscar Ramirez, Pablo Castro, Ethan Holly, Sam Fishman, Ke Wang, Ekaterina Gonina, Chris Harris, Vincent Vanhoucke, and Eugene Brevdo. *TF-Agents: A library for Reinforcement Learning in TensorFlow*. 2018. URL: https://github.com/tensorflow/agents (visited on 11/30/2018).

[Gua18] Shaobo Guan. *TL-GAN: transparent latent-space GAN*. 2018. URL: https://github.com/SummitKwan/transparent_latent_gan (visited on 01/30/2020).

[Gut+18] Lisa Gutzeit, Alexander Fabisch, Marc Otto, Jan Hendrik Metzen, Jonas Hansen, Frank Kirchner, and Elsa Andrea Kirchner. "The BesMan Learning Platform for Automated Robot Skill Learning". In: *Frontiers in Robotics and AI* 5 (2018), p. 43. ISSN: 2296-9144. DOI: 10.3389/frobt.2018.00043.

[Gut+19] Lisa Gutzeit, Alexander Fabisch, Christoph Petzoldt, Hendrik Wiese, and Frank Kirchner. "Automated Robot Skill Learning from Demonstration for Various Robot Systems". In: *KI: Advances in Artificial Intelligence*. Ed. by Christoph Benzmüller and Heiner Stuckenschmidt. Springer International Publishing, 2019, pp. 168–181. ISBN: 978-3-030-30179-8. DOI: 10.1007/978-3-030-30179-8_14.

[GWD14] Alex Graves, Greg Wayne, and Ivo Danihelka. "Neural Turing Machines". In: *CoRR* abs/1410.5401 (2014). arXiv: 1410.5401.

[Ha+20] Sehoon Ha, Peng Xu, Zhenyu Tan, Sergey Levine, and Jie Tan. "Learning to Walk in the Real World with Minimal Human Effort". In: *CoRR* abs/2002.08550 (2020). arXiv: 2002.08550.

[Haa+18a] Tuomas Haarnoja, Vitchyr Pong, Aurick Zhou, Murtaza Dalal, Pieter Abbeel, and Sergey Levine. "Composable Deep Reinforcement Learning for Robotic Manipulation". In: *IEEE International Conference on Robotics and Automation (ICRA)*. Ed. by Kevin Lynch. 2018, pp. 6244–6251.

[Haa+18b] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor". In: *International Conference on Machine Learning (ICML)*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. Stockholmsmässan, Stockholm Sweden: PMLR, 2018, pp. 1861–1870.

[Haa+18c] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. "Soft Actor-Critic Algorithms and Applications". In: *CoRR* (2018). arXiv: 1812.05905.

[Haa+19] Tuomas Haarnoja, Aurick Zhou, Sehoon Ha, Jie Tan, George Tucker, and Sergey Levine. "Learning to Walk via Deep Reinforcement Learning". In: *Robotics: Science and Systems (RSS)*. Ed. by Antonio Bicchi, Hadas Kress-Gazit, and Seth Hutchinson. 2019. ISBN: 978-0-9923747-5-4.

[Had+09] Sami Haddadin, Tim Laue, Udo Frese, Sebastian Wolf, Alin Albu-Schäffer, and Gerd Hirzinger. "Kick it with elasticity: Safety and performance in human–robot soccer". In: *Robotics and Autonomous Systems* 57.8 (2009). Humanoid Soccer Robots, pp. 761–775. ISSN: 0921-8890. DOI: 10.1016/j.robot.2009.03.004.

[Han+08] Nikolaus Hansen, Raymond Ros, Nikolas Mauny, Marc Schoenauer, and Anne Auger. *PSO Facing Non-Separable and Ill-Conditioned Problems*. Research Report RR-6447. INRIA, 2008. URL: https://hal.inria.fr/inria-00250078.

[Han+10] Nikolaus Hansen, Anne Auger, Raymond Ros, Steffen Finck, and Petr Pošík. "Comparing Results of 31 Algorithms from the Black-box Optimization Benchmarking BBOB-2009". In: *Genetic and Evolutionary Computation Conference Companion*. Ed. by Jürgen Branke. GECCO '10. Portland, Oregon, USA: Association for Computing Machinery, 2010, pp. 1689–1696. ISBN: 9781450300735. DOI: 10.1145/1830761.1830790.

[Han+14] Awni Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, and Andrew Y. Ng. "Deep Speech: Scaling up end-to-end speech recognition". In: *CoRR* (2014). arXiv: 1412.5567 [cs.CL].

[Han+16] Nikolaus Hansen, Anne Auger, Olaf Mersmann, Tea Tušar, and Dimo Brockhoff. "COCO: A Platform for Comparing Continuous Optimizers in a Black-Box Setting". In: *CoRR* abs/1603.08785 (2016). arXiv: 1603.08785.

[Han15] Jonas Hansen. “Contextual Policy Search for Ball-Throwing on a Real Robot”. MA thesis. Bremen, Germany: University of Bremen, 2015.

[Han19] Nikolaus Hansen. “A Global Surrogate Assisted CMA-ES”. In: *Genetic and Evolutionary Computation Conference*. Ed. by Manuel López-Ibáñez. GECCO ’19. Prague, Czech Republic: Association for Computing Machinery, 2019, pp. 664–672. ISBN: 9781450361118. DOI: 10.1145/3321707.3321842.

[Has10] Hado V. Hasselt. “Double Q-learning”. In: *Advances in Neural Information Processing Systems*. Ed. by J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta. Curran Associates, Inc., 2010, pp. 2613–2621.

[HCM15] Assaf Hallak, Dotan Di Castro, and Shie Mannor. “Contextual Markov Decision Processes”. In: *CoRR* (2015). arXiv: 1502.02259 [stat.ML].

[Hee+17] Nicolas Heess, Dhruva TB, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, S. M. Ali Eslami, Martin A. Riedmiller, and David Silver. “Emergence of Locomotion Behaviours in Rich Environments”. In: *CoRR* abs/1707.02286 (2017). arXiv: 1707.02286.

[Hen+19] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. “Deep Reinforcement Learning that Matters”. In: *CoRR* abs/1709.06560 (2019). arXiv: 1709.06560. URL: http://arxiv.org/abs/1709.06560.

[Hen14] Alexander David Henning. *Approximate Inverse Kinematics Using a Database*. Tech. rep. Worcester Polytechnic Institute, 2014.

[Hes+18] Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Gheshlaghi Azar, and David Silver. “Rainbow: Combining Improvements in Deep Reinforcement Learning”. In: *AAAI Conference on Artificial Intelligence*. Ed. by Sheila A. McIlraith and Kilian Q. Weinberger. AAAI Press, 2018, pp. 3215–3222.

[HGS16] Hado van Hasselt, Arthur Guez, and David Silver. “Deep Reinforcement Learning with Double Q-Learning”. In: *AAAI Conference on Artificial Intelligence*. Ed. by Dale Schuurmans and Michael Wellman. Phoenix, Arizona: AAAI Press, 2016, pp. 2094–2100.

[HI08] Verena Heidrich-Meisner and Christian Igel. “Evolution Strategies for Direct Policy Search”. In: *Parallel Problem Solving from Nature (PPSN)*. Ed. by Thomas Jansen, Simon Lucas, and Carlo Poloni. 2008, pp. 428–437.

[HI09] Verena Heidrich-Meisner and Christian Igel. “Neuroevolution strategies for episodic reinforcement learning”. In: *Journal of Algorithms* 64.4 (2009). Special Issue: Reinforcement Learning, pp. 152–168. ISSN: 0196-6774. DOI: 10.1016/j.jalgor.2009.04.002.

[Hig+17] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. "β-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework". In: *International Conference on Learning Representations (ICLR)*. Ed. by Hugo Larochelle, Oriol Vinyals, and Tara Sainath. 2017.

[Hin+12] Geoffrey Hinton, Li Deng, Dong Yu, George Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara Sainath, and Brian Kingsbury. "Deep Neural Networks for Acoustic Modeling in Speech Recognition". In: *Signal Processing Magazine* (2012).

[HKY18] Sehoon Ha, Joohyung Kim, and Katsu Yamane. "Automated Deep Reinforcement Learning Environment for Hardware of a Modular Legged Robot". In: *International Conference on Ubiquitous Robots (UR)*. Ed. by Frank C. Park and Paul Oh. 2018, pp. 348–354. DOI: 10.1109/URAI.2018.8442201.

[HO01] Nikolaus Hansen and Andreas Ostermeier. "Completely Derandomized Self-Adaptation in Evolution Strategies". In: *Evolutionary Computation* 9.2 (2001), pp. 159–195. ISSN: 1063-6560. DOI: 10.1162/106365601750190398.

[Hoo+15] Herke van Hoof, Tucker Hermans, Gerhard Neumann, and Jan Peters. "Learning robot in-hand manipulation with tactile features". In: *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*. 2015, pp. 121–127. DOI: 10.1109/HUMANOIDS.2015.7363524.

[HQS10] Todd Hester, Michael Quinlan, and Peter Stone. "Generalized model learning for Reinforcement Learning on a humanoid robot". In: *IEEE International Conference on Robotics and Automation (ICRA)*. Ed. by Wesley Snyder and Vijay Kumar. 2010, pp. 2369–2374. DOI: 10.1109/ROBOT.2010.5509181.

[HRJ17] Matt Hoffman, Carlos Riquelme, and Matthew Johnson. "The β-VAE's Implicit Prior". In: *Workshop on Bayesian Deep Learning, Advances in Neural Information Processing Systems*. 2017. URL: http://bayesiandeeplearning.org/2017/papers/66.pdf.

[HRP18] Peter Henderson, Joshua Romoff, and Joelle Pineau. "Where Did My Optimum Go?: An Empirical Analysis of Gradient Descent Optimization in Policy Gradient Methods". In: *CoRR* (2018). arXiv: 1810.02525.

[HS12] Philipp Hennig and Christian J. Schuler. "Entropy Search for Information-Efficient Global Optimization". In: *Journal of Machine Learning Research* 13.57 (2012), pp. 1809–1837. URL: http://jmlr.org/papers/v13/hennig12a.html.

[HS18] David Ha and Jürgen Schmidhuber. "Recurrent World Models Facilitate Policy Evolution". In: *Advances in Neural Information Processing Systems*. Ed. by Samy Bengio, Hanna Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett. Curran Associates, Inc., 2018, pp. 2451–2463.

[Hu+20] Yuanming Hu, Luke Anderson, Tzu-Mao Li, Qi Sun, Nathan Carr, Jonathan Ragan-Kelley, and Fredo Durand. "Differentiable Programming for Physical Simulation". In: *International Conference on Learning Representations (ICLR)*. Ed. by Dawn Song, Kyunghyun Cho, and Martha White. 2020. URL: https://openreview.net/forum?id=B1eB5xSFvr.

[Hua+18] Yanlong Huang, Joao Silverio, Leonel Rozo, and Darwin G. Caldwell. "Hybrid Probabilistic Trajectory Optimization Using Null-Space Exploration". In: *IEEE International Conference on Robotics and Automation (ICRA)*. Ed. by Kevin Lynch. IEEE, 2018, pp. 7226–7232.

[Huy09] Du Q. Huynh. "Metrics for 3D Rotations: Comparison and Analysis". In: *Journal of Mathematical Imaging and Vision* 35.2 (2009), pp. 155–164. ISSN: 0924-9907.

[Hwa+19] Jemin Hwangbo, Joonho Lee, Alexey Dosovitskiy, Dario Bellicoso, Vassilios Tsounis, Vladlen Koltun, and Marco Hutter. "Learning agile and dynamic motor skills for legged robots". In: *Science Robotics* 4.26 (2019). DOI: 10.1126/scirobotics.aau5872.

[Ijs+13] Auke Jan Ijspeert, Jun Nakanishi, Peter Pastor, Heiko Hoffmann, and Stefan Schaal. "Dynamical Movement Primitives: Learning Attractor Models for Motor Behaviors". In: *Neural Computation* 25.2 (2013), pp. 328–373.

[Ily+18] Andrew Ilyas, Logan Engstrom, Shibani Santurkar, Dimitris Tsipras, Firdaus Janoos, Larry Rudolph, and Aleksander Madry. "Are Deep Policy Gradient Algorithms Truly Policy Gradient Algorithms?" In: *CoRR* (2018). arXiv: 1811.02553.

[Inn+19] Mike Innes, Alan Edelman, Keno Fischer, Christopher Rackauckas, Elliot Saba, Viral B. Shah, and Will Tebbutt. "A Differentiable Programming System to Bridge Machine Learning and Scientific Computing". In: *CoRR* abs/1907.07587 (2019). arXiv: 1907.07587.

[INS02] Auke Jan Ijspeert, Jun Nakanishi, and Stefan Schaal. "Movement imitation with nonlinear dynamical systems in humanoid robots". In: *IEEE International Conference on Robotics and Automation (ICRA)*. Ed. by William R. Hamel and Anthony A. Maciejewski. Washington (DC), USA, 2002. URL: http://www-clmc.usc.edu/publications/I/ijspeert-ICRA2002.pdf.

[Irp18] Alex Irpan. *Deep Reinforcement Learning Doesn't Work Yet*. 2018. URL: https://www.alexirpan.com/2018/02/14/rl-hard.html (visited on 10/13/2018).

[IS15] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *International Conference on Machine Learning (ICML)*. Ed. by Francis Bach and David Blei. Vol. 37. Proceedings of Machine Learning Research. PMLR, 2015, pp. 448–456.

[Isl+17] Riashat Islam, Peter Henderson, Maziar Gomrokchi, and Doina Precup. "Reproducibility of Benchmarked Deep Reinforcement Learning Tasks for Continuous Control". In: *International Conference on Machine Learning (ICML)*. Ed. by Doina Precup and Yee Whye Teh. 2017. URL: http://arxiv.org/abs/1708.04133.

[JA06] Grahame A. Jastrebski and Dirk V. Arnold. "Improving Evolution Strategies through Active Covariance Matrix Adaptation". In: *International Conference on Evolutionary Computation (CEC)*. 2006, pp. 2814–2821.

[Jac+19] Jörn-Henrik Jacobsen, Jens Behrmann, Richard S. Zemel, and Matthias Bethge. "Excessive Invariance Causes Adversarial Vulnerability". In: *International Conference on Learning Representations (ICLR)*. Ed. by Sergey Levine, Karen Livescu, and Shakir Mohamed. 2019. URL: https://openreview.net/forum?id=BkfbpsAcF7.

[Jad+19] Max Jaderberg, Wojciech M. Czarnecki, Iain Dunning†, Luke Marris, Guy Lever, Antonio Garcia Castañeda, Charles Beattie, Neil C. Rabinowitz, Ari S. Morcos, Avraham Ruderman, Nicolas Sonnerat, Tim Green, Louise Deason, Joel Z. Leibo, David Silver, Demis Hassabis, Koray Kavukcuoglu, and Thore Graepel. "Human-level performance in 3D multiplayer games with population-based reinforcement learning". In: *Science* 364 (6443 2019), pp. 859–865. DOI: 10.1126/science.aau6249.

[JHH95] Nick Jakobi, Phil Husbands, and Inman Harvey. "Noise and the reality gap: The use of simulation in evolutionary robotics". In: *Advances in Artificial Life*. Ed. by Federico Morán, Alvaro Moreno, Juan Julián Merelo, and Pablo Chacón. Berlin, Heidelberg: Springer Berlin Heidelberg, 1995, pp. 704–720.

[JLD16] Edward Johns, Stefan Leutenegger, and Andrew J. Davison. "Deep learning a grasp function for grasping under gripper pose uncertainty". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Ed. by Il Hong Suh and Dong-Soo Kwon. 2016, pp. 4461–4468. DOI: 10.1109/IROS.2016.7759657.

[Joa02] Thorsten Joachims. "Optimizing Search Engines Using Clickthrough Data". In: *International Conference on Knowledge Discovery and Data Mining (KDD)*. Ed. by David Hand, Daniel Keim, and Raymond Ng. KDD '02. Edmonton, Alberta, Canada: Association for Computing Machinery, 2002, pp. 133–142. ISBN: 158113567X. DOI: 10.1145/775047.775067.

[JPS93] Donald R. Jones, Cary D. Perttunen, and Bruce E. Stuckman. "Lipschitzian optimization without the Lipschitz constant". In: *Journal of Optimization Theory and Applications* 79.1 (1993), pp. 157–181.

[JRB18] Rico Jonschkowski, Divyam Rastogi, and Oliver Brock. "Differentiable Particle Filters: End-to-End Learning with Algorithmic Priors". In: *Robotics: Science and Systems (RSS)*. Ed. by Hadas Kress-Gazit, Siddhartha Srinivasa, Tom Howard, and Nikolay Atanasov. 2018. ISBN: 978-0-9923747-4-7. DOI: 10.15607/RSS.2018.XIV.001.

[Jul16] Arthur Juliani. *Simple Reinforcement Learning with Tensorflow Part 8: Asynchronous Actor-Critic Agents (A3C)*. 2016. URL: https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-part-8-asynchronous-actor-critic-agents-a3c-c88f72a5e9f2 (visited on 08/03/2020).

[Kah+17] Gregory Kahn, Adam Villaflor, Vitchyr Pong, Pieter Abbeel, and Sergey Levine. "Uncertainty-Aware Reinforcement Learning for Collision Avoidance". In: *CoRR* abs/1702.01182 (2017). arXiv: 1702.01182.

[Kaj+01] Shuuji Kajita, Fumio Kanehiro, Kenji Kaneko, Kazuhito Yokoi, and Hirohisa Hirukawa. "The 3D linear inverted pendulum mode: a simple modeling for a biped walking pattern generation". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Ed. by Tzyh Jong Tarn and Joel Burdick. Vol. 1. 2001, pp. 239–246. DOI: 10.1109/IROS.2001.973365.

[Kal+09] Mrinal Kalakrishnan, Jonas Buchli, Peter Pastor, and Stefan Schaal. "Learning locomotion over rough terrain using terrain templates". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Ed. by Ning Xi and Zhidong Wang. 2009, pp. 167–172. DOI: 10.1109/IROS.2009.5354701.

[Kal+11a] Mrinal Kalakrishnan, Sachin Chitta, Evangelos A. Theodorou, Peter Pastor, and Stefan Schaal. "STOMP: Stochastic trajectory optimization for motion planning". In: *IEEE International Conference on Robotics and Automation (ICRA)*. Ed. by Zexiang Li and Yuan Fang Zheng. 2011, pp. 4569–4574. DOI: 10.1109/ICRA.2011.5980280.

[Kal+11b] Mrinal Kalakrishnan, Ludovic Righetti, Peter Pastor, and Stefan Schaal. "Learning force control policies for compliant manipulation". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Ed. by Oussama Khatib and Gaurav Sukhatme. 2011, pp. 4639–4644. DOI: 10.1109/IROS.2011.6095096.

[Kal60] Rudolph Emil Kalman. "A New Approach to Linear Filtering and Prediction Problems". In: *Transactions of the ASME–Journal of Basic Engineering* 82.Series D (1960), pp. 35–45.

[KAN08] J. Zico Kolter, Pieter Abbeel, and Andrew Y. Ng. "Hierarchical Apprenticeship Learning with Application to Quadruped Locomotion". In: *Advances in Neural Information Processing Systems*. Ed. by J. C. Platt, D. Koller, Y. Singer, and S. T. Roweis. Curran Associates, Inc., 2008, pp. 769–776. URL: http://papers.nips.cc/paper/3253-hierarchical-apprenticeship-learning-with-application-to-quadruped-locomotion.pdf.

[Kar17] Andrej Karpathy. *Software 2.0*. 2017. URL: https://medium.com/@karpathy/software-2-0-a64152b37c35 (visited on 02/11/2020).

[Kas+08] Yohannes Kassahun, Jose de Gea, Mark Edgington, Jan Hendrik Metzen, and Frank Kirchner. "Accelerating Neuroevolutionary Methods Using a Kalman Filter". In: *Genetic and Evolutionary Computation Conference*. Ed. by Maarten Keijzer. GECCO '08. Atlanta, GA, USA: ACM, 2008, pp. 1397–1404. ISBN: 978-1-60558-130-9. DOI: 10.1145/1389095.1389365.

[KB11] Seyed Mohammad Khansari-Zadeh and Aude Billard. "Learning Stable Nonlinear Dynamical Systems With Gaussian Mixture Models". In: *IEEE Transactions on Robotics* 27.5 (2011), pp. 943–957.

[KB15] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *International Conference on Learning Representations (ICLR)*. Ed. by Hugo Larochelle, Oriol Vinyals, and Tara Sainath. 2015.

[KBP13] Jens Kober, J. Andrew Bagnell, and Jan Peters. "Reinforcement Learning in Robotics: A Survey". In: *International Journal of Robotics Research* 32.11 (2013), pp. 1238–1274. ISSN: 0278-3649. DOI: 10.1177/0278364913495721.

[KCC10a] Petar Kormushev, Sylvain Calinon, and Darwin G. Caldwell. "Approaches for Learning Human-like Motor Skills which Require Variable Stiffness During Execution". In: *Workshop on Humanoid Robots Learning from Human Interaction, IEEE-RAS International Conference on Humanoid Robots (Humanoids)*. Ed. by Sonia Chernova and Çetin Meriçli. Nashville, USA, 2010. URL: http://kormushev.com/papers/Kormushev_Humanoids2010_workshop.pdf.

[KCC10b] Petar Kormushev, Sylvain Calinon, and Darwin G. Caldwell. "Robot motor skill coordination with EM-based Reinforcement Learning". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Ed. by Ren C. Luo and Huei-Yung Lin. 2010, pp. 3232–3237. DOI: 10.1109/IROS.2010.5649089.

[KCC11] Petar Kormushev, Sylvain Calinon, and Darwin G. Caldwell. "Imitation Learning of Positional and Force Skills Demonstrated via Kinesthetic Teaching and Haptic Input". In: *Advanced Robotics* 25.5 (2011), pp. 581–603. URL: http://kormushev.com/papers/Kormushev_AdvancedRobotics_2011.pdf.

[KCC13] Petar Kormushev, Sylvain Calinon, and Darwin G. Caldwell. “Reinforcement Learning in Robotics: Applications and Real-World Challenges”. In: *Robotics* 2.3 (2013), pp. 122–148. ISSN: 2218-6581. DOI: 10.3390/robotics2030122.

[KD18] Durk P Kingma and Prafulla Dhariwal. “Glow: Generative Flow with Invertible 1x1 Convolutions”. In: *Advances in Neural Information Processing Systems*. Ed. by Samy Bengio, Hanna Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett. Curran Associates, Inc., 2018, pp. 10215–10224. URL: http://papers.nips.cc/paper/8224-glow-generative-flow-with-invertible-1x1-convolutions.pdf.

[KF04] Cody Kwok and Dieter Fox. “Reinforcement learning for sensing strategies”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Ed. by Kazuhiro Kosuge and Hajime Asama. Vol. 4. 2004, pp. 3158–3163. DOI: 10.1109/IROS.2004.1389903.

[KG17] Alex Kendall and Yarin Gal. “What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision?” In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Curran Associates, Inc., 2017, pp. 5574–5584.

[KGB11] Scott Kuindersma, Roderic A. Grupen, and Andrew G. Barto. “Learning dynamic arm motions for postural recovery”. In: *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*. Ed. by Aleš Ude and Nancy Pollard. 2011, pp. 7–12. DOI: 10.1109/Humanoids.2011.6100881.

[KH04] Nathan P. Koenig and Andrew Howard. “Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Ed. by Kazuhiro Kosuge and Hajime Asama. 2004, pp. 2149–2154.

[Kim+17] Su-Kyoung Kim, Elsa Andrea Kirchner, Arne Stefes, and Frank Kirchner. “Intrinsic interactive reinforcement learning - Using error-related potentials for real world human-robot interaction”. In: *Nature Scientific Reports* 7.17562 (2017). DOI: 10.1038/s41598-017-17682-7.

[Kir97] Frank Kirchner. “Q-learning of complex behaviours on a six-legged walking machine”. In: *EUROMICRO Workshop on Advanced Mobile Robots*. 1997, pp. 51–58. DOI: 10.1109/EURBOT.1997.633565.

[Kit+97] Hiroaki Kitano, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda, Eiichi Osawa, and Hitoshi Matsubara. “RoboCup: A Challenge Problem for AI”. In: *AI Magazine* 18.1 (1997), pp. 73–85.

[KKB11] Klas Kronander, Seyed Mohammad Khansari-Zadeh, and Aude Billard. "Learning to control planar hitting motions in a minigolf-like task". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Ed. by Oussama Khatib and Gaurav Sukhatme. 2011, pp. 710–717.

[KKB12] Seyed Mohammad Khansari-Zadeh, Klas Kronander, and Aude Billard. "Learning to Play Minigolf: A Dynamical System-based Approach". In: *Advanced Robotics* 26.17 (2012), pp. 1967–1993. DOI: 10.1080/01691864.2012.728692.

[KMD13] Sylvain Koos, Jean-Baptiste Mouret, and Stéphane Doncieux. "The Transferability Approach: Crossing the Reality Gap in Evolutionary Robotics". In: *IEEE Transactions on Evolutionary Computation* 17.1 (2013), pp. 122–145. DOI: 10.1109/TEVC.2012.2185849.

[KMP08] Jens Kober, Betty Mohler, and Jan Peters. "Learning perceptual coupling for motor primitives". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Ed. by Raja Chatila, Jean-Pierre Merlet, and Christian Laugier. 2008, pp. 834–839. DOI: 10.1109/IROS.2008.4650953.

[KN09] J. Zico Kolter and Andrew Y. Ng. "Policy search via the signed derivative". In: *Robotics: Science and Systems (RSS)*. Ed. by Jeff Trinkle, Yoky Matsuoka, and Jose A. Castellanos. 2009. ISBN: 978-0-262-51463-7. URL: http://www.roboticsproceedings.org/rss05/p27.html.

[Kob+10] Jens Kober, Katharina Mülling, Oliver Krömer, Christoph H. Lampert, Bernhard Schölkopf, and Jan Peters. "Movement templates for learning of hitting and batting". In: *IEEE International Conference on Robotics and Automation (ICRA)*. Ed. by Wesley Snyder and Vijay Kumar. 2010, pp. 853–858. DOI: 10.1109/ROBOT.2010.5509672.

[Kob+12] Jens Kober, Andreas Wilhelm, Erhan Öztop, and Jan Peters. "Reinforcement learning to adjust parametrized motor primitives to new situations". In: *Autonomous Robots* 33.4 (2012), pp. 361–379.

[Koe+16] Dorothea Koert, Guilherme Maeda, Rudolf Lioutikov, Gerhard Neumann, and Jan Peters. "Demonstration based trajectory optimization for generalizable robot motions". In: *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*. Ed. by Eduardo Bayro-Corrochano and Paul Oh. 2016, pp. 515–522. DOI: 10.1109/HUMANOIDS.2016.7803324.

[Kor+10] Petar Kormushev, Sylvain Calinon, Ryo Saegusa, and Giorgio Metta. "Learning the skill of archery by a humanoid robot iCub". In: *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*. Nashville, USA, 2010, pp. 417–423. URL: http://kormushev.com/papers/Kormushev_Humanoids-2010.pdf.

[Kor+11a] Petar Kormushev, Dragomir N. Nenchev, Sylvain Calinon, and Darwin G. Caldwell. "Upper-body Kinesthetic Teaching of a Free-standing Humanoid Robot". In: *IEEE International Conference on Robotics and Automation (ICRA)*. Ed. by Zexiang Li and Yuan Fang Zheng. Shanghai, China, 2011, pp. 3970–3975. URL: http://kormushev.com/papers/Kormushev_ICRA_2011.pdf.

[Kor+11b] Petar Kormushev, Barkan Ugurlu, Sylvain Calinon, Nikolas G. Tsagarakis, and Darwin G. Caldwell. "Bipedal Walking Energy Minimization by Reinforcement Learning with Evolving Policy Parameterization". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Ed. by Oussama Khatib and Gaurav Sukhatme. San Francisco, USA, 2011, pp. 318–324. URL: http://kormushev.com/papers/Kormushev-IROS2011.pdf.

[Kou+13] Jan Koutník, Giuseppe Cuccu, Jürgen Schmidhuber, and Faustino Gomez. "Evolving Large-Scale Neural Networks for Vision-Based Reinforcement Learning". In: *Genetic and Evolutionary Computation Conference*. Ed. by Christian Blum. GECCO '13. Amsterdam, The Netherlands: Association for Computing Machinery, 2013, pp. 1061–1068. ISBN: 9781450319638. DOI: 10.1145/2463372.2463509.

[KP09] Jens Kober and Jan Peters. "Policy Search for Motor Primitives in Robotics". In: *Advances in Neural Information Processing Systems*. Ed. by D. Koller, D. Schuurmans, Y. Bengio, and Leon Bottou. Curran Associates, Inc., 2009, pp. 849–856. URL: http://papers.nips.cc/paper/3545-policy-search-for-motor-primitives-in-robotics.pdf.

[KP11] Jens Kober and Jan Peters. "Policy search for motor primitives in robotics". In: *Machine Learning* 84.1–2 (2011), pp. 171–203.

[KR08] Thomas Kollar and Nicholas Roy. "Trajectory Optimization using Reinforcement Learning for Map Exploration". In: *International Journal of Robotics Research* 27.2 (2008), pp. 175–196. DOI: 10.1177/0278364907087426.

[Kra+16] Aljaž Kramberger, Rok Piltaver, Bojan Nemec, Matjaž Gams, and Aleš Ude. "Learning of assembly constraints by demonstration and active exploration". In: *Industrial Robot* 5.43 (2016), pp. 524–534. ISSN: 0143-991X. DOI: 10.1108/IR-02-2016-0058.

[Kre15] Mario Michael Krell. "Generalizing, Decoding, and Optimizing Support Vector Machine Classification". PhD thesis. Bremen, Germany: University of Bremen, 2015.

[Kro+09] Oliver Kroemer, Renaud Detry, Justus Piater, and Jan Peters. "Active learning using mean shift optimization for robot grasping". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Ed.

by Ning Xi and Zhidong Wang. 2009, pp. 2610–2615. DOI: 10.1109/IROS.2009.5354345.

[Kro+10] Oliver Kroemer, Renaud Detry, Justus H. Piater, and Jan Peters. "Combining active learning and reactive control for robot grasping". In: *Robotics and Autonomous Systems* 58.9 (2010), pp. 1105–1116. ISSN: 0921-8890. DOI: https://doi.org/10.1016/j.robot.2010.06.001.

[Krü+14] Norbert Krüger, Aleš Ude, Henrik Gordon Petersen, Bojan Nemec, Lars-Peter Ellekilde, Thiusius Rajeeth Savarimuthu, Jimmy Alison Rytz, Kerstin Fischer, Anders Glent Buch, Dirk Kraft, Wail Mustafa, Eren Erdal Aksoy, Jeremie Papon, Aljaž Kramberger, and Florentin Wörgötter. "Technologies for the Fast Set-Up of Automated Assembly Processes". In: *KI – Künstliche Intelligenz* 28 (4 2014), pp. 305–313. URL: 10.1007/s13218-014-0329-9.

[KS04] Nate Kohl and Peter Stone. "Machine Learning for Fast Quadrupedal Locomotion". In: *AAAI Conference on Artificial Intelligence*. Ed. by George Ferguson and Deborah McGuinness. 2004, pp. 611–616. URL: http://nn.cs.utexas.edu/?kohl:aaai04.

[KS06] Levente Kocsis and Csaba Szepesvári. "Discounted UCB". In: *2nd PASCAL Challenges Workshop*. Venice, Italy, 2006.

[KS17] Oliver Kroemer and Gaurav S. Sukhatme. "Feature selection for learning versatile manipulation skills based on observed and desired trajectories". In: *IEEE International Conference on Robotics and Automation (ICRA)*. Ed. by I-Ming Chen and Yoshihiko Nakamura. 2017, pp. 4713–4720. DOI: 10.1109/ICRA.2017.7989546.

[KSB10] Sanjeev Kumar, Nagarajan Sukavanam, and Raman Balasubramanian. "An optimization approach to solve the inverse kinematics of redundant manipulator". In: *International Journal of Information and System Sciences* 6 (4 2010), pp. 414–423.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira, C. J. C. Burges, Leon Bottou, and K. Q. Weinberger. Curran Associates, Inc., 2012, pp. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[Kue+14] Daniel Kuehn, Felix Bernhard, Armin Burchardt, Moritz Schilling, Tobias Stark, Martin Zenzes, and Frank Kirchner. "Distributed Computation in a Quadrupedal Robotic System". In: *International Journal of Advanced Robotic Systems* 11.7 (2014), p. 110. DOI: 10.5772/58733.

[Kui+16] Scott Kuindersma, Robin Deits, Maurice Fallon, Andrés Valenzuela, Hongkai Dai, Frank Permenter, Twan Koolen, Pat Marion, and Russ Tedrake. "Optimization-based locomotion planning, estimation, and control design for the atlas humanoid robot". In: *Autonomous Robots* 40.3 (2016), pp. 429–455. ISSN: 1573-7527. DOI: 10.1007/s10514-015-9479-3.

[Kuk20] Kuka AG. *LBR iiwa*. 2020. URL: https://www.kuka.com/en-us/products/robotics-systems/industrial-robots/lbr-iiwa (visited on 05/06/2020).

[Kup+13] Andras Gabor Kupcsik, Marc Peter Deisenroth, Jan Peters, and Gerhard Neumann. "Data-Efficient Generalization of Robot Skills with Contextual Policy Search". In: *AAAI Conference on Artificial Intelligence*. Ed. by Marie desJardins and Michael L. Littman. AAAI'13. Bellevue, Washington: AAAI Press, 2013, pp. 1401–1407.

[Kup+17] Andras Kupcsik, Marc Peter Deisenroth, Jan Peters, Ai Poh Loh, Prahlad Vadakkepat, and Gerhard Neumann. "Model-based contextual policy search for data-efficient generalization of robot skills". In: *Artificial Intelligence* 247 (2017). Special Issue on AI and Robotics, pp. 415–439. ISSN: 0004-3702. DOI: 10.1016/j.artint.2014.11.005.

[KW14] Diederik P. Kingma and Max Welling. "Auto-Encoding Variational Bayes". In: *International Conference on Learning Representations (ICLR)*. Ed. by Yoshua Bengio and Yann LeCun. 2014.

[LA14] Sergey Levine and Pieter Abbeel. "Learning Neural Network Policies with Guided Policy Search under Unknown Dynamics". In: *Advances in Neural Information Processing Systems*. Ed. by Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger. Curran Associates, Inc., 2014, pp. 1071–1079. URL: http://papers.nips.cc/paper/5444-learning-neural-network-policies-with-guided-policy-search-under-unknown-dynamics.pdf.

[Lau+18] Johan Sund Laursen, Lars Carøe Sørensen, Ulrik Pagh Schultz, Dirk Kraft, and Lars-Peter Ellekilde. "Adapting Parameterized Motions using Iterative Learning and Online Collision Detection". In: *IEEE International Conference on Robotics and Automation (ICRA)*. Ed. by Kevin Lynch. IEEE, 2018, pp. 7587–7594. DOI: 10.1109/ICRA.2018.8463208.

[LBB07] Tobias Latzke, Sven Behnke, and Maren Bennewitz. "Imitative Reinforcement Learning for Soccer Playing Robots". In: *RoboCup: Robot Soccer World Cup*. Ed. by Gerhard Lakemeyer, Elizabeth Sklar, Domenico G. Sorrenti, and Tomoichi Takahashi. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 47–58. ISBN: 978-3-540-74024-7.

[LBH15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep Learning". In: *Nature* 512 (2015), pp. 436–444. DOI: 10.1038/nature14539.

[LeC+89] Yann LeCun, Bernhard E. Boser, John S. Denker, Donnie Henderson, Richard E. Howard, Wayne E. Hubbard, and Lawrence D. Jackel. "Backpropagation Applied to Handwritten Zip Code Recognition". In: *Neural Computation* 1.4 (1989), pp. 541–551. ISSN: 0899-7667. DOI: 10.1162/neco.1989.1.4.541.

[Lem+14] Andre Lemme, Klaus Neumann, René Felix Reinhart, and Jochen J. Steil. "Neural learning of vector fields for encoding stable dynamical systems". In: *Neurocomputing* 141 (2014), pp. 3–14. ISSN: 0925-2312. DOI: 10.1016/j.neucom.2014.02.012.

[Lev+16] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. "End-to-End Training of Deep Visuomotor Policies". In: *Journal of Machine Learning Research* 17.39 (2016), pp. 1–40. URL: http://jmlr.org/papers/v17/15-522.html.

[Lev+18] Sergey Levine, Peter Pastor, Alex Krizhevsky, Julian Ibarz, and Deirdre Quillen. "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection". In: *International Journal of Robotics Research* 37.4-5 (2018), pp. 421–436. DOI: 10.1177/0278364917710318.

[Lia+19] Thomas Liao, Grant Wang, Brian Yang, Rene Lee, Kristofer Pister, Sergey Levine, and Roberto Calandra. "Data-efficient Learning of Morphology and Controller for a Microrobot". In: *IEEE International Conference on Robotics and Automation (ICRA)*. Ed. by Ayanna Howard. 2019, pp. 2488–2494.

[Lil+16] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. "Continuous control with deep reinforcement learning". In: *International Conference on Learning Representations (ICLR)*. Ed. by Samy Bengio and Brian Kingsbury. 2016.

[Lin+17] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. "Focal Loss for Dense Object Detection". In: *IEEE International Conference on Computer Vision (ICCV)*. Ed. by Rita Cucchiara, Yasuyuki Matsushita, Nicu Sebe, and Stefano Soatto. 2017, pp. 2999–3007. DOI: 10.1109/ICCV.2017.324.

[Lin92] Long-Ji Lin. "Self-improving reactive agents based on reinforcement learning, planning and teaching". In: *Machine Learning* 8 (3 1992). Ed. by John Case and Anselm Blumer, pp. 293–321. ISSN: 0885-6125.

[Lio+16] Rudolf Lioutikov, Oliver Kroemer, Guilherme Maeda, and Jan Peters. "Learning Manipulation by Sequencing Motor Primitives with a Two-Armed Robot". In: *International Conference on Intelligent Autonomous Systems*. Ed. by Emanuele Menegatti, Nathan Michael, Karsten Berns, and Hiroaki Yamaguchi. 2016, pp. 1601–1611.

[Liu+13] M. Liu, Bruno Depraetere, Gregory Pinte, Ivo Grondman, and Robert Babuška. "Model-free and model-based time-optimal control of a badminton robot". In: *Asian Control Conference (ASCC)*. 2013, pp. 1–6. DOI: 10.1109/ASCC.2013.6606242.

[Liz+07] Daniel Lizotte, Tao Wang, Michael Bowling, and Dale Schuurmans. "Automatic Gait Optimization with Gaussian Process Regression". In: *International Joint Conference on Artificial Intelligence (IJCAI)*. Ed. by Manuela M. Veloso. IJCAI'07. Hyderabad, India: Morgan Kaufmann Publishers Inc., 2007, pp. 944–949.

[LK13] Sergey Levine and Vladlen Koltun. "Guided Policy Search". In: *International Conference on Machine Learning (ICML)*. Ed. by Sanjoy Dasgupta and David McAllester. Vol. 28. Proceedings of Machine Learning Research 3. Atlanta, Georgia, USA: PMLR, 2013, pp. 1–9. URL: http://proceedings.mlr.press/v28/levine13.html.

[LKS15] Ian Lenz, Ross Knepper, and Ashutosh Saxena. "DeepMPC: Learning Deep Latent Features for Model Predictive Control". In: *Robotics: Science and Systems (RSS)*. Ed. by Lydia E. Kavraki, David Hsu, and Jonas Buchli. Robotics: Science and Systems Foundation, 2015. ISBN: 978-0-9923747-1-6. DOI: 10.15607/RSS.2015.XI.012.

[LLF09] Daniel A. Levitis, William Z. Lidicker, and Glenn Freund. "Behavioural biologists do not agree on what constitutes behaviour". In: *Animal Behaviour* 78.1 (2009), pp. 103–110. ISSN: 0003-3472. DOI: 10.1016/j.anbehav.2009.03.018.

[LLS15] Ian Lenz, Honglak Lee, and Ashutosh Saxena. "Deep Learning for Detecting Robotic Grasps". In: *International Journal of Robotics Research* 34.4–5 (2015), pp. 705–724. ISSN: 0278-3649. DOI: 10.1177/0278364914549607.

[Lop+19] Nestor Gonzalez Lopez, Yue Leire Erro Nuin, Elias Barba Moral, Lander Usategui San Juan, Alejandro Solano Rueda, Víctor Mayoral Vilches, and Risto Kojcev. "gym-gazebo2, a toolkit for reinforcement learning using ROS 2 and Gazebo". In: *CoRR* (2019). arXiv: 1903.06278 [cs.RO].

[Loq+18] Antonio Loquercio, Ana I. Maqueda, Carlos R. del-Blanco, and Davide Scaramuzza. "DroNet: Learning to Fly by Driving". In: *IEEE Robotics and Automation Letters (RA-L)* 3.2 (2018), pp. 1088–1095. DOI: 10.1109/LRA.2018.2795643.

[Los17] Ilya Loshchilov. "LM-CMA: An Alternative to L-BFGS for Large-Scale Black Box Optimization". In: *Evolutionary Computation* 25.1 (2017), pp. 143–171. ISSN: 1063-6560. DOI: 10.1162/EVCO_a_00168.

[LR13] Thomas Lampe and Martin Riedmiller. "Acquiring visual servoing reaching and grasping skills using neural reinforcement learning". In: *International Joint Conference on Neural Networks (IJCNN)*. Ed. by Peter Erdi. 2013, pp. 1–8. DOI: 10.1109/IJCNN.2013.6707053.

[LRJ06] Martin Loetzsch, Max Risler, and Matthias Jungel. "XABSL - A Pragmatic Approach to Behavior Engineering". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Ed. by Yunhui Liu and Ning Xi. 2006, pp. 5124–5129. DOI: 10.1109/IROS.2006.282605.

[LRP19] Michael Lutter, Christian Ritter, and Jan Peters. "Deep Lagrangian Networks: Using Physics as Model Prior for Deep Learning". In: *International Conference on Learning Representations (ICLR)*. Ed. by Sergey Levine, Karen Livescu, and Shakir Mohamed. 2019. URL: https://openreview.net/forum?id=BklHpjCqKm.

[LSK14] Malte Langosz, Kai Alexander von Szadkowski, and Frank Kirchner. "Introducing Particle Swarm Optimization into a Genetic Algorithm to Evolve Robot Controllers". In: *Genetic and Evolutionary Computation Conference Companion*. Ed. by Christian Igel. GECCO '14. Vancouver, BC, Canada: ACM, 2014, pp. 9–10. ISBN: 978-1-4503-2881-4. DOI: 10.1145/2598394.2598474.

[LSS10] Ilya Loshchilov, Marc Schoenauer, and Michèle Sebag. "Comparison-Based Optimizers Need Comparison-Based Surrogates". In: *Parallel Problem Solving from Nature (PPSN)*. Ed. by Robert Schaefer, Carlos Cotta, Joanna Kołodziej, and Günter Rudolph. Springer Berlin Heidelberg, 2010, pp. 364–373. ISBN: 978-3-642-15844-5. DOI: 10.1007/978-3-642-15844-5_37.

[Mae+16] Guilherme Maeda, Marco Ewerton, Dorothea Koert, and Jan Peters. "Acquiring and Generalizing the Embodiment Mapping From Human Observations to Robot Skills". In: *IEEE Robotics and Automation Letters (RA-L)* 1.2 (2016), pp. 784–791. ISSN: 2377-3766.

[Mae+17] Guilherme J. Maeda, Gerhard Neumann, Marco Ewerton, Rudolf Lioutikov, Oliver Kroemer, and Jan Peters. "Probabilistic movement primitives for coordination of multiple human–robot collaborative tasks". In: *Autonomous Robots* 41.3 (2017), pp. 593–612. ISSN: 1573-7527. DOI: 10.1007/s10514-016-9556-2.

[Mah+17] Jeffrey Mahler, Jacky Liang, Sherdil Niyaz, Michael Laskey, Richard Doan, Xinyu Liu, Juan Aparicio Ojea, and Ken Goldberg. "Dex-Net 2.0: Deep Learning to Plan Robust Grasps with Synthetic Point Clouds and Analytic Grasp Metrics". In: *Robotics: Science and Systems (RSS)*. Ed. by Nancy Amato, Siddhartha Srinivasa, Nora Ayanian, and Scott Kuindersma. 2017. ISBN: 978-0-9923747-3-0. DOI: 10.15607/RSS.2017.XIII.058.

[Mah+18] A. Rupam Mahmood, Dmytro Korenkevych, Gautham Vasan, William Ma, and James Bergstra. "Benchmarking Reinforcement Learning Algorithms on Real-World Robots". In: *Conference on Robot Learning*. Ed. by Aude Billard, Anca Dragan, Jan Peters, and Jun Morimoto. Vol. 87. Proceedings of Machine Learning Research. PMLR, 2018, pp. 561–591. URL: http://proceedings.mlr.press/v87/mahmood18a.html.

[Man+16] Simon Manschitz, Michael Gienger, Jens Kober, and Jan Peters. "Probabilistic decomposition of sequential force interaction tasks into Movement Primitives". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Ed. by Il Hong Suh and Dong-Soo Kwon. 2016, pp. 3920–3927. DOI: 10.1109/IROS.2016.7759577.

[Man+18] Simon Manschitz, Michael Gienger, Jens Kober, and Jan Peters. "Mixture of Attractors: A novel Movement Primitive Representation for Learning Motor Skills from Demonstrations". In: *IEEE Robotics and Automation Letters (RA-L)* 3.2 (2018), pp. 926–933.

[Mar+16] Alonso Marco, Philipp Hennig, Jeannette Bohg, Stefan Schaal, and Sebastian Trimpe. "Automatic LQR Tuning Based on Gaussian Process Global Optimization". In: *IEEE International Conference on Robotics and Automation (ICRA)*. Ed. by Danica Kragic, Antonio Bicchi, and Alessandro De Luca. 2016, pp. 270–277. DOI: 10.1109/ICRA.2016.7487144.

[MAR20] MARS development team. *Machina Arte Robotum Simulans (MARS)*. 2020. URL: https://github.com/rock-simulation/mars (visited on 05/27/2020).

[Mas12] Matthew T. Mason. "Creation Myths: The Beginnings of Robotics Research". In: *IEEE Robotics Automation Magazine* 19.2 (2012), pp. 72–77. ISSN: 1070-9932. DOI: 10.1109/MRA.2012.2191437.

[Mat+05] Takamitsu Matsubara, Jun Morimoto, Jun Nakanishi, Masa-Aki Sato, and Kenji Doya. "Learning CPG-based biped locomotion with a policy gradient method". In: *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*. 2005, pp. 208–213. DOI: 10.1109/ICHR.2005.1573569.

[Mát65] I. Mátyáš. "Random optimization". In: *Automation and Remote Control* 26 (2 1965), pp. 246–253.

[MB15] Marcell Missura and Sven Behnke. "Online Learning of Bipedal Walking Stabilization". In: *KI – Künstliche Intelligenz* 29.4 (2015), pp. 401–405. ISSN: 1610-1987. DOI: 10.1007/s13218-015-0387-7.

[MB17] Jose R. Medina and Aude Billard. "Learning Stable Task Sequences from Demonstration with Linear Parameter Varying Systems and Hidden Markov Models". In: *Conference on Robot Learning (CoRL)*. Ed. by Sergey Levine, Vincent Vanhoucke, and Ken Goldberg. Vol. 78. Proceedings of Machine Learning Research. PMLR, 2017, pp. 175–184. URL: http://proceedings.mlr.press/v78/medina17a.html.

[MB90] Pattie Maes and Rodney A. Brooks. "Learning to Coordinate Behaviors". In: *AAAI Conference on Artificial Intelligence*. Ed. by Tom Dietterich and Bill Swartout. AAAI'90. Boston, Massachusetts: AAAI Press, 1990, pp. 796–802. ISBN: 0-262-51057-X.

[MC89] Michael McCloskey and Neal J. Cohen. "Catastrophic Interference in Connectionist Networks: The Sequential Learning Problem". In: *Psychology of Learning and Motivation* 24 (1989). Ed. by Gordon H. Bower, pp. 109–165. ISSN: 0079-7421. DOI: 10.1016/S0079-7421(08)60536-8.

[MC92] Sridhar Mahadevan and Jonathan Connell. "Automatic programming of behavior-based robots using reinforcement learning". In: *Artificial Intelligence* 55.2 (1992), pp. 311–365. ISSN: 0004-3702. DOI: 10.1016/0004-3702(92)90058-6.

[MCM13] Stefano Michieletto, Nicola Chessa, and Emanuele Menegatti. "Learning how to approach industrial robot tasks from natural demonstrations". In: *IEEE Workshop on Advanced Robotics and its Social Impacts*. Ed. by Takashi Yoshimi, Hiroki Murakami, Sandra Hirche, and Katsu Yamane. 2013, pp. 255–260. DOI: 10.1109/ARSO.2013.6705538.

[MD01] Jun Morimoto and Kenji Doya. "Acquisition of stand-up behavior by a real robot using hierarchical reinforcement learning". In: *Robotics and Autonomous Systems* 36.1 (2001), pp. 37–51. ISSN: 0921-8890. DOI: 10.1016/S0921-8890(01)00113-0.

[Mei06] Nicolai Meinshausen. "Quantile Regression Forests". In: *Journal of Machine Learning Research* 7 (2006), pp. 983–999. ISSN: 1532-4435. URL: http://jmlr.csail.mit.edu/papers/v7/meinshausen06a.html.

[Met+14] Jan Hendrik Metzen, Alexander Fabisch, Lisa Senger, José de Gea Fernández, and Elsa Andrea Kirchner. "Towards Learning of Generic Skills for Robotic Manipulation". In: *KI – Künstliche Intelligenz* 28.1 (2014), pp. 15–20. ISSN: 1610-1987. DOI: 10.1007/s13218-013-0280-1.

[Met15] Jan Hendrik Metzen. "Active Contextual Entropy Search". In: *Workshop on Bayesian Optimization, Advances in Neural Information Processing Systems*. Ed. by Nando de Freitas, Ryan P. Adams, Bobak Shahriari, Roberto Calandra, and Amar Shah. Montreal, Quebec, Canada, 2015. URL: http://arxiv.org/abs/1511.04211.

[Met16] Jan Hendrik Metzen. "Minimum Regret Search for Single- and Multi-Task Optimization". In: *International Conference on Machine Learning (ICML)*. Ed. by Maria Florina Balcan and Kilian Q. Weinberger. Vol. 48. Proceedings of Machine Learning Research. New York, New York, USA: PMLR, 2016, pp. 192–200. URL: http://proceedings.mlr.press/v48/metzen16.html.

[Mew14] Florian Mewes. "Entwicklung einer dynamischen Spielstrategie auf der humanoiden Roboterplattform NAO". Available online at robocup.imn.htwk-leipzig.de/documents/BA_Florian_Mewes.pdf. Bachelor's thesis. HTWK Leipzig, 2014.

[Mey+14] Johannes Meyer, Markus Kuderer, Jörg Müller, and Wolfram Burgard. "Online Marker Labeling for Fully Automatic Skeleton Tracking in Optical Motion Capture". In: *IEEE International Conference on Robotics and Automation (ICRA)*. Ed. by Ning Xi and William R. Hamel. Hong Kong, China, 2014, pp. 5652–5657.

[MF20] Jan Hendrik Metzen and Alexander Fabisch. *Bayesian Optimization (Python library)*. 2020. URL: https://github.com/rock-learning/bayesian_optimization (visited on 06/05/2020).

[MFH15] Jan Hendrik Metzen, Alexander Fabisch, and Jonas Hansen. "Bayesian Optimization for Contextual Policy Search". In: *Machine Learning in Planning and Control of Robot Motion (MLPC) Workshop, IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Ed. by Aleksandra Faust. 2015. URL: https://www.cs.unm.edu/~afaust/MLPC15_proceedings/MLPC15_paper_Metzen.pdf.

[MFW85] Virgil Mathiowetz, Susan Federman, and Diana Wiemer. "Box and Block Test of Manual Dexterity: Norms for 6–19 Year Olds". In: *Canadian Journal of Occupational Therapy* 52.5 (1985), pp. 241–245. DOI: 10.1177/000841748505200505.

[MGR18] Horia Mania, Aurelia Guy, and Benjamin Recht. "Simple random search of static linear policies is competitive for reinforcement learning". In: *Advances in Neural Information Processing Systems*. Ed. by Samy Bengio, Hanna Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett. Curran Associates, Inc., 2018, pp. 1800–1809. URL: http://papers.nips.cc/paper/7451-simple-random-search-of-static-linear-policies-is-competitive-for-reinforcement-learning.pdf.

[MHM10] Takamitsu Matsubara, Sang-Ho Hyon, and Jun Morimoto. "Learning Stylistic Dynamic Movement Primitives from multiple demonstrations". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Ed. by Ren C. Luo and Huei-Yung Lin. 2010, pp. 1277–1283. DOI: 10.1109/IROS.2010.5651049.

[MHN13] Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. "Rectifier nonlinearities improve neural network acoustic models". In: *Workshop on Deep Learning for Audio, Speech, and Language Processing, International Conference on Machine Learning (ICML)*. Ed. by Sanjoy Dasgupta and David McAllester. 2013.

[MKP11] Katharina Mülling, Jens Kober, and Jan Peters. "A biomimetic approach to robot table tennis". In: *Adaptive Behavior* 19.5 (2011), pp. 359–376.

[ML93] Matthew T. Mason and Kevin Lynch. "Dynamic Manipulation". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Ed. by Masatsugu Kidoe and Tomomasa Sato. Vol. 1. 1993, pp. 152–159.

[MM98] Olvi L. Mangasarian and David R. Musicant. "Successive Overrelaxation for Support Vector Machines". In: *IEEE Transactions on Neural Networks* 10 (1998), pp. 1032–1037.

[Mni+15] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (2015), pp. 529–533. ISSN: 00280836. DOI: 10.1038/nature14236.

[Mni+16] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. "Asynchronous Methods for Deep Reinforcement Learning". In: *International Conference on Machine Learning (ICML)*. Ed. by Maria Florina Balcan and Kilian Q. Weinberger. Vol. 48. Proceedings of Machine Learning Research. New York, New York, USA: PMLR, 2016, pp. 1928–1937.

[Moh+19] Shakir Mohamed, Mihaela Rosca, Michael Figurnov, and Andriy Mnih. "Monte Carlo Gradient Estimation in Machine Learning". In: *CoRR* abs/1906.10652 (2019). arXiv: 1906.10652.

[Mor88] Hans Moravec. *Mind Children: The Future of Robot and Human Intelligence*. Cambridge, MA, USA: Harvard University Press, 1988. ISBN: 0-674-57616-0.

[MRG03] Shie Mannor, Reuven Rubinstein, and Yohai Gat. "The Cross Entropy Method for Fast Policy Search". In: *International Conference on Machine Learning (ICML)*. Ed. by Tom Fawcett and Nina Mishra. ICML'03. Washington, DC, USA: AAAI Press, 2003, pp. 512–519. ISBN: 1577351894.

[MT19] Aditya Modi and Ambuj Tewari. "Contextual Markov Decision Processes using Generalized Linear Models". In: *International Conference on Machine Learning (ICML), Workshop RL4RealLife*. Ed. by Alborz Geramifard, Lihong Li, Yuxi Li, Csaba Szepesvari, and Tao Wang. 2019. URL: https://openreview.net/forum?id=BklhOSiQiN.

[Mül+07] Heiko Müller, Martin Lauer, Roland Hafner, Sascha Lange, Artur Merke, and Martin Riedmiller. "Making a Robot Learn to Play Soccer Using Reward and Punishment". In: *KI: Advances in Artificial Intelligence*. Ed. by Joachim Hertzberg, Michael Beetz, and Roman Englert. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 220–234. ISBN: 978-3-540-74565-5.

[Mül+13] Katharina Mülling, Jens Kober, Oliver Krömer, and Jan Peters. "Learning to Select and Generalize Striking Movements in Robot Table Tennis". In: *International Journal of Robotics Research* 32.3 (2013).

[Mun+16] Remi Munos, Tom Stepleton, Anna Harutyunyan, and Marc Bellemare. "Safe and Efficient Off-Policy Reinforcement Learning". In: *Advances in Neural Information Processing Systems*. Ed. by D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett. Curran Associates, Inc., 2016, pp. 1054–1062.

[Mur12] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. MIT Press, 2012. ISBN: 978-0-262-01802-9.

[Nag+17] Anusha Nagabandi, Gregory Kahn, Ronald S. Fearing, and Sergey Levine. "Neural Network Dynamics for Model-Based Deep Reinforcement Learning with Model-Free Fine-Tuning". In: *CoRR* (2017). arXiv: 1708.02596 [cs.LG].

[ND02] Chrystopher L. Nehaniv and Kerstin Dautenhahn. "The Correspondence Problem". In: *Imitation in Animals and Artifacts*. Ed. by Kerstin Dautenhahn and Chrystopher L. Nehaniv. Cambridge, MA, USA: MIT Press, 2002, pp. 41–61. ISBN: 0262042037.

[Nel+12] Gabe Nelson, Aaron Saunders, Neil Neville, Ben Swilling, Joe Bondaryk, Devin Billings, Chris Lee, Robert Playter, and Marc Raibert. "PETMAN: A Humanoid Robot for Testing Chemical Protective Clothing". In: *Journal of the Robotics Society of Japan* 30.4 (2012), pp. 372–377. DOI: 10.7210/jrsj.30.372.

[Nem+18] Bojan Nemec, Ken'ichi Yasuda, Nathaneal Mullennix, Nejc Likar, and Aleš Ude. "Learning by Demonstration and Adaptation of Finishing Operations Using Virtual Mechanism Approach". In: *IEEE International Conference on Robotics and Automation (ICRA)*. Ed. by Kevin Lynch. 2018, pp. 7219–7225.

[Neu11] Gerhard Neumann. "Variational Inference for Policy Search in changing situations". In: *International Conference on Machine Learning (ICML)*. Ed. by Lise Getoor and Tobias Scheffer. Bellevue, Washington, USA, 2011. ISBN: 978-1-4503-0619-5.

[Nie+15] Scott Niekum, Sarah Osentoski, George Konidaris, Sachin Chitta, Bhaskara Marthi, and Andrew G. Barto. "Learning grounded finite-state representations from unstructured demonstrations". In: *International Journal of Robotics Research* 34.2 (2015), pp. 131–157. DOI: 10.1177/0278364914554471.

[Nik+13] Stefanos Nikolaidis, Przemyslaw Lasota, Gregory Rossano, Carlos Martinez, Thomas Fuhlbrigge, and Julie A. Shah. "Human-robot collaboration in manufacturing: Quantitative evaluation of predictable, convergent joint action". In: *International Symposium on Robotics (ISR)*. 2013, pp. 1–6. DOI: 10.1109/ISR.2013.6695625.

[Nil09] Rickard Nilsson. "Inverse kinematics". MA thesis. Luleå University of Technology, 2009.

[NM65] John A. Nelder and Roger Mead. "A Simplex Method for Function Minimization". In: *The Computer Journal* 7.4 (1965), pp. 308–313. ISSN: 0010-4620. DOI: 10.1093/comjnl/7.4.308.

[Noc80] Jorge Nocedal. "Updating quasi-Newton matrices with limited storage". In: *Mathematics of Computation* 35 (151 1980), pp. 773–782. DOI: 10.2307/2006193.

[Nor13] Donald A. Norman. *The Design of Everyday Things*. Revised and expanded edition. Basic Books, 2013. ISBN: 9780465050659.

[NR00] Andrew Y. Ng and Stuart J. Russell. "Algorithms for Inverse Reinforcement Learning". In: *International Conference on Machine Learning (ICML)*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2000, pp. 663–670. ISBN: 1-55860-707-2.

[NŽU17] Bojan Nemec, Leon Žlajpah, and Aleš Ude. "Door opening by joining reinforcement learning and intelligent control". In: *International Conference on Advanced Robotics (ICAR)*. 2017, pp. 222–228. DOI: 10.1109/ICAR.2017.8023522.

[OG08] David E. Orin and Ambarish Goswami. "Centroidal Momentum Matrix of a humanoid robot: Structure and properties". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Ed. by Raja Chatila, Jean-Pierre Merlet, and Christian Laugier. 2008, pp. 653–659. DOI: 10.1109/IROS.2008.4650772.

[OGL13] David E. Orin, Ambarish Goswami, and Sung-Hee Lee. "Centroidal dynamics of a humanoid robot". In: *Autonomous Robots* 35.2 (2013), pp. 161–176. ISSN: 1573-7527. DOI: 10.1007/s10514-013-9341-4.

[OHB10] Stefan Oßwald, Armin Hornung, and Maren Bennewitz. "Learning reliable and efficient navigation with a humanoid". In: *IEEE International Conference on Robotics and Automation (ICRA)*. Ed. by Wesley Snyder and Vijay Kumar. 2010, pp. 2375–2380. DOI: 10.1109/ROBOT.2010.5509420.

[OK04] Pierre-Yves Oudeyer and Frederic Kaplan. "Intelligent adaptive curiosity: a source of self-development". In: *International Workshop on Epigenetic Robotics*. Ed. by Luc Berthouze, Hideki Kozima, Christopher G. Prince, Giulio Sandini, Georgi Stojanov, G. Metta, and C. Balkenius. Lund University Cognitive Studies, 2004, pp. 127–130.

[Ola15] Chris Olah. *Neural Networks, Types, and Functional Programming*. 2015. URL: http://colah.github.io/posts/2015-09-NN-Types-FP/ (visited on 02/11/2020).

[Ope+19a] OpenAI, : Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique Pondé de Oliveira Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. "Dota 2 with Large Scale Deep Reinforcement Learning". In: *CoRR* (2019). arXiv: 1912.06680 [cs.LG].

[Ope+19b] OpenAI, Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, Jonas Schneider, Nikolas Tezak, Jerry Tworek, Peter Welinder, Lilian Weng, Qiming Yuan, Wojciech Zaremba, and Lei Zhang. "Solving Rubik's Cube with a Robot Hand". In: *CoRR* (2019). arXiv: 1910.07113 [cs.LG].

[Ope+20] OpenAI, Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Józefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, Jonas Schneider, Szymon Sidor, Josh Tobin, Peter Welinder, Lilian Weng, and Wojciech Zaremba. "Learning dexterous in-hand manipulation". In: *International Journal of Robotics Research* 39.1 (2020), pp. 3–20. DOI: 10.1177/0278364919887447.

[Ope18] OpenAI. *OpenAI Five*. 2018. URL: https://blog.openai.com/openai-five/ (visited on 03/26/2020).

[Ope19] OpenAI. *OpenAI Five Defeats Dota 2 World Champions*. 2019. URL: https://openai.com/blog/openai-five-defeats-dota-2-world-champions/ (visited on 03/26/2020).

[OPL15] Omair Ali, Affan Pervez, and Dongheui Lee. "Robotic Calligraphy: Learning from Character Images". In: *International Workshop on Human-Friendly Robotics*. Munich, Germany, 2015.

[Osa+18] Takayuki Osa, Joni Pajarinen, Gerhard Neumann, J. Andrew Bagnell, Pieter Abbeel, and Jan Peters. "An Algorithmic Perspective on Imitation Learning". In: *Foundations and Trends in Robotics* 7.1–2 (2018), pp. 1–179. ISSN: 1935-8253. DOI: 10.1561/2300000053.

[Ott15] Marc Otto. "Crossing the "reality gap" with the Transferability Approach". MA thesis. Bremen, Germany: University of Bremen, 2015.

[Par+13] Alexandros Paraschos, Christian Daniel, Jan R Peters, and Gerhard Neumann. "Probabilistic Movement Primitives". In: *Advances in Neural Information Processing Systems*. Ed. by C. J. C. Burges, Leon Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger. Curran Associates, Inc., 2013, pp. 2616–2624. URL: http://papers.nips.cc/paper/5177-probabilistic-movement-primitives.pdf.

[Par+15] Simone Parisi, Hany Abdulsamad, Alexandros Paraschos, Christian Daniel, and Jan Peters. "Reinforcement learning vs human programming in tetherball robot games". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Ed. by Jianwei Zhang and Alois Knoll. 2015, pp. 6428–6434. DOI: 10.1109/IROS.2015.7354296.

[Par+18] Alexandros Paraschos, Christian Daniel, Jan Peters, and Gerhard Neumann. "Using probabilistic movement primitives in robotics". In: *Autonomous Robots* 42.3 (2018), pp. 529–551. ISSN: 1573-7527. DOI: 10.1007/s10514-017-9648-7.

[Pas+09] Peter Pastor, Heiko Hoffmann, Tamim Asfour, and Stefan Schaal. "Learning and generalization of motor skills by learning from demonstration". In: *IEEE International Conference on Robotics and Automation (ICRA)*. Ed. by Kazuhiro Kosuge and Katsushi Ikeuchi. 2009, pp. 763–768.

[Pas+11] Peter Pastor, Mrinal Kalakrishnan, Sachin Chitta, Evangelos Theodorou, and Stefan Schaal. "Skill learning and task outcome prediction for manipulation". In: *IEEE International Conference on Robotics and Automation (ICRA)*. Ed. by Zexiang Li and Yuan Fang Zheng. Shanghai, China, 2011, pp. 3828–3834.

[Pas+19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.nips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[Pea01] Karl Pearson. "On lines and planes of closest fit to systems of points in space". In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 2.11 (1901), pp. 559–572. DOI: 10.1080/14786440109462720.

[Pen+17] Xue Bin Peng, Glen Berseth, Kangkang Yin, and Michiel Van De Panne. "DeepLoco: dynamic locomotion skills using hierarchical deep reinforcement learning". In: *ACM Transactions on Graphics* 36.4 (2017), pp. 1–13. ISSN: 07300301. DOI: 10.1145/3072959.3073602.

[Pen+18] Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel van de Panne. "DeepMimic: Example-Guided Deep Reinforcement Learning of Physics-Based Character Skills". In: *ACM Transactions on Graphics* 37.4 (2018). ISSN: 0730-0301. DOI: 10.1145/3197517.3201311.

[Pen+20] Xue Bin Peng, Erwin Coumans, Tingnan Zhang, Tsang-Wei Lee, Jie Tan, and Sergey Levine. "Learning Agile Robotic Locomotion Skills by Imitating Animals". In: *CoRR* (2020). arXiv: 2004.00784 [cs.RO].

[Pet+12] Jan Peters, Katharina Mülling, Jens Kober, Duy Nguyen-Tuong, and Oliver Krömer. "Robot Skill Learning". In: *European Conference on Artificial Intelligence*. Ed. by Luc De Raedt, Christian Bessiere, Didier Dubois, Patrick Doherty, and Paolo Frasconi. 2012, pp. 40–45.

[Pet+14] Tadej Petrič, Andrej Gams, Leon Žlajpah, and Aleš Ude. "Online learning of task-specific dynamics for periodic tasks". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Ed. by Kevin Lynch and Lynne Parker. 2014, pp. 1790–1795. DOI: 10.1109/IROS.2014.6942797.

[Pfe+17] Mark Pfeiffer, Michael Schaeuble, Juan Nieto, Roland Siegwart, and Cesar Cadena. "From perception to decision: A data-driven approach to end-to-end motion planning for autonomous ground robots". In: *IEEE International Conference on Robotics and Automation (ICRA)*. Ed. by I-Ming Chen and Yoshihiko Nakamura. 2017, pp. 1527–1533. DOI: 10.1109/ICRA.2017.7989182.

[PG16] Lerrel Pinto and Abhinav Gupta. "Supersizing self-supervision: Learning to grasp from 50K tries and 700 robot hours". In: *IEEE International Conference on Robotics and Automation (ICRA)*. Ed. by Danica Kragic, Antonio Bicchi, and Alessandro De Luca. 2016, pp. 3406–3413. DOI: 10.1109/ICRA.2016.7487517.

[PGA18] Dario Pavllo, David Grangier, and Michael Auli. "QuaterNet: A Quaternion-based Recurrent Model for Human Motion". In: *British Machine Vision Conference*. Ed. by Hubert P. H. Shum and Timothy Hospedales. BMVA Press, 2018.

[PH04] Nancy S. Pollard and Jessica K. Hodgins. "Generalizing Demonstrated Manipulation Tasks". In: *Algorithmic Foundations of Robotics V*. Ed. by Jean-Daniel Boissonnat, Joel Burdick, Ken Goldberg, and Seth Hutchinson. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 523–539. ISBN: 978-3-540-45058-0. DOI: 10.1007/978-3-540-45058-0_31.

[PHS08] Peter Pastor, Heiko Hoffmann, and Stefan Schaal. "Movement generation by learning from demonstration and generalization to new targets". In: *Adaptive Motion of Animals and Machines (AMAM)*. 2008.

[Pin+19] Robert Pinsler, Peter Karkus, Andras Gabor Kupcsik, David Hsu, and Wee Sun Lee. "Factored Contextual Policy Search with Bayesian optimization". In: *IEEE International Conference on Robotics and Automation (ICRA)*. Ed. by Ayanna Howard. 2019, pp. 7242–7248.

[Pla98] John Platt. *Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines*. Tech. rep. MSR-TR-98-14. Microsoft Research, 1998.

[PMA10] Jan Peters, Katharina Mülling, and Yasemin Altün. "Relative Entropy Policy Search". In: *AAAI Conference on Artificial Intelligence*. Ed. by D. Poole Fox M. Atlanta, Georgia, USA: AAAI Press, 2010, pp. 1607–1612. ISBN: 978-1-577-35463-5.

[PML17] Affan Pervez, Yuecheng Mao, and Dongheui Lee. "Learning deep movement primitives using convolutional neural networks". In: *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*. Ed. by Tamim Asfour. 2017, pp. 191–197. DOI: 10.1109/HUMANOIDS.2017.8246874.

[Pol+02] Nancy S. Pollard, Jessica K. Hodgins, Marcia J. Riley, and Christopher G. Atkeson. "Adapting human motion for the control of a humanoid robot". In: *IEEE International Conference on Robotics and Automation (ICRA)*. Ed. by William R. Hamel and Anthony A. Maciejewski. Vol. 2. 2002, pp. 1390–1397. DOI: 10.1109/ROBOT.2002.1014737.

[PS06] Jan Peters and Stefan Schaal. "Policy gradient methods for robotics". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Ed. by Yunhui Liu and Ning Xi. 2006, pp. 2219–2225.

[PS07] Jan Peters and Stefan Schaal. "Reinforcement Learning by Reward-weighted Regression for Operational Space Control". In: *International Conference on Machine Learning (ICML)*. Ed. by Zoubin Ghahramani. ICML '07. Corvalis, Oregon, USA: Association for Computing Machinery, 2007, pp. 745–750. ISBN: 9781595937933. DOI: 10.1145/1273496.1273590.

[PS08a] Jan Peters and Stefan Schaal. "Natural Actor-Critic". In: *Neurocomputing* 71.7-9 (2008), pp. 1180–1190. ISSN: 0925-2312.

[PS08b] Jan Peters and Stefan Schaal. "Reinforcement learning of motor skills with policy gradients". In: *Neural Networks* 21.4 (2008). Robotics and Neuroscience, pp. 682–697. ISSN: 0893-6080. DOI: 10.1016/j.neunet.2008.02.003.

[PVS05] Jan Peters, Sethu Vijayakumar, and Stefan Schaal. "Natural Actor-Critic". In: *European Conference on Machine Learning*. Ed. by J. Gama, R. Camacho, P. Brazdil, A. Jorge, and L. Torgo. Vol. 3720. Springer, 2005, pp. 280–291.

[Qua20a] Qualisys AB. *5+, 6+ and 7+ series*. 2020. URL: https://www.qualisys.com/hardware/5-6-7/ (visited on 05/27/2020).

[Qua20b] Qualisys AB. *Qualisys Track Manager (QTM)*. 2020. URL: https://www.qualisys.com/software/qualisys-track-manager/ (visited on 04/04/2020).

[Qui+09] Morgan Quigley, Brian Gerkey, Ken Conley, Josh Faust, Tully Foote, Jeremy Leibs, Eric Berger, Rob Wheeler, and Andrew Y. Ng. "ROS: an open-source Robot Operating System". In: *Workshop on Open Source Software, IEEE International Conference on Robotics and Automation (ICRA)* 3 (2009).

[Rah+18] Rouhollah Rahmatizadeh, Pooya Abolghasemi, Aman Behal, and Ladislau Bölöni. "From Virtual Demonstration to Real-World Manipulation Using LSTM and MDN". In: *AAAI Conference on Artificial Intelligence*. Ed. by Sheila McIlraith and Kilian Weinberger. 2018. URL: https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16194.

[Rai+08] Marc Raibert, Kevin Blankespoor, Gabriel Nelson, and Rob Playter. "BigDog, the Rough-Terrain Quadruped Robot". In: *IFAC World Congress*. Ed. by Dongil Cho and Shinji Hara. Vol. 41. 2. 2008, pp. 10822–10825. DOI: 10.3182/20080706-5-KR-1001.01833.

[Raj+17a] Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel V. Todorov, and Sergey Levine. "Learning Complex Dexterous Manipulation with Deep Reinforcement Learning and Demonstrations". In: *CoRR* abs/1709.10087 (2017). arXiv: 1709.10087.

[Raj+17b] Aravind Rajeswaran, Kendall Lowrey, Emanuel V. Todorov, and Sham M. Kakade. "Towards Generalization and Simplicity in Continuous Control". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Curran Associates, Inc., 2017, pp. 6550–6561. URL: http://papers.nips.cc/paper/7233-towards-generalization-and-simplicity-in-continuous-control.pdf.

[Rat90] Roger Ratcliff. "Connectionist models of recognition memory: constraints imposed by learning and forgetting functions". In: *Psychological Review* 97 (2 1990), pp. 285–308.

[Rau+12] Christian Rauch, Tim Köhler, Martin Schröer, Elmar Berghöfer, and Frank Kirchner. "A Concept of a Reliable Three-Layer Behaviour Control System for Cooperative Autonomous Robots". In: *KI: Advances in Artificial Intelligence*. Ed. by Birte Glimm and Antonio Krüger. 2012.

[Rau+19] Paulo Rauber, Avinash Ummadisingu, Filipe Mutz, and Jürgen Schmidhuber. "Hindsight policy gradients". In: *International Conference on Learning Representations (ICLR)*. Ed. by Sergey Levine, Karen Livescu, and Shakir Mohamed. 2019. URL: https://openreview.net/forum?id=Bkg2viA5FQ.

[RE13] Paul Ruvolo and Eric Eaton. "Scalable Lifelong Learning with Active Task Selection". In: *AAAI Spring Symposium: Lifelong Machine Learning*. Vol. SS-13-05. AAAI Technical Report. AAAI, 2013.

[Rec71] Ingo Rechenberg. "Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution". PhD thesis. Berlin, Germany: TU Berlin, 1971.

[RF18] Joseph Redmon and Ali Farhadi. "YOLOv3: An Incremental Improvement". In: *CoRR* abs/1804.02767 (2018). arXiv: 1804.02767.

[RG07] Martin Riedmiller and Thomas Gabel. "On Experiences in a Complex and Competitive Gaming Domain: Reinforcement Learning Meets RoboCup". In: *IEEE Symposium on Computational Intelligence and Games*. Ed. by Alan Blair, Sung-Bae Cho, and Simon M. Lucas. 2007, pp. 17–23. DOI: 10.1109/CIG.2007.368074.

[RGB11] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. "A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning". In: *International Conference on Artificial Intelligence and Statistics (AISTATS)*. Ed. by Geoffrey Gordon, David Dunson, and Miroslav Dudík. Vol. 15. Proceedings of Machine Learning Research. Fort Lauderdale, FL, USA: PMLR, 2011, pp. 627–635.

[Rie+09] Martin Riedmiller, Thomas Gabel, Roland Hafner, and Sascha Lange. "Reinforcement learning for robot soccer". In: *Autonomous Robots* 27.1 (2009), pp. 55–73. ISSN: 1573-7527. DOI: 10.1007/s10514-009-9120-4.

[Rie05] Martin Riedmiller. "Neural Fitted Q Iteration - First Experiences with a Data Efficient Neural Reinforcement Learning Method". In: *European Conference on Machine Learning*. Ed. by J. Gama, R. Camacho, P. Brazdil, A. Jorge, and L. Torgo. Springer, 2005, pp. 317–328.

[RK17] Nemanja Rakicevic and Petar Kormushev. "Efficient Robot Task Learning and Transfer via Informed Search in Movement Parameter Space". In: *Workshop on Acting and Interacting in the Real World: Challenges in Robot Learning, Advances in Neural Information Processing Systems*. Ed. by Ingmar Posner, Raia Hadsell, Martin Riedmiller, Markus Wulfmeier, and Rohan Paul. 2017.

[RMD07] Martin Riedmiller, Michael Montemerlo, and Hendrik Dahlkamp. "Learning to Drive a Real Car in 20 Minutes". In: *Frontiers in the Convergence of Bioscience and Information Technologies*. Ed. by Daniel Howard, Phill Kyu Rhee, Saman Halgamuge, and Seong-Joon Yoo. 2007, pp. 645–650. DOI: 10.1109/FBIT.2007.37.

[RMG17] Daniel Rakita, Bilge Mutlu, and Michael Gleicher. "A Motion Retargeting Method for Effective Mimicry-Based Teleoperation of Robot Arms". In: *ACM/IEEE International Conference on Human-Robot Interaction*. Ed. by Astrid Weiss and James Young. HRI '17. Vienna, Austria: Association for Computing Machinery, 2017, pp. 361–370. ISBN: 9781450343367. DOI: 10.1145/2909824.3020254.

[RMG18] Daniel Rakita, Bilge Mutlu, and Michael Gleicher. "RelaxedIK: Real-time Synthesis of Accurate and Feasible Robot Arm Motion". In: *Robotics: Science and Systems (RSS)*. Ed. by Hadas Kress-Gazit, Siddhartha Srinivasa, Tom Howard, and Nikolay Atanasov. 2018. ISBN: 978-0-9923747-4-7. DOI: 10.15607/RSS.2018.XIV.043.

[RN94] Gavin Adrian Rummery and Mahesan Niranjan. *On-line Q-learning using connectionist systems*. Tech. rep. CUED/F-INFENG/TR 166. Engineering Department, Cambridge University, 1994.

[Rob52] Herbert Robbins. "Some aspects of the sequential design of experiments". In: *Bulletin of the American Mathematical Society* 58.5 (1952), pp. 527–535.

[Röf18] Thomas Röfer. "CABSL - C-based Agent Behavior Specification Language". In: *RoboCup: Robot Soccer World Cup*. Ed. by Hidehisa Akiyama, Oliver Obst, Claude Sammut, and Flavio Tonidandel. Lecture Notes in Artificial Intelligence. Springer, 2018.

[Ros+13] Stéphane Ross, Narek Melik-Barkhudarov, Kumar Shaurya Shankar, Andreas Wendel, Debadeepta Dey, J. Andrew Bagnell, and Martial Hebert. "Learning monocular reactive UAV control in cluttered natural environments". In: *IEEE International Conference on Robotics and Automation (ICRA)*. Ed. by Lynne E. Parker. 2013, pp. 1765–1772.

[Roz+15] Leonel Rozo, Danilo Bruno, Sylvain Calinon, and Darwin G. Caldwell. "Learning optimal controllers in human-robot cooperative transportation tasks with position and force constraints". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Ed. by Jianwei Zhang and Alois Knoll. 2015, pp. 1024–1030. DOI: 10.1109/IROS.2015.7353496.

[RP20] Filipe Rodrigues and Francisco C. Pereira. "Beyond Expectation: Deep Joint Mean and Quantile Regression for Spatiotemporal Problems". In: *IEEE Transactions on Neural Networks and Learning Systems* (2020), pp. 1–13.

[RSF13] Eric Rohmer, Surya P. N. Singh, and Marc Freese. "V-REP: A versatile and scalable robot simulation framework". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Ed. by Shigeki Sugano and Makoto Kaneko. 2013, pp. 1321–1326. DOI: 10.1109/IROS.2013.6696520.

[Rub99] Reuven Rubinstein. "The Cross-Entropy Method for Combinatorial and Continuous Optimization". In: *Methodology And Computing In Applied Probability* 1 (2 1999), pp. 127–190. DOI: 10.1023/A:1010091220143.

[Rue+15] Elmar Rueckert, Jan Mundo, Alexandros Paraschos, Jan Peters, and Gerhard Neumann. "Extracting low-dimensional control variables for movement primitives". In: *IEEE International Conference on Robotics and Automation (ICRA)*. Ed. by Lynne Parker and Nancy Amato. 2015, pp. 1511–1518. DOI: 10.1109/ICRA.2015.7139390.

[RW05] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2005.

[Sal+17] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. "Evolution Strategies as a Scalable Alternative to Reinforcement Learning". In: *CoRR* (2017). arXiv: 1703.03864 [stat.ML].

[San+18] Jose Sanchez, Juan-Antonio Corrales, Belhassen-Chedli Bouzgarrou, and Youcef Mezouar. "Robotic manipulation and sensing of deformable objects in domestic and industrial applications: a survey". In: *International Journal of Robotics Research* (2018), pp. 1–29. ISSN: 0278-3649. DOI: 10.1177/0278364918779698.

[SB18] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. 2nd ed. MIT Press, 2018. ISBN: 9780262039246.

[SBS10] David Silver, J. Andrew Bagnell, and Anthony Stentz. "Learning from Demonstration for Autonomous Navigation in Complex Unstructured Terrain". In: *International Journal of Robotics Research* 29.12 (2010), pp. 1565–1592. ISSN: 0278-3649. DOI: 10.1177/0278364910369715.

[SBW92] Richard S. Sutton, Andrew G. Barto, and Ronald J. Williams. "Reinforcement learning is direct adaptive optimal control". In: *IEEE Control Systems Magazine* 12.2 (1992), pp. 19–22. ISSN: 1066-033X. DOI: 10.1109/37.126844.

[SC78] Hiroaki Sakoe and Seibi Chiba. "Dynamic programming algorithm optimization for spoken word recognition". In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 26.1 (1978), pp. 43–49. ISSN: 0096-3518. DOI: 10.1109/TASSP.1978.1163055.

[Sch+04] Stefan Schaal, Dagmar Sternad, Rieko Osu, and Mitsuo Kawato. "Rhythmic movement is not discrete". In: *Nature Neuroscience* 7.10 (2004), pp. 1137–1144.

[Sch+14] Jakob Schwendner, Thomas M. Roehr, Stefan Haase, Malte Wirkus, Marc Manz, Sascha Arnold, and Janosch Machowinski. "The Artemis Rover as an Example for Model Based Engineering in Space Robotics". In: *Workshop on Modelling, Estimation, Perception and Control of All Terrain Mobile Robots, IEEE International Conference on Robotics and Automation (ICRA)*. Ed. by Philippe Martinet, Kasuya Yoshida, and Marcel Bergerman. Hong Kong, China: IEEE, 2014. URL: http://wmepc14.irccyn.ec-nantes.fr/material/paper/paper-Artemis.pdf.

[Sch+15a] Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. "Universal Value Function Approximators". In: *International Conference on Machine Learning (ICML)*. Ed. by Francis Bach and David Blei. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, 2015, pp. 1312–1320.

[Sch+15b] Tobias Schubert, Alexis Gkogkidis, Tonio Ball, and Wolfram Burgard. "Automatic initialization for skeleton tracking in optical motion capture". In: *IEEE International Conference on Robotics and Automation (ICRA)*. Ed. by Lynne Parker and Nancy Amato. 2015, pp. 734–739.

[Sch+15c] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. "Trust Region Policy Optimization". In: *International Conference on Machine Learning (ICML)*. Ed. by Francis Bach and David Blei. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, 2015, pp. 1889–1897.

[Sch+16a] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. "Prioritized Experience Replay". In: *International Conference on Learning Representations (ICLR)*. Ed. by Samy Bengio and Brian Kingsbury. Puerto Rico, 2016.

[Sch+16b] Tobias Schubert, Katharina Eggensperger, Alexis Gkogkidis, Frank Hutter, Tonio Ball, and Wolfram Burgard. "Automatic bone parameter estimation for skeleton tracking in optical motion capture". In: *IEEE International Conference on Robotics and Automation (ICRA)*. Ed. by Danica Kragic, Antonio Bicchi, and Alessandro De Luca. 2016, pp. 5548–5554.

[Sch+16c] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. "High-Dimensional Continuous Control Using Generalized Advantage Estimation". In: *International Conference on Learning Representations (ICLR)*. Ed. by Samy Bengio and Brian Kingsbury. 2016.

[Sch+17a] Connor Schenck, Jonathan Tompson, Sergey Levine, and Dieter Fox. "Learning Robotic Manipulation of Granular Media". In: *Conference on Robot Learning (CoRL)*. Ed. by Sergey Levine, Vincent Vanhoucke, and Ken Goldberg. Vol. 78. Proceedings of Machine Learning Research. PMLR, 2017, pp. 239–248. URL: http://proceedings.mlr.press/v78/schenck17a.html.

[Sch+17b] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. "Proximal Policy Optimization Algorithms". In: *CoRR* abs/1707.06347 (2017). arXiv: 1707.06347.

[Sch+19] Roy Schwartz, Jesse Dodge, Noah A. Smith, and Oren Etzioni. "Green AI". In: *CoRR* abs/1907.10597 (2019). arXiv: 1907.10597.

[Sch14] Jürgen Schmidhuber. "Deep Learning in Neural Networks: An Overview". In: *CoRR* abs/1404.7828 (2014). arXiv: 1404.7828.

[Sen+14] Lisa Senger, Martin Schröer, Jan Hendrik Metzen, and Elsa Andrea Kirchner. "Velocity-based Multiple Change-point Inference for Unsupervised Segmentation of Human Movement Behavior". In: *International Conference on Pattern Recognition (ICPR)*. Ed. by Cheng-Lin Liu, Rama Chellappa, and Matti Pietikainen. 2014, pp. 4564–4569. DOI: 10.1109/ICPR.2014.781.

[Ser+18] Pierre Sermanet, Corey Lynch, Yevgen Chebotar, Jasmine Hsu, Eric Jang, Stefan Schaal, and Sergey Levine. "Time-Contrastive Networks: Self-Supervised Learning from Video". In: *IEEE International Conference on Robotics and Automation (ICRA)*. Ed. by Kevin Lynch. 2018, pp. 486–487.

[Set10] Burr Settles. *Active Learning Literature Survey*. Tech. rep. 1648. University of Wisconsin–Madison, 2010.

[SFS98] Marco Santello, Martha Flanders, and John F. Soechting. "Postural hand synergies for tool use". In: *Journal of Neuroscience* 18 23 (1998), pp. 10105–10115.

[Sha70] David F. Shanno. "Conditioning of quasi-Newton methods for function minimization". In: *Mathematics of Computation* 24.111 (1970), pp. 647–656. DOI: 10.1090/S0025-5718-1970-0274029-X.

[SHS09] Ingo Steinwart, Don Hush, and Clint Scovel. "Training SVMs without offset". In: *Journal of Machine Learning Research* 12 (2009), pp. 141–202.

[Sil+14] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. "Deterministic Policy Gradient Algorithms". In: *International Conference on Machine Learning (ICML)*. Ed. by Eric P. Xing and Tony Jebara. Vol. 32. Proceedings of Machine Learning Research 1. Bejing, China: PMLR, 2014, pp. 387–395.

[Sil+16] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. "Mastering the Game of Go with Deep Neural Networks and Tree Search". In: *Nature* 529.7587 (2016), pp. 484–489. DOI: 10.1038/nature16961.

[Sil+17] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. "Mastering the game of Go without human knowledge". In: *Nature* 550 (7676 2017), pp. 354–359. ISSN: 1476-4687. DOI: 10.1038/nature24270.

[SK06] Luis Sentis and Oussama Khatib. "A whole-body control framework for humanoids operating in human environments". In: *IEEE International Conference on Robotics and Automation (ICRA)*. Ed. by Normal Caplan and C.S. George Lee. 2006, pp. 2641–2648. DOI: 10.1109/ROBOT.2006.1642100.

[SKB12] Bruno Castro da Silva, George Konidaris, and Andrew G. Barto. "Learning Parameterized Skills". In: *International Conference on Machine Learning (ICML)*. Ed. by John Langford and Joelle Pineau. ICML'12. Edinburgh, Scotland: Omnipress, 2012, pp. 1443–1450. ISBN: 9781450312851.

[SLB09] Satinder Singh, Richard L. Lewis, and Andrew G. Barto. "Where Do Rewards Come From?" In: *Annual Conference of the Cognitive Science Society (CogSci)*. Ed. by N. Taatgen. 2009, pp. 2601–2606.

[SLD17] Evan Shelhamer, Jonathan Long, and Trevor Darrell. "Fully Convolutional Networks for Semantic Segmentation". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39.4 (2017), pp. 640–651. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2016.2572683.

[SLS99] Nadeem Ahmed Syed, Huan Liu, and Kah Kay Sung. "Handling concept drifts in incremental learning with support vector machines". In: *International Conference on Knowledge Discovery and Data Mining (KDD)*. Ed. by Usama Fayyad, Surajit Chaudhuri, and David Madigan. New York, New York, USA: ACM Press, 1999, pp. 317–321.

[Son+20] Xingyou Song, Wenbo Gao, Yuxiang Yang, Krzysztof Choromanski, Aldo Pacchiano, and Yunhao Tang. "ES-MAML: Simple Hessian-Free Meta Learning". In: *International Conference on Learning Representations (ICLR)*. Ed. by Dawn Song, Kyunghyun Cho, and Martha White. 2020.

[SPE18] Hubert Soyer, Drew Purves, and Lasse Espeholt. *Scalable agent architecture for distributed training*. 2018. URL: https://deepmind.com/blog/article/impala-scalable-distributed-deeprl-dmlab-30 (visited on 08/03/2020).

[SR20] Kai von Szadkowski and Simon Reichel. "Phobos: A tool for creating complex robot models". In: *Journal of Open Source Software* 5.45 (2020), p. 1326. DOI: 10.21105/joss.01326.

[SRD19] Stefan Scherzinger, Arne Roennau, and Rüdiger Dillmann. "Contact Skill Imitation Learning for Robot-Independent Assembly Programming". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Ed. by Dong Sun and Fumihito Arai. 2019, pp. 4309–4316.

[Sri+10] Niranjan Srinivas, Andreas Krause, Sham M. Kakade, and Matthias Seeger. "Gaussian Process Optimization in the Bandit Setting: No Regret and Experimental Design". In: *International Conference on Machine Learning (ICML)*. Ed. by Johannes Fürnkranz and Thorsten Joachims. ICML'10. Haifa, Israel: Omnipress, 2010, pp. 1015–1022. ISBN: 9781605589077.

[SS12] Freek Stulp and Olivier Sigaud. "Path Integral Policy Improvement with Covariance Matrix Adaptation". In: *International Conference on Machine Learning (ICML)*. Ed. by John Langford and Joelle Pineau. ICML'12. Edinburgh, Scotland: Omnipress, 2012, pp. 1547–1554. ISBN: 9781450312851.

[SS19] Olivier Sigaud and Freek Stulp. "Policy search in continuous action domains: An overview". In: *Neural Networks* 113 (2019), pp. 28–40. ISSN: 0893-6080. DOI: 10.1016/j.neunet.2019.01.011.

[Ste+04] Jochen J. Steil, Frank Röthling, Robert Haschke, and Helge Ritter. "Situated robot learning for multi-modal instruction and imitation of grasping". In: *Robotics and Autonomous Systems* 47.2 (2004). Robot Learning from Demonstration, pp. 129–141. ISSN: 0921-8890. DOI: 10.1016/j.robot.2004.03.007.

[STS12] Freek Stulp, Evangelos A. Theodorou, and Stefan Schaal. "Reinforcement Learning With Sequences of Motion Primitives for Robust Manipulation". In: *IEEE Transactions on Robotics* 28.6 (2012), pp. 1360–1370. ISSN: 1552-3098. DOI: 10.1109/TRO.2012.2210294.

[Stu+11] Freek Stulp, Evangelos A. Theodorou, Jonas Buchli, and Stefan Schaal. "Learning to grasp under uncertainty". In: *IEEE International Conference on Robotics and Automation (ICRA)*. Ed. by Zexiang Li and Yuan Fang Zheng. Shanghai, China, 2011, pp. 5703–5708.

[Sut+00] Richard S. Sutton, David A. McAllester, Satinder P. Singh, and Yishay Mansour. "Policy Gradient Methods for Reinforcement Learning with Function Approximation". In: *Advances in Neural Information Processing Systems*. Ed. by S. A. Solla, T. K. Leen, and K. Müller. MIT Press, 2000, pp. 1057–1063. URL: http://papers.nips.cc/paper/1713-policy-gradient-methods-for-reinforcement-learning-with-function-approximation.pdf.

[Sut88] Richard S. Sutton. "Learning to Predict by the Methods of Temporal Differences". In: *Machine Learning* 3.1 (1988), pp. 9–44. ISSN: 0885-6125. DOI: 10.1023/A:1022633531479.

[Sut90] Richard S. Sutton. "Integrated Architectures for Learning, Planning, and Reacting Based on Approximating Dynamic Programming". In: *International Conference on Machine Learning (ICML)*. Ed. by Bruce Porter and Raymond Mooney. San Francisco (CA): Morgan Kaufmann, 1990, pp. 216–224. ISBN: 978-1-55860-141-3. DOI: 10.1016/B978-1-55860-141-3.50030-4.

[SVL14] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. "Sequence to Sequence Learning with Neural Networks". In: *Advances in Neural Information Processing Systems*. Ed. by Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger. Curran Associates, Inc., 2014, pp. 3104–3112. URL: http://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks.pdf.

[SW05] Reza Shadmehr and Steven P. Wise. *The Computational Neurobiology of Reaching and Pointing*. MIT Press, 2005. ISBN: 0-262-19508-9.

[SYL13] Daniel L. Silver, Qiang Yang, and Lianghao Li. "Lifelong Machine Learning Systems: Beyond Learning Algorithms". In: *AAAI Spring Symposium: Lifelong Machine Learning*. Vol. SS-13-05. AAAI Technical Report. AAAI, 2013.

[Sze+14] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. "Intriguing properties of neural networks". In: *International Conference on Learning Representations (ICLR)*. Ed. by Aaron Courville, Rob Fergus, and Chris Manning. 2014. arXiv: 1312.6199.

[Tam+11] Minija Tamosiunaite, Bojan Nemec, Aleš Ude, and Florentin Wörgötter. "Learning to pour with a robot arm combining goal and shape learning for dynamic movement primitives". In: *Robotics and Autonomous Systems* 59.11 (2011), pp. 910–922. ISSN: 0921-8890. DOI: 10.1016/j.robot.2011.07.004.

[Tan+17] Voot Tangkaratt, Herke van Hoof, Simone Parisi, Gerhard Neumann, Jan Peters, and Masashi Sugiyama. "Policy Search with High-Dimensional Context Variables". In: *AAAI Conference on Artificial Intelligence*. Ed. by Satinder Singh and Shaul Markovitch. 2017, pp. 2632–2638.

[TBF05] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005. ISBN: 0262201623.

[TBS10a] Evangelos A. Theodorou, Jonas Buchli, and Stefan Schaal. "A Generalized Path Integral Control Approach to Reinforcement Learning". In: *Journal of Machine Learning Research* 11.104 (2010), pp. 3137–3181. URL: http://jmlr.org/papers/v11/theodorou10a.html.

[TBS10b] Evangelos A. Theodorou, Jonas Buchli, and Stefan Schaal. "Reinforcement learning of motor skills in high dimensions: A path integral approach". In: *IEEE International Conference on Robotics and Automation (ICRA)*. Ed. by Wesley Snyder and Vijay Kumar. 2010, pp. 2397–2403.

[Tes92] Gerald Tesauro. "Practical issues in temporal difference learning". In: *Machine Learning* 8 (3 1992), pp. 257–277. ISSN: 1573-0565. DOI: 10.1007/BF00992697.

[Tes95] Gerald Tesauro. "Temporal Difference Learning and TD-Gammon". In: *Communications of the ACM* 38.3 (1995), pp. 58–68. ISSN: 0001-0782. DOI: 10.1145/203330.203343.

[TET12a] Yuval Tassa, Tom Erez, and Emanuel V. Todorov. "Synthesis and stabilization of complex behaviors through online trajectory optimization". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Ed. by Anibal T. de Almeida, Urbano Nunes, and Eugenio Guglielmelli. 2012, pp. 4906–4913. DOI: 10.1109/IROS.2012.6386025.

[TET12b] Emanuel V. Todorov, Tom Erez, and Yuval Tassa. "MuJoCo: A physics engine for model-based control". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Ed. by Anibal T. de Almeida, Urbano Nunes, and Eugenio Guglielmelli. IEEE, 2012, pp. 5026–5033. ISBN: 978-1-4673-1737-5.

[Tha+17] Brijen Thananjeyan, Animesh Garg, Sanjay Krishnan, Carolyn Chen, Lauren Miller, and Ken Goldberg. "Multilateral surgical pattern cutting in 2D orthotropic gauze with deep reinforcement learning policies for tensioning". In: *IEEE International Conference on Robotics and Automation (ICRA)*.

Ed. by I-Ming Chen and Yoshihiko Nakamura. 2017, pp. 2371–2378. DOI: 10.1109/ICRA.2017.7989275.

[The16] Theano Development Team. "Theano: A Python framework for fast computation of mathematical expressions". In: *CoRR* abs/1605.02688 (2016). arXiv: 1605.02688 [cs.SC].

[Tib+14] Abraham Temesgen Tibebu, Bingbin Yu, Yohannes Kassahun, Emmanuel Vander Poorten, and Phuong Toan Tran. "Towards Autonomous Robotic Catheter Navigation Using Reinforcement Learning". In: *Joint Workshop on New Technologies for Computer/Robot Assisted Surgery (CRAS)*. Ed. by Leonardo Mattos, Paolo Fiorini, and Emmanuel Vander Porten. Genoa, Italy, 2014, pp. 163–166.

[Tim18] Tim Head et al. *scikit-optimize/scikit-optimize: v0.5.2*. Deposited at Zenodo. 2018. DOI: 10.5281/zenodo.1207017.

[TL05] Emanuel V. Todorov and Weiwei Li. "A generalized iterative LQG method for locally-optimal feedback control of constrained nonlinear stochastic systems". In: *American Control Conference*. Ed. by Sivasubramanya N. Balakrishnan. 2005, pp. 300–306.

[TL16] Lei Tai and Ming Liu. "Deep-learning in Mobile Robotics - from Perception to Control Systems: A Survey on Why and Why not". In: *CoRR* abs/1612.07139 (2016). URL: http://arxiv.org/abs/1612.07139.

[TLL16] Lei Tai, Shaohua Li, and Ming Liu. "A deep-network solution towards model-less obstacle avoidance". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Ed. by Il Hong Suh and Dong-Soo Kwon. 2016, pp. 2759–2764. DOI: 10.1109/IROS.2016.7759428.

[TM95] Sebastian Thrun and Tom M. Mitchell. "Lifelong robot learning". In: *Robotics and Autonomous Systems* 15.1 (1995), pp. 25–46. ISSN: 0921-8890. DOI: 10.1016/0921-8890(95)00004-Y.

[Tou11] Marc Toussaint. *Lecture Notes: Gaussian identities*. 2011. URL: https://ipvs.informatik.uni-stuttgart.de/mlr/marc/notes/gaussians.pdf (visited on 04/17/2020).

[TSB99] Matthew C. Tresch, Philippe Saltiel, and Emilio Bizzi. "The construction of movement by the spinal cord". In: *Nature Neuroscience* 2 (2 1999), pp. 162–167. DOI: 10.1038/5721.

[TSL00] Joshua B. Tenenbaum, Vin De Silva, and John C. Langford. "A Global Geometric Framework for Nonlinear Dimensionality Reduction". In: *Science* 290.5500 (2000), pp. 2319–2323.

[TSP20] Samuele Tosatto, Jonas Stadtmueller, and Jan Peters. "Dimensionality Reduction of Movement Primitives in Parameter Space". In: *CoRR* (2020). arXiv: 2003.02634 [cs.RO].

[UAS04] Holger Urbanek, Alin Albu-Schaffer, and Patrick van der Smagt. "Learning from demonstration: repetitive movements for autonomous service robotics". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Ed. by Kazuhiro Kosuge and Hajime Asama. Vol. 4. 2004, pp. 3495–3500. DOI: 10.1109/IROS.2004.1389957.

[Ude+10] Aleš Ude, Andrej Gams, Tamim Asfour, and Jun Morimoto. "Task-Specific Generalization of Discrete and Periodic Dynamic Movement Primitives". In: *IEEE Transactions on Robotics* 26.5 (2010), pp. 800–815.

[Ude+14] Aleš Ude, Bojan Nemec, Tadej Petrić, and Jun Morimoto. "Orientation in Cartesian space dynamic movement primitives". In: *IEEE International Conference on Robotics and Automation (ICRA)*. Ed. by Ning Xi and William R. Hamel. 2014, pp. 2997–3004.

[Uni20] Universal Robots A/S. *Meet the CB3 Family*. 2020. URL: https://www.universal-robots.com/cb3/ (visited on 05/06/2020).

[Vas+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aiden N. Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention is All you Need". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Curran Associates, Inc., 2017, pp. 5998–6008. URL: http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf.

[VB05] Miomir Vukobratović and Branislav Borovac. "Zero-Moment Point - Thirty Five Years of its Life". In: *IEEE-RAS International Conference on Humanoid Robots (Humanoids)* (2005), pp. 157–173. URL: http://www.cs.cmu.edu/~cga/legs/vukobratovic.pdf.

[Vin+19] Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John P. Agapiou, Max Jaderberg, Alexander S. Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yury Sulsky, James Molloy, Tom L. Paine, Caglar Gulcehre, Ziyu Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver. "Grandmaster level in StarCraft II using multi-agent reinforcement learning". In: *Nature* 575 (7782 2019), pp. 350–354. DOI: 10.1038/s41586-019-1724-z.

[Vla+09] Nikos Vlassis, Marc Toussaint, Georgios Kontes, and Savas Piperidis. "Learning model-free robot control by a Monte Carlo EM algorithm". In: *Autonomous Robots* 27.2 (2009), pp. 123–130. ISSN: 1573-7527. DOI: 10.1007/s10514-009-9132-0.

[Wan+16a] Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Rémi Munos, Koray Kavukcuoglu, and Nando de Freitas. "Sample Efficient Actor-Critic with Experience Replay". In: *International Conference on Learning Representations (ICLR)*. Ed. by Samy Bengio and Brian Kingsbury. 2016.

[Wan+16b] Ziyu Wang, Frank Hutter, Masrour Zoghi, David Matheson, and Nando De Freitas. "Bayesian Optimization in a Billion Dimensions via Random Embeddings". In: *Journal of Artificial Intelligence Research* 55.1 (2016), pp. 361–387. ISSN: 1076-9757.

[Wan+16c] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. "Dueling Network Architectures for Deep Reinforcement Learning". In: *International Conference on Machine Learning (ICML)*. Ed. by Maria Florina Balcan and Kilian Q. Weinberger. Vol. 48. Proceedings of Machine Learning Research. New York, New York, USA: PMLR, 2016, pp. 1995–2003.

[Wat89] Christopher Watkins. "Learning from Delayed Rewards". PhD thesis. Cambridge, UK: King's College, 1989. URL: http://www.cs.rhul.ac.uk/~chrisw/new_thesis.pdf.

[WdK12] Malte Wirkus, José de Gea Fernández, and Yohannes Kassahun. "Realizing Target-Directed Throwing With a Real Robot Using Machine Learning Techniques". In: *Workshop on Evolutionary and Reinforcement Learning for Autonomous Robot Systems*. Ed. by Stéphane Doncieux. 2012, pp. 37–43.

[Wei+19] Richard Wei, Dan Zheng, Marc Rasi, and Bart Chrzaszcz. *Differentiable Programming Manifesto*. 2019. URL: https://github.com/apple/swift/blob/master/docs/DifferentiableProgramming.md (visited on 02/11/2020).

[Wie+14] Daan Wierstra, Tom Schaul, Tobias Glasmachers, Yi Sun, Jan Peters, and Jürgen Schmidhuber. "Natural Evolution Strategies". In: *Journal of Machine Learning Research* 15 (2014), pp. 949–980. URL: http://jmlr.org/papers/v15/wierstra14a.html.

[Wik18] Wikipedia contributors. *Glossary of climbing terms — Wikipedia, The Free Encyclopedia*. 2018. URL: https://en.wikipedia.org/w/index.php?title=Glossary_of_climbing_terms&oldid=859314596 (visited on 10/06/2018).

[Wil92] Ronald J. Williams. "Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning". In: *Machine Learning* 8.3–4 (1992), pp. 229–256. ISSN: 0885-6125. DOI: 10.1007/BF00992696.

[Wir14] Malte Wirkus. "Towards Robot-independent Manipulation Behavior Description". In: *International Workshop on Domain-Specific Languages and models for ROBotic systems (DSLRob)*. 2014.

[WS01] Christopher K. I. Williams and Matthias Seeger. "Using the Nyström Method to Speed Up Kernel Machines". In: *Advances in Neural Information Processing Systems*. Ed. by T. K. Leen, T. G. Dietterich, and V. Tresp. MIT Press, 2001, pp. 682–688. URL: http://papers.nips.cc/paper/1866-using-the-nystrom-method-to-speed-up-kernel-machines.pdf.

[WS13] Wen Hao Lui and Ashutosh Saxena. "Tangled: Learning to untangle ropes with RGB-D perception". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Ed. by Shigeki Sugano and Makoto Kaneko. 2013, pp. 837–844. DOI: 10.1109/IROS.2013.6696448.

[Xse20] Xsens Technologies B.V. *MTw Awinda*. 2020. URL: https://www.xsens.com/products/mtw-awinda (visited on 05/27/2020).

[Xu+17] Huazhe Xu, Yang Gao, Fisher Yu, and Trevor Darrell. "End-To-End Learning of Driving Models From Large-Scale Video Datasets". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Ed. by Jim Rehg, Yanxi Liu, Ying Wu, and Camillo Taylor. 2017.

[Yan+17] Brian Yang, Grant Wang, Roberto Calandra, Daniel Contreras, Sergey Levine, and Kristofer Pister. "Learning Locomotion Primitives from Contextual Bayesian Optimization". In: *Workshop on Bayesian Optimization, Advances in Neural Information Processing Systems*. Ed. by José Miguel Hernández-Lobato, Javier Gonzalez, and Ruben Martinez-Cantin. Long Beach, USA, 2017. URL: https://pdfs.semanticscholar.org/06f2/c307c886f97e95a8f4cba99475f946e2c085.pdf.

[YCW19] Yuhui Yuan, Xilin Chen, and Jingdong Wang. "Object-Contextual Representations for Semantic Segmentation". In: *CoRR* (2019). arXiv: 1909.11065 [cs.CV].

[YKL17] Chuanyu Yang, Taku Komura, and Zhibin Li. "Emergence of human-comparable balancing behaviours by deep reinforcement learning". In: *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*. Ed. by Tamim Asfour. 2017, pp. 372–377. DOI: 10.1109/HUMANOIDS.2017.8246900.

[Yu+18] Tianhe Yu, Chelsea Finn, Annie Xie, Sudeep Dasari, Tianhao Zhang, Pieter Abbeel, and Sergey Levine. "One-Shot Imitation from Observing Humans via Domain-Adaptive Meta-Learning". In: *International Conference on Learning Representations (ICLR), Workshop Track Proceedings*. Ed. by Iain Murray, Marc'Aurelio Ranzato, and Oriol Vinyals. 2018.

[ZBH07] Franziska Zacharias, Christoph Borst, and Gerd Hirzinger. "Capturing robot workspace structure: representing robot capabilities". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Ed. by Edward Grant and Thomas C. Henderson. 2007, pp. 3229–3236.

[ZDM19] Hongyi Zhang, Yann N. Dauphin, and Tengyu Ma. "Residual Learning Without Normalization via Better Initialization". In: *International Conference on Learning Representations (ICLR)*. Ed. by Sergey Levine, Karen Livescu, and Shakir Mohamed. 2019.

[Zha+18a] Tianhao Zhang, Zoe McCarthy, Owen Jow, Dennis Lee, Xi Chen, Ken Goldberg, and Pieter Abbeel. "Deep Imitation Learning for Complex Manipulation Tasks from Virtual Reality Teleoperation". In: *IEEE International Conference on Robotics and Automation (ICRA)*. Ed. by Kevin Lynch. 2018, pp. 5628–5635.

[Zha+18b] Leidi Zhao, Yiwen Zhao, Siddharth Patil, Dylan Davies, Cong Wang, Lu Lu, and Bo Ouyang. "Robot Composite Learning and the Nunchaku Flipping Challenge". In: *IEEE International Conference on Robotics and Automation (ICRA)*. Ed. by Kevin Lynch. IEEE, 2018, pp. 3160–3165.

[Zhu+17] Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J. Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. "Target-driven visual navigation in indoor scenes using deep reinforcement learning". In: *IEEE International Conference on Robotics and Automation (ICRA)*. Ed. by I-Ming Chen and Yoshihiko Nakamura. 2017, pp. 3357–3364. DOI: 10.1109/ICRA.2017.7989381.

[ZP12] Vladimir M. Zatsiorsky and Boris I. Prilutsky. *Biomechanics of Skeletal Muscles*. Human Kinetics, 2012. ISBN: 9781450428842.

[Zuc+11] Matt Zucker, Nathan Ratliff, Martin Stolle, Joel Chestnutt, J Andrew Bagnell, Christopher G. Atkeson, and James Kuffner. "Optimization and learning for rough terrain legged locomotion". In: *International Journal of Robotics Research* 30.2 (2011), pp. 175–191. DOI: 10.1177/0278364910392608.