Towards Computational Design of Shape and Control for Rigid Robots

by

Jie Xu

B.Eng., Tsinghua University (2016)
S.M., Massachusetts Institute of Technology (2019)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2022

Author .....
Department of Electrical Engineering and Computer Science
August 26, 2022

Certified by .....
Wojciech Matusik
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by .....
Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

Towards Computational Design of Shape and Control for Rigid Robots

by

Jie Xu

Submitted to the Department of Electrical Engineering and Computer Science
on August 26, 2022, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

Designing a performing robotic system for given tasks is traditionally labor-intensive for finding the optimal configurations of its hardware shape and/or software controller. The underlying coupling of the hardware shape and the software control of a robot results in an enormous parameter space involving both discrete parameters (*i.e.*, topology structure of the robot) and continuous parameters (*i.e.*, morphological dimensions of each robot link, the control parameters), optimizing which traditionally requires significant amounts of expert knowledge from roboticists and many manual design iterations. Intending to automate the robot design process, computational robot design has attracted increasing attention from robotics, graphs, and artificial intelligence researchers. However, building a general computational robot design process is extremely hard due to several challenging problems, including but not limited to representation, performance evaluation, and optimization problems. This thesis identifies some key challenges in the computational robot design pipeline and proposes our corresponding solutions. We first take the manipulator design as an example to present our robot design representations for both discrete and continuous robot shape parameters. With the proposed robot representations, we then explore the corresponding robot optimization techniques. In this part, we first introduce how we leverage differentiable simulators to efficiently optimize the robot control policy with the robot configuration fixed. Next, we delve into the more complicated co-design problems requiring optimization of both the shape and control of a robot. We present two novel algorithms for optimizing discrete shape parameters and continuous shape parameters, respectively. Finally, we step further toward the more realistic multi-objective robot design problems and present our solutions for finding a set of Pareto-optimal robot designs trading off multiple different objectives and tasks. We conclude this thesis by envisioning an ultimate computational design pipeline and discussing open research directions toward this ultimate goal.

Thesis Supervisor: Wojciech Matusik
Title: Professor of Electrical Engineering and Computer Science

Acknowledgments

Six years ago, when I started my Ph.D. journey at MIT, everything surrounding me was new – new country, new school, new people, and new research topics. The pandemic makes this Ph.D. journey even special. Throughout this incredible journey, I am so lucky to meet so many unbelievable friends and supervisors. It is impossible to complete my Ph.D. without the help I have received from countless people, and I would like to express my deepest gratitude to all who supported me either technically or emotionally.

First and foremost, I would like to thank my advisor, Prof. Wojciech Matusik, for offering me this special journey ticket to MIT and supervising me without any reservations. Wojciech always encourages me to conduct impactful research and not get distracted from low-hanging fruits. There were countless times when Wojciech spent hours with me discussing and debating different next-project ideas and sharing his unique thoughts on the projects. I still remember the nights Wojciech and I spent over zoom whiteboard to discuss the deformation-based shape representation ideas when preparing the submission to RSS 2021. I would also express my deepest appreciation to Wojciech for his extreme patience with the hybrid UAV project. That was the first project led by myself ever in my academic life, and it was also my first attempt to get outside my research comfort zone and step into the robotics field. It was a two-year-long project where I experienced through tons of failures on different parts of the project. I was near to giving up many times, but Wojciech showed this unbelievable tolerance for my failures and always motivated me to never give up. Without his patience and motivation, I may never knock out the door of robotics and discover this incredible field of research that deserved my lifelong exploration.

I would also like to thank Prof. Daniela Rus and Prof. Pulkit Agrawal for serving on my thesis committee. I thank Daniela for always sharing her broad knowledge and broad view to us, for always encouraging us to "keep the gradients", and for letting me know nothing is impossible as long as I pay enough efforts on the projects. I thank Prof. Pulkit Agrawal for offering his professional advice from a perspective

of a learning expert. The biggest lesson I learned from Pulkit is to always conduct high-standard research and always keep improving the paper even if it already gets accepted and published.

I would also express my special thanks to Prof. Shinjiro Sueda for offering his generous help and answering my endless questions about developing a differentiable articulated rigid body simulator. Shin is the best ever collaborator I have ever seen during my PhD. Not only can he share with me his expertise in simulation, he also even implements code for me to illustrate and validate his mathematical derivation and provides me with responsive support on paper and rebuttal writing. I am absolutely sure that I cannot have those publications happening so smoothly without Shin's sincere mentoring. I am also thankful to my academic advisor, Prof. Stefanie Jegelka, for offering her emotional support and career guidance whenever I need them. I would also like to thank Prof. Shi-Min Hu for taking me into the academic research and mentoring me when I was an undergraduate student at Tsinghua University.

My thanks should also go to the senior students in the lab when I first came to MIT. Big thanks to Adriana Schulz, Bo Zhu, and Tao Du for taking me to conduct the very first two projects at MIT. I would also extend my gratitude to Desai Chen for helping me fit into the lab. I would also be thankful to Liang Shi and Yuanming Hu for all the enjoyable moments such as making the "flying cups" in the lab and working hard on debugging the Raft system of 6.824.

My Ph.D. would not have been possible without the support from all my collaborators: Michael Foshey, Beichen Li, Allan Zhao, Andrew Spielberg, Yunsheng Tian, Pingchuan Ma, Tao Chen, Lara Zlokapa, Viktor Makoviychuk, Miles Macklin, Yashraj Narang, Fabil Ramos, Animesh Garg, Sangwoon Kim, Alberto Rodriguez, Mina Konakoivć Luković, Josephine Hughes, Yifei Li, Kui Wu, Jagdeep Singh Bhatia, Holly Jackson, Juan Salazar, Wei Wang, Michal Piovarĉi, Timothy Erps, Vahid Babaei, Piotr Didyk, Szymon Rusinkiewicz, and Bernd Bickel.

I would also extend my sincere to the reset of the CDFG members: Yagiz Aksoy, Bolei Deng, Minghao Guo, Ryan Gulland, Alexandre Kasper, Changil Kim, Petr

Kellnhofer, Yiyue Luo, Yichen Li, James Minor, Liane Makatura, Ruben Castro Ornelas, Tae-Hyun Oh, Mélina Skouras, Ruitao Su, Subra Sundaram, Wan Shou, Bohan Wang, Harrison Wang. We had so many BBQ happy hours at MIT sailing pavilion, end-of-year holiday group dinners, and CGGAR retreats. I believe those would be my unforgettable moments.

I also want to thank the current and past members of the MIT table tennis team for the practices in the T-club lounge and the games we fought together for in Division and Regional competitions.

I would also like to express my thanks to all my friends during my Ph.D. journey. This journey must be much less fun without the trips, casual chat, board games, go kart, and hiking with my friends. I would extend my special thank to Yuzhe Yang and Luxin Zhang for travelling to many incredible places with us and host those Mario Party nights at home. I would also appreciate Yiyue Luo and Zeyu Wu for the countless Poker Card Friday nights, and I believe I would never forget the "disguise" of each of us when drawing the cards. I am also very fortunate to have my dudes – Wenbo Tao, Songtao He, and Shichao Yue. I'll remember those joyful moments when we played billiard on the first floor in SidPac and supported Celtics at TD Garden. Last but not least, I also want to thank Lijie Fan and Tianhong Li for the teammate nights of "Honor of Kings" at the dorm.

Next, I would like to thank my girlfriend, Xia Xiao. We met in February 2019, and it has been our fourth year. All the publications in this thesis happened after we got together, and you accompanied me so many days and nights when I was on those crazy conference deadlines. You are my lucky star, and my Ph.D. would be incomplete without your appearance and unconditional support.

Finally, I would like to extend my biggest thanks to my parents for their permanent support and unconditional love. They always support and trust each of my decisions. And whenever I felt depressed, they always encouraged me to relax from the great pressure. They always tell me that study and career are just a small portion of your life and that physical and mental health is the most important piece of your whole life.

They also always encourage me never to get disappointed when I encounter failures since I am always the best in their eyes and can overcome any difficulty.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Most organisms in nature have been "evolved" over millions of years to be adept at their unique skills. For example, a lizard hunting prey may need to be proficient at climbing trees and running; a duck in migration needs to be able to swim; a human hurdler must be fast at running along bends and straightaways. The core to what makes these organisms so capable of their particular skills is their specialized body mechanisms and highly optimized brain systems with complex neuron arrays. Because form informs function and vice versa, we similarly should expect that the best-performing robots for different desired tasks have remarkably different designs.

The design, control, and construction of performing robotic systems for given tasks are traditionally labor-intensive for finding the optimal configurations of its hardware shape and/or software control. The underlying coupling of the hardware shape and the software control of a robot results in an enormous parameter space involving both discrete parameters (*i.e.*, topology structure of the robot) and continuous parameters (*i.e.*, morphological dimensions of each robot link, the control parameters), optimizing which today requires significant amounts of expert knowledge from roboticists and many manual design iterations. For example, as shown in Figure 1-1, hardware components and control algorithms are typically constructed sequentially based on the empirical experience of the robot designer (*Design Phase*). Then the designed robot needs to be manufactured, and physical testing will be conducted to evaluate its

performance (*Evaluation Phase*). Based on the evaluated performance, the designer has to come back to the first stage to improve the robot's shape and control accordingly (*Human Improvement Phase*). A performing robotic system typically requires many such slow and manual iterations.

Intending to automate the robot design process, computational robot design has attracted increasing attention from robotics, graphs, and artificial intelligence researchers. Computational robot design first replaces the *Design Phase* with the *Computational Design Representation* to be able to numerically represent the design of a robot by a set of shape and control parameters. Next, instead of manufacturing and evaluating the robots in real experiments, it computationally evaluates the robots inside the digital simulators. Finally and most importantly, the manual improvement phase is replaced by a computational optimization algorithm to automatically improve the robot parameters. The replacements of these three stages then successfully build a fully automated computational robot design process. Not only a computational robot design automation can significantly improve productivity by freeing up the human labor from this time-consuming process, but it may also find potential robot designs beyond human expertise limitations. Despite its attractive advantages, building a general computational robot design process is extremely hard due to several challenging problems, including but not limited to representation, optimization, and performance evaluation problems.

In this thesis, we tackle the problems of computational design of task-driven robots, and mainly focus on the context of *articulated rigid robots*. We identify some of the key challenges overlooked by previous computational robot design works and propose our corresponding solutions for them in the hope of making a pace towards the ultimate goal of a general computational robot design pipeline.

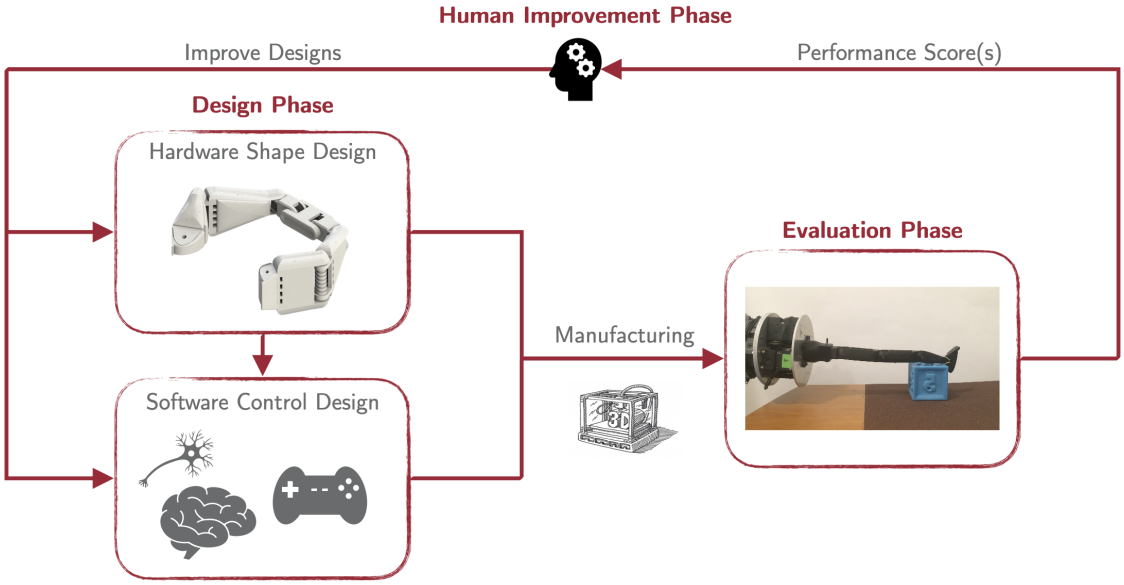


Figure 1-1: **Traditional robot design pipeline.** Traditionally, a robotic expert need to first empirically determine the robot hardware shape and software control strategy (*Design Phase*). Then she needs to manufacture the design and test its performance in real experiments (*Evaluation Phase*). Based on the performance, she has to come back to improve the design based on her personal experience (*Human Improvement Phase*). A successful design typically requires tens or hundreds of such manual design iterations.

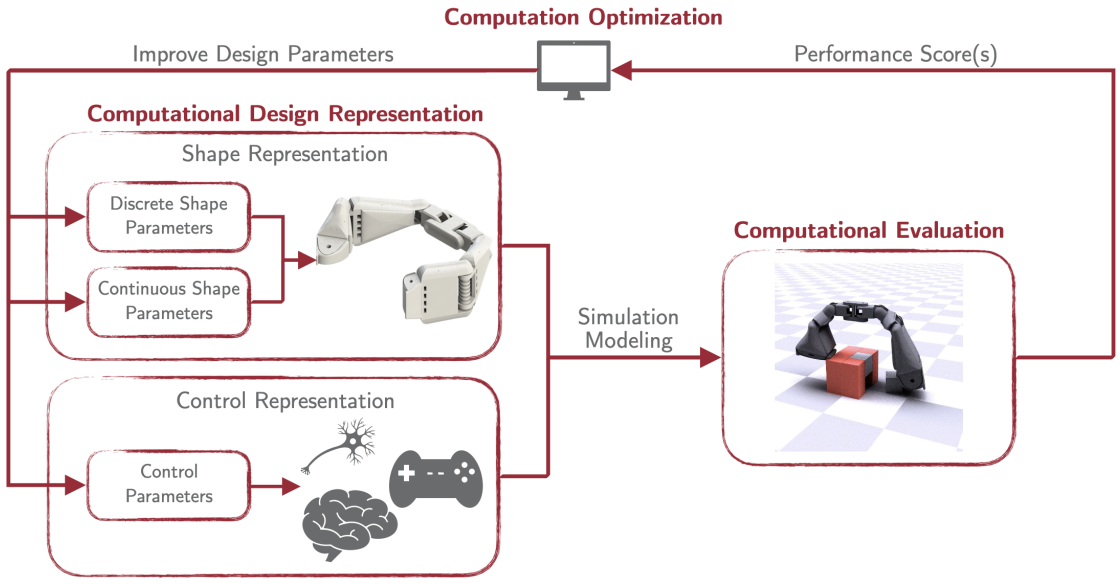


Figure 1-2: **Computational robot design pipeline.** Computational robot design improve the process by replacing the three phases with *Computational Design Representation*, *Computational Evaluation*, and *Computational Optimization* modules.

1.1 Key Challenges in Computational Robot Design

1.1.1 Hardware Shape Representation

To achieve the goal of computationally constructing the robots, the first significant challenge is how to represent the robot shape designs in a computational way. A complete description of a robot shape should contain both discrete shape parameters and continuous shape parameters. The discrete parameters may define the topological structure of the robot (*e.g.*, number of body links, how links connect together), and the types of the joints connecting body links. On the other hand, the continuous shape parameters usually contain the morphological dimensions of each robot link (*i.e.*, length, radius, width), the placements of the connection joints, stiffness and damping coefficients of the joints, and the material parameters of the robot materials.

Traditionally, people overlooked this shape representation problem and typically used arbitrary connections of primitive shapes (*e.g.*, cylinders or cuboids) to represent the robot geometry. However, while easy to optimize, these primitives often over-simplify the desired shape and are insufficient to model complex robot designs. Furthermore, arbitrarily connecting different pieces of components also unintentionally produces robot designs that are infeasible in real manufacturing. Those limitations of the existing works indicate us to find a proper way to represent the robot shape so as to concisely and comprehensively structure this hybrid and vast robot shape space while being amenable to optimization.

1.1.2 Control Representations

Once the robot hardware configuration is determined, the next thing we need to consider for a robot is its controller. As a robot's brain, the robotic control strategy plays a crucial role in its task performance. The controller of a robot can usually be represented in two approaches.

Traditionally, researchers define the control as an open-loop state-action trajectory and then, during execution, apply classic feedback control algorithms (*e.g.*, PID,

LQR) to control the robot to follow the predefined open-loop trajectory. However, the design of the feedback control algorithm typically relies on a deep understanding of the task-specific control strategy structure; thus, the design philosophy can hardly be generalized to more complex robot problems. More recently, neural network policy has become an increasingly popular alternative for controller representation due to its feedback nature and demonstrated generalizability across various complex control problems (*e.g.*, in-hand manipulation control tasks). Therefore, selecting the proper control representation for the best task performance and robustness is one of the challenging system design choices in a computational robot design process.

1.1.3 Robot Performance Evaluation

In order to optimize the desired robot designs, we need the performance metrics to guide the optimization algorithms. Effective performance metrics must reflect human subjective evaluations of different robot designs. The robot performance can usually be represented by either a *single objective* or *multiple different objectives*. While single-objective representation is simple to start with, most physical tasks in the real world are evaluated by multiple metrics. For example, when we evaluate a driver's driving skill, we will consider both the speed and safety of her driving. A similar analogy exists in robotic control problems. Robots in real life are versatile, general-purpose, and rarely designed for maximizing their performance on a single objective only. For instance, when designing a control policy for a running quadruped robot, we need to consider two conflicting objectives: running speed and energy efficiency. When multiple conflicting objectives exist, there is no single optimal solution that is optimal in every objective, instead we face a much more challenging task to find a Pareto set of solutions making different trade offs among different objectives. Therefore, in a computational robot design process, we have to decide on the proper performance evaluation to make the best trade off between simplicity of the optimization algorithm and the comprehensiveness of the robot evaluation.

1.1.4 Single-Objective Robot Optimization

Given the performance metrics and the robot representation, the next challenge is finding an efficient algorithm to optimize the best robot in the represented robot design space under the selected performance evaluation. We first constrain our problems within single-objective settings. We argue that even with a single objective, robot optimization is nontrivial.

The very first question is that with a fixed robot hardware configuration, how to optimize a control policy for it efficiently? Among various policy optimization techniques, reinforcement learning (RL) has been a particularly successful tool for learning policies for complex robotic systems using only high-level reward definitions. Despite this success, RL requires large amounts of training data to approximate the policy gradient, making learning expensive and time-consuming. The recent development of differentiable simulators opens up new possibilities for accelerating the learning and optimization of control policies via analytical gradient-based approaches. However, the well-known local minima problem notoriously limits the application of differentiable simulation on complex tasks. Therefore, an interesting question is, can we actually combine reinforcement learning and differentiable simulation to achieve the best of both worlds on the policy optimization problems?

While the controller serves as a brain of a robot, robot hardware actually plays an equally important role as its control algorithm in its task performance. As an example, a quadruped with longer legs can be easily controlled to run faster than a quadruped with shorter legs, no matter how optimal the control algorithm of the short-leg quadruped is. Thus, one meaningful question for us is how to effectively co-optimize the robot shape and robot control simultaneously to achieve the best performance. Despite its importance, an automated computational algorithm for robot co-design is still an active and challenging research question. This is because the hardware shape of a robot introduces an enormous and mixed optimization space involving both discrete parameters and continuous parameters as we discussed above, which imposes an intractable search space on the classical optimization algorithms.

1.1.5 Multi-Objective Robot Optimization

The challenges brought by a multi-objective performance evaluation are even more substantial. While in single-objective problems, the main challenge comes from the enormous and mixed search space for the optimization algorithms, the multi-objective performance evaluation further increases the dimension of the optimal solution space for the optimizers. This is because, in a *multi-objective* problem, different objectives may conflict to each other. In such cases, multiple optimal solutions exist depending on the chosen trade-off between these metrics, known as the *Pareto* set. Take the quadruped control problem as an example again. One effective policy may prefer high speed at the cost of lower energy efficiency, whereas another optimal policy might prefer high energy efficiency at the cost of lower speed.

Most robotic works simplify this multi-objective problem by converting it into a single scalar objective as a weighted sum of different objectives and then finding the optimal solution for that specific preference weight setting. However, such simplification relies on a set of pre-selected preference weights of objectives, which is typically unavailable in most cases. Thus the users typically need to refine those weights back and forth to achieve the desired behaviors. Besides, if the task and the desired preference among objectives change, the computed solution immediately becomes sub-optimal, and the users need to re-run the expensive optimization for the new set of preference weights from scratch. Thus we desire to invent novel optimization algorithms for multi-objective robot problems, which are capable of finding the *Pareto Optimal* set of solutions. A human then has the full flexibility to select the preference among different metrics based on the specific task, and this determines the corresponding optimal policy in that particular scenario.

1.2 Thesis Overview

In this thesis, we demonstrate our solutions for the aforementioned challenges in a computational robot design pipeline. In Chapter 2, we review the previous works

related to the topics of this thesis. In Chapter 3, we take the manipulator design as an example and present our compact and expressive robot shape representations for both discrete and continuous robot parameters that are amenable to optimization [2, 3]. Our robot shape representations take the idea of using graph grammar for discrete robot structure space and leveraging cage-based deformation to define the continuous shape morphology space. With the proposed robot shape representations, we then explore the corresponding robot optimization techniques. In Chapter 4, we first introduce how we develop a differentiable simulator to help efficiently optimize the tactile-based control policy for a manipulator robot with a fixed hardware configuration [4], and then we demonstrate how to effectively combine the differentiable simulation and reinforcement learning together to further speed up the policy learning process [5]. In Chapter 5, we explore the more complicated co-design problems requiring optimization of both shape and control of a robot. We presents two novel algorithms for co-design purpose: a Graph Heuristic Search algorithm which utilizes a learning-based heuristic function to guide the search for the optimal combination of the discrete shape structures and control for terrestrial robots [6]; and an end-to-end differentiable framework to efficiently co-optimize the continuous shape morphology and control for manipulator robots [2]. Finally in Chapter 6, we step further towards the more sophisticated multi-objective robot design problems and presents our solutions on multi-objective control policy optimization problems [7] and multi-objective robot co-design problems [8]. Our proposed multi-objective robot design algorithms search for a set of Pareto-optimal robot designs trading off multiple different objectives and tasks.

Chapter 2

Related Work

2.1 Robot Shape Representation and Parameterization

Computational design of robot shape typically requires a compact and expressive representation of shape topological structure among links (*e.g.*, number of links and how links connect together) and continuous shape geometrical morphology (*e.g.*, size). For the shape topological structure, a graph representation has been naturally adopted [9, 10, 11]. However previous works usually allow the different parts can be connected arbitrarily, which results in a physically invalid robot design. For example, two parts may be connected at the middle of their body, or too many parts art connected to one single part. As a part of computational design for the robots that can be manufactured in real, we need to consider whether a connection we need to impose more physics constraints on the topological design space to make sure that all the robot design can be represented are physically feasible.

There are more variants on the continuous geometrical morphology representation side. Most existing works on robot co-design use simple primitive shapes to approximate the geometry of each robot link so that they can optimize the parameters easily [12, 13, 14, 15, 16, 17, 18, 19, 20, 21]. However, such over-simplified morphology

parameterization based on primitive shapes precludes the possibility of generating complex geometric morphology for the robot. It is inadequate especially when the geometry does affect the task dynamics with a rich amount of contact, such as in in-hand manipulation tasks. CAD-based parameterization is one of the options to support natural organic shapes, but it typically suffers from its slow inference speed and non-differentiability. Schulz et al. [22] proposed an interactive system for CAD models with an expensive precomputation cost. Hafner et al. [23] developed a differentiable parameterization of CAD models for FEM simulation, but the differentiability is not preserved in our rigid, multi-body simulation setting. Furthermore, CAD-based parameterization requires extra expertise to select and constrain each parameter in order to preserve model manufacturability throughout the optimization. It is also non-trivial to support the connectivity constraints required in an articulated robot structure. Truncated Signed Distance Functions (TSDF) and mesh-based parameterization are used by Ha et al. [24] to optimize the shape of a free-form gripper. These methods only work for a single body system, introduce a large number of parameters, and usually do not result in natural organic shapes. Compared to these, our deformation-based parameterization allows us to have a constrained but expressive design space for natural shapes and seamlessly works for articulated robot designs.

2.2 Differentiable Physics-Based Simulation

Physics-based simulation has been widely used for various robotic applications [25, 26, 27]. Among them, differentiable physics-based simulators have gained increasing popularity recently since their differentiability facilitates efficient gradient-based optimization for robotic control. Due to their inherent differentiability, neural network based simulators have also been proposed to approximate physics [28, 29, 30, 31]. However, these works sacrifice generality and accuracy of physics for differentiability of the neural network. Differentiable simulators have been developed for rigid bodies [32, 33, 34, 35, 36, 37], soft bodies [38, 39, 40, 41, 42, 43], and cloth [44, 45]. Handling contact response is the most important task in building a differentiable physics

simulator. Several approaches have been proposed for making the frictional contact response differentiable such as differentiation of the coefficient matrices and vectors of the linear complementarity problem arising from collisions [32, 33] or those that use impulse-based velocity stepping methods [34]. However, the discontinuity stemming from such approaches can cause difficulties for contact-rich tasks. Geilinger et al. [35] proposed a differentiable penalty-based frictional contact model. However, their contact model was only demonstrated to work for contact between the robot and a stationary surface (*e.g.*, ground and walls), and the simulation uses stiff springs to approximate articulation.

2.3 Computational Robot Design for Single Objective

2.3.1 Control Optimization and Learning

Traditionally, an open-loop state-action trajectory is optimized via trajectory optimization for a robot and classical feedback control is applied to control the robot stablizing around the open-loop trajectory [46, 47, 48, 49].

More recently, neural network control policies have been explored to control the robot movements. Deep reinforcement learning has become a prevalent tool for learning control policies for systems ranging from robots [50, 51, 52, 53, 54, 55], to complex animation characters [56, 57, 58, 1]. Model-free RL algorithms treat the underlying dynamics as a black box in the policy learning process. Among them, on-policy RL approaches [59, 60] improve the policy from the experience generated by the current policy, while off-policy methods [61, 62, 63, 64] leverage all the past experience as a learning resource to improve sample efficiency. On the other side, model-based RL methods [65, 66, 67, 68] have been proposed to learn an approximated dynamics model from little experience and then fully exploit the learned dynamics model during policy learning.

The recent development of differentiable simulators enables the optimization of control policies via the provided gradient information. Backpropagation Through Time (BPTT) [69] has been widely used in previous work to showcase differentiable systems [40, 41, 44, 42, 38]. However, the noisy optimization landscape and exploding/vanishing gradients in long-horizon tasks make such straightforward first-order methods ineffective. A few methods have been proposed to resolve this issue. [70] present a sample enhancement method to increase RL sample-efficiency for the simple MuJoCo Ant environment. However, as the method follows a model-based learning framework, it is significantly slower than state-of-the-art on-policy methods such as PPO [60]. [71] propose interleaving a trajectory optimization stage and an imitation learning stage to detach the policy from the computation graph in order to alleviate the exploding gradient problem. They demonstrate their methods on simple control tasks (*e.g.,* stopping a pendulum). However, gradients flowing back through long trajectories of states can still create challenging optimization landscapes for more complex tasks. Furthermore, both methods require the full simulation Jacobian, which is not commonly or efficiently available in reverse-mode differentiable simulators.

2.3.2 Control and Shape Co-Design

Co-design of robots typically involves optimizing geometry, mass properties, and control parameters, and involves of optimizing both discrete shape topology parameters and continuous shape and control parameters. For articulated robot co-design, one must consider extra variables for kinematics and dynamics relationships among links, such as joint and body translations/orientations.

To optimize the discrete topology of the robot, evolutionary approaches and search-based approaches have been adopted. Evolutionary approaches apply evolutionary algorithms to iteratively improve robot form and behavior across generations. Early notable work in this space includes [10] and [72], which evolved rigid robot morphologies and controllers to produce agile creatures. It was recently demonstrated that such techniques could be applied to the robust space of neural network con-

trollers [11]. Such techniques have also been applied to soft robots [73, 74, 75], evolving robots over geometry, actuation, and open-loop control. Evolutionary approaches lead to high diversity but poor convergence. Search-based methods can both handle combinatorial topology search and still have promising efficiency. In these methods, discrete searches of joint and limb configurations are guided by heuristic functions [76].

To optimize only the continuous shape geometrical morphology parameters, analytical methods are the main trend. Analytical methods use model gradients in order to inform search. Rigid co-optimization over continuous morphological and control parameters has been extensively studied over the last decade [18, 77, 12, 78, 13, 14, 16, 17]. Learning based methods have attracted more attention these days, where the Reinforcement Learning framework is extended with the ability to optimize the continuous morphology parameters [19, 20, 21]. However, the limitations of all of those existing methods is the the limited expressivity of the primitive shape representation as discussed in previous sections.

2.4 Computational Robot Design for Multiple Objectives

2.4.1 Multi-Objective Optimization

Multi-objective optimization algorithms have been successfully deployed in a wide variety of engineering domains, including material design [79], automotive engineering [80], thermodynamics [81], and medicine [82], to name a few. Core to these applications is the development of search algorithms that can retrieve dense Pareto fronts that are close to the ground-truth, with high sample efficiency. Two popular categories of strategies exist. The first is evolutionary algorithms; see [83] for an introduction. Popular methods in this space include NGSA-II [84], NEAT [85], CMA-ES [86], and MOEA/D [87]. These algorithms employ principled heuristics to

efficiently trade off exploration of the design space with exploitation of the estimated Pareto front. Contrasted with evolutionary approaches are analytical methods such as [88] or [89], which combine probabilistic search with local, gradient-based optimization for increased efficiency. Particularly popular in this space are scalarization approaches (like [88]), which perform continuous optimizations over sampled weight combinations.

2.4.2 Multi-Objective Control Policy Optimization

Most of the previous multi-objective policy optimization work can be classified into three categories. Single-policy methods convert the multi-objective problem into a single-objective problem [90, 91] using a scalarization function. The main drawback of these methods is that the preference weights must be set in advance. Multi-policy methods compute a set of policies to approximate the real Pareto-optimal set [92, 93, 94]. The main bottleneck of these methods is the high computational requirement. This prevents these methods from finding dense Pareto solutions for complex control problems. Our work falls into this category but resolves this limitation by dynamically allocating computing resource by a prediction-guided selection optimization. Meta policy methods and single universal policy methods either compute a meta policy and adapt it to different preferences, or directly generate output control conditioned on input preference weights [95, 96, 97, 98]. Those methods share the same shortcomings as assuming the optimal control strategies for different objectives trade offs are similar to each others and thus can be represented by a single neural network. Finally, most methods in this class still only work for problems with discrete action space and simple mechanisms (*e.g.*, deep sea treasure environment).

While multi-objective optimization techniques have been explored to robotic control problems, it is still under-explored for co-design problems yet.

Chapter 3

Robot Shape Representation

To achieve the goal of computationally constructing the shape and control of robotic systems, the first significant challenge is how to computationally represent the robot shape designs. A complete description of a robot shape should contain both discrete shape parameters and continuous shape parameters. The discrete parameters may define the topological structure of the robot (*e.g.*, number of body links, how links connect together), and the types of the joints connecting body links. The modifications on the discrete design parameters typically result in changes in the degrees of freedom of the system, and the kinematic structure of the robot. On the other hand, the continuous shape parameters usually contain the morphology dimensions of each robot link (*i.e.*, size), the placements of the connection joints, stiffness and damping coefficients of the joints, and the material parameters of the robot materials. While the changes of the continuous parameters usually does not affect the structure of the robot, they are deeply related to the dynamics response behavior of the robots.

Since the hybrid nature of the robot shape parameters introduces an enormous design search space, we need to find a proper way to represent the robot shape so as to effectively structure this hybrid and vast robot shape space. An effective robot shape representation has to describe the discrete part and the continuous part of the robot shape concisely and comprehensively, as well as be amenable to optimization. In this chapter, we introduce our solution to represent both the discrete part and

the continuous part of the robot shape, which covers a incredibly wide variety of complex robot shapes while guaranteeing the represented designs to be always valid for real manufacturing. We take the shape representation of the *manipulator hand* design problem as a case study to illustrate our representation methods. We will show in later chapters that our robot shape representations allow us to devise robot shape optimization algorithms to search for the optimal robotic cyberphysical system effectively and efficiently.

3.1 Graph Grammar Representation for Discrete Robot Shape Topology

In this section, we present our graph-based shape representation for manipulator's topological structures and the corresponding graph grammar-based design space representation for generating inherently fabricable manipulator hand designs.

Specifically, we represent each manipulator assembly design in the form of graph $G = (V, E)$. Each node $v \in V$ of the graph is corresponding to a physically realizable component module as shown in Figure 3-1, which can be either a finger segment (*e.g.*, fingertip, finger phalanx) or a joint. Each graph edge $e \in E$ is corresponding to a physical link between those components and encodes the connection-related variables (*e.g.*, relative rotation, translation, etc.). An example graph representation of a manipulator design is shown in Figure 3-2. This choice of graph representation guarantees that each assembly has a unique graph, and each graph corresponds to a unique assembly.

In order to generate diverse manipulator structure designs for computational construction purpose, we need an effective representation for the manipulator design space. The graph representation of manipulator topology help reduce the task of generating manipulator designs to generating design graphs. A desired design space representation for computational robot design need to satisfy two challenging requirements: first, it should be *expressive* to produce various manipulator designs not only

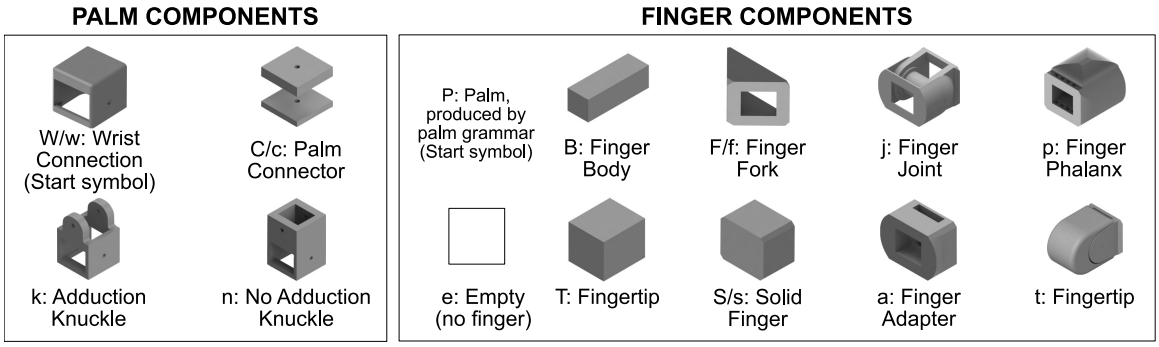


Figure 3-1: **3D models of the grammar's components with associated symbols.** Capital letters indicate that the component is a non-terminal symbol, while lowercase letters indicate a terminal symbol.

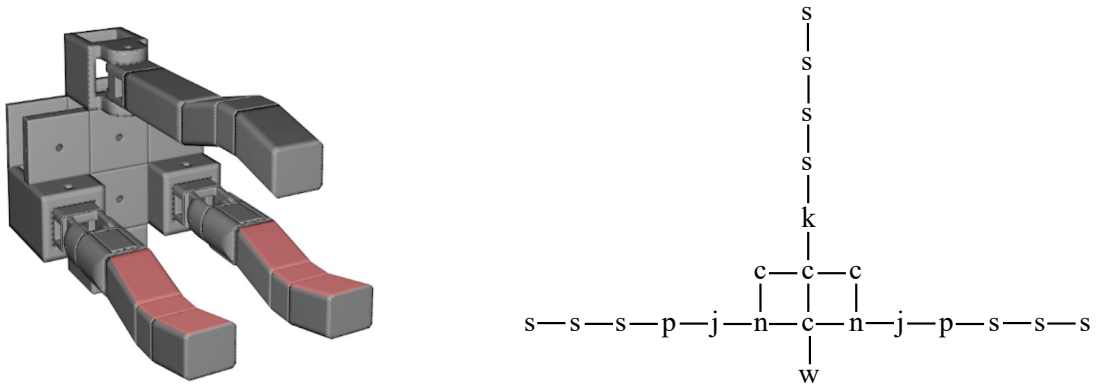


Figure 3-2: **A manipulator structure expressed by our graph representation and its corresponding graph representation.**

including the traditional manipulator designs but also including tons of novel manipulator designs which were not covered by human experts yet; second, it should be *valid*, which means that the generated design graphs need to correspond to physically fabricable manipulator designs.

We use a context-sensitive graph grammar for the manipulator design space representation to achieve the desired requirements. The grammar consists of a set of rules to combine input components in Fig. 3-1 to create sub-assemblies and from the sub-assemblies create a graph of a manipulator. More specifically, the grammar consists of

1) *Terminal symbols* (noted as lowercase letters). These represent the nodes and edges of a graph.

2) *Non-terminal symbols* (noted as uppercase letters). These represent sub-assemblies or sections of a graph.

3) *A start symbol.* A non-terminal symbol that initializes the design.

4) *Expansion rules.* These convert non-terminal symbols into other non-terminal and terminal symbols. They allow the creation of many different graphs based the order and selection of rules applied.

Fig. 3-1 shows the terminal and non-terminal components (with their associated letter symbols) used to create the manipulators in this paper. It should be noted that the "library" of input components shown here consists only of the components required to make the manipulators in the experiments and that the "library" can be augmented as desired.

More specifically, our manipulator grammar consists of two sub-grammars, a *palm grammar* and a *finger grammar* with rules defined in Fig. 3-3. The *palm grammar* is used to generate palms of different sizes, shapes, and numbers of finger slots. Once the palm grammar produces a palm, the grammar proceeds with the *finger grammar* to grow fingers from the palm.

To allow the *palm grammar* to create various shapes of palms, we built modularized palm components (shown in Fig. 3-1 left). These components can be connected in a grid-like, planar manner via the palm grammar rules to create a planar palm of almost any shape (see Fig. 3-4). It should be noted that each palm grammar rule has three variants by rotating the rule by $90k(k \in 1, 2, 3)$ degrees. For example, a knuckle node (k) can be connected to either left, right, top and bottom to a connector node (C). Once the palm has been built, it is transformed into a start symbol for the finger grammar to attach fingers to the palm if desired.

Similar to the structure of human fingers, the *finger grammar* combines the finger components in Fig. 3-1 in a linear fashion: components can be added distally to the fingertips to "grow" them in length until the finger is terminated with a fingertip.

Fig. 3-5 shows some of the diverse manipulator configurations that can be generated from our grammar. The number of different manipulator configurations increases

PALM GRAMMAR (R_p)			
1. W □ → w C	3. W □ → w k	5. C □ → C C	7. C → c
2. W □ → w n	4. C □ → C k	6. C □ → C n	
FINGER GRAMMAR (R_f)			
1. $P(k^+) \rightarrow P(k-1)$ e	5. B → S	9. B → F(2)	13. T → a S
2. P(k) → p	6. B → t	10. $F(k^+) \rightarrow F(k-1)$ B	14. S → s S
3. $P(k^+) \rightarrow P(k-1)$ B	7. B → j p B	11. F(k) → f	15. S → s
4. B → j F(2)	8. B → j T	12. T → a t	

Figure 3-3: **Grammar expansion rules for constructing fingers and palms.** The palm grammar is defined on a grid layout and the finger grammar is a parametric grammar where the palm node "P" and fork node "F" contain an integer parameter k to denote the number of rule expansions can be made on the node. k^+ means that rule can be applied only when k is positive.

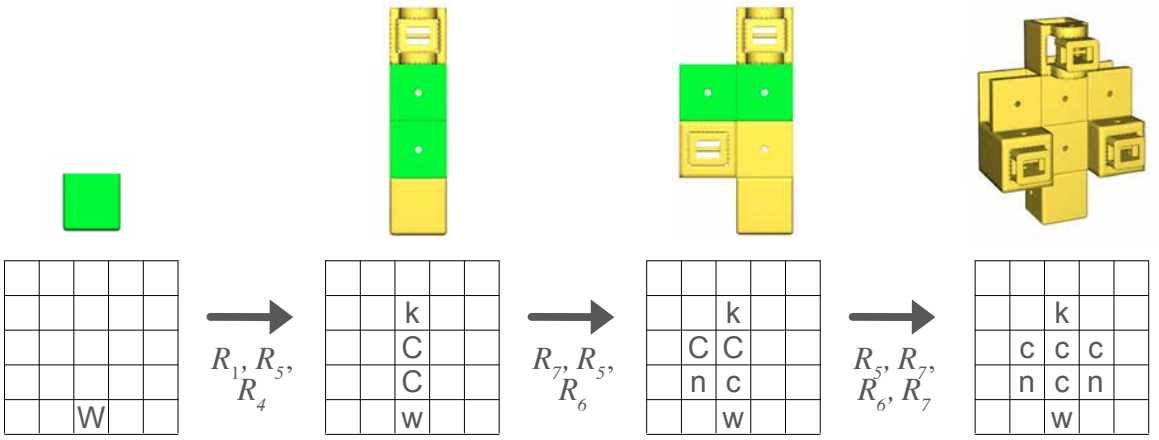


Figure 3-4: **An example of palm grammar rule application.** The palm grammar rules are applied to grow the start symbol (W), add connector components (C), and attach knuckles (k and n) to create the grid-based water bottle palm. The green components are non-terminal, and the yellow components are terminal.

exponentially as we increase the number of applied rules in our grammar. As seen in the figure, with only a small number of input components (thirteen in our work), our grammar is able to produce a large variety of manipulator configurations in different shapes of palm and fingers. On the order of 10^8 unique fingers can be generated from fifteen finger expansion rules and six terminal finger components, assuming the

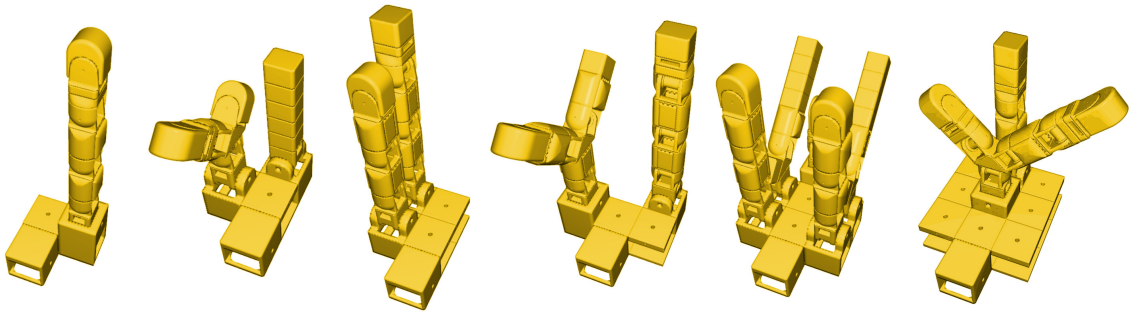


Figure 3-5: **A sample of the diversity of manipulator designs that the grammar rules and components can produce.** The manipulators are shown before deformation.

fingers are arbitrarily limited to at most three finger segments in length for calculation purposes. Restricting the palm to a three by three grid (again for calculation purposes) which may have at most six fingers attached, at least 10^{49} unique hands exist within our constraints. Such a broad design space can be drastically increased with additional grammar pieces.

Although we demonstrate the graph grammar representation approach on the manipulator hand design problem, it is actually a general approach for other types of robots. We will show another example later in Chapter 5 where we use the graph grammar representation for the terrestrial robot topology space to facilitate the optimal design search.

3.2 Deformation-Based Representation for Continuous Robot Morphology

3.2.1 Motivation

The graph grammar representation allows us to represent tons of different robot topology structures with different combinations of the components. However, the continuous morphology parameters cannot be easily supported by the graph grammar representation due to the discrete nature of the graph grammar approach. Indeed, defining a representation of the continuous robot morphology design that is amenable

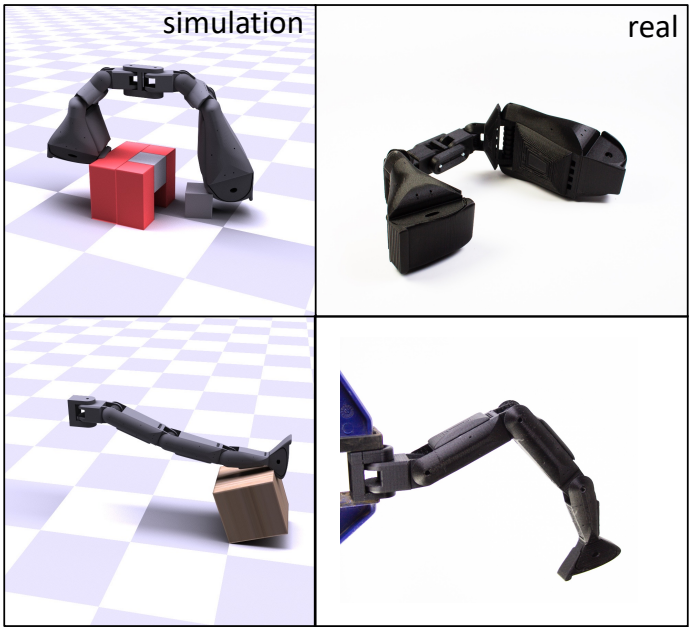


Figure 3-6: **Manipulator morphology designs generated by our cage-based representation.** *Left*: a two-finger gripper and a single-finger gripper with complex geometry shapes in simulation; *Right:* pictures of 3D-printed manipulators. Our method outputs designs that are easy to print and assemble.

to optimization is an even more significant challenge than the discrete robot topology. A good continuous morphology representation should: (a) result in designs that can be manufactured; (b) enable design of articulated robots with complex geometric shapes; and (c) allow the use of powerful optimization methods such as gradient descent.

Several recent works have studied the design representation problem [6, 9, 20, 21, 24]. One strategy is to represent the design as a graph, where each edge denotes connection between two components of the robot [6, 9]. However, just defining the topology is insufficient: it is also necessary to parameterize the shape of each component. While there are many ways to represent the shape of a single object, the shape representation of an articulated system is a challenge. It is because the shape representation must span a rich space of geometric designs while simultaneously satisfying connectivity constraints between components and the manufacturing limitations. A popular strategy is to model each component by a simple primitive shape (*i.e.*, cylinder, cuboids, etc.) [9, 13, 17, 20, 21]. However, while easy to opti-

mize, these primitives are often an over-simplification of the desired shape and are insufficient to model complex gripper designs as shown in Figure 3-6.

An alternative to basic shape primitives is the CAD parameterization. However, this approach has several downsides: generating models from CAD parameterization is slow [22], and updates in CAD model parameters often results in failures such as the model being disconnected or even failure in generating the model. Other options for rich shape representations are voxel grid, point cloud, signed distance functions [24], etc. Unfortunately, these are not suitable for articulated robot design because it is hard to impose connectivity constraints between individual components. A drastic alternative to overcoming the connectivity problem without compromising rich shape parameterization is to directly optimize the shape of the entire robot instead of individual components. But this does not solve the problem either because now identifying individual joints and components, which is necessary for simulation and manufacturing, becomes a problem. Another consideration worth discussing is that for learning to control one must first import the shape into the simulator. While CAD designs can be easily imported they do not provide analytical gradients. Other representations discussed above must be first converted to a mesh, a step that is non-differentiable. Without analytical gradients one must rely on data inefficient gradient free methods for solving the shape optimization problem.

Motivated by the challenges of the problem and the limitations of the existing approaches, we study the effective parameterization for the continuous shape morphology assuming a fixed robot topology. We propose a general morphology representation for articulated robot designs based on cage-based deformation (CBD) models. Cage-based deformations [99, 100] have been widely used in computer graphics to deform a mesh through a few number of cage "handles" (*i.e.*, cage vertices) in real-time while preserving local geometric features. Instead of specifying a large number of optimization parameters for modelling complex shapes, CBD maintains an *expressive* shape design space with only a few parameters. Furthermore, as we will describe later the cage representation allows imposition of a rich set of manufacturing and connectivity constraints. Most importantly, CBD is not tied to a specific shape

Table 3.1: **Summary of different Morphology Parameterization Methods**

Parameterization Method	Primitive Shapes [12, 13, 17, 18, 77]	CAD Parameterization [22, 23]	TSDF [24]	Mesh-based	Deformation-based (ours)
Mesh Inference	Fast	Slow	Slow	Fast	Fast
Complex Shape	No	Support	Support	Support	Support
Differentiability	Support	No	No	Support	Support
Dimension	Low	Controllable	High	High	Controllable
Feature-preserving	No	Yes	No	No	Yes
Articulated Design	Support	No	No	No	Support

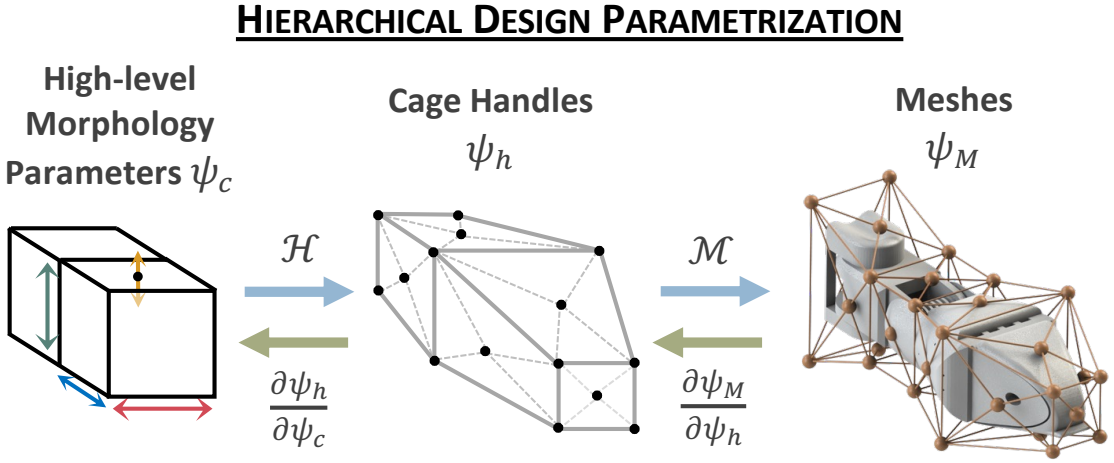


Figure 3-7: **Hierarchical design parameterization for articulated robot morphology designs.** Blue arrows labeled as $\mathcal{H}, \mathcal{M}$. The corresponding green arrows are the derivatives.

representation and can be easily used with different representations such as meshes, point clouds etc. It is computationally inexpensive for inference, flexible (*i.e.*, user can easily control the degrees of freedom describing the shape by changing the number of cage handles), and differentiable. We summarize the comparison among different morphology parameterization methods in Table 3.1.

We test our framework on multiple manipulation problems, some of which are shown in Figure 3-6. The experiments show that our deformation-based parameterization provides us an expressive design space and the expressed designs can be easily manufactured (Figure 3-6 right).

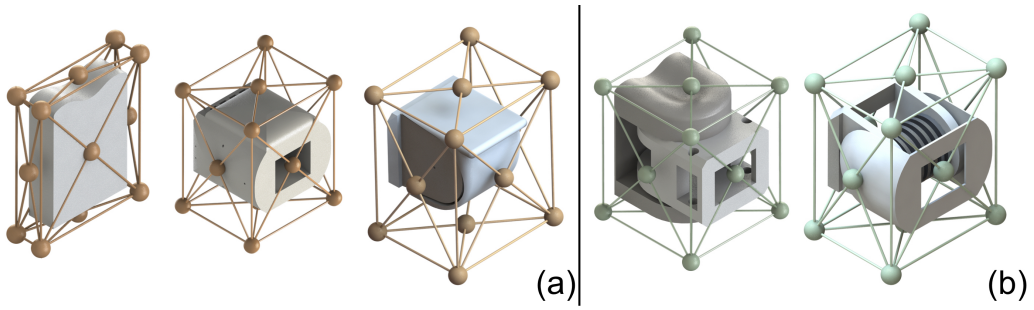


Figure 3-8: **Our component database for our manipulators.** From left to right: finger base, phalanx segment, finger tip, knuckle, and joint. Each component comes with its own deformation cage. (a) The components in the yellow cages can be deformed arbitrarily with the cage, whereas (b) the components in the green cages can only be expanded along the axis of rotation.

3.2.2 Hierarchical Morphology Parameterization

We now describe our novel deformation-based design space for articulated robot morphology. A key insight of our morphology design space is the use of cage-based deformation, which allows us to morph the underlying mesh using a small number of cage "handles". More specifically, as shown in Figure 3-7, we use a two-level hierarchy to parameterize the shape. The high-level morphology parameters (shown in red, green, blue, and yellow arrows) controls the positions of the cage vertices (handles), and the cage handles in turn deform the underlying mesh.

Morphology optimization relies on an effective morphology design space, which further depends on an effective morphology shape parameterization. In this section, we describe our approach to leverage the cage-based deformation as the morphology parameterization for articulated robots with *complex* component shapes. As shown in Figure 3-7, we use a two-level hierarchy to parameterize the shape of the robot: the cage-based deformation $\mathcal{M}$ and high-level morphology parameterization $\mathcal{H}$.

Cage-based Deformation ($\mathcal{M}$) Cage-based deformation (CBD) is a classic geometry processing technique in computer graphics used to deform a high-resolution mesh in a real-time and feature-preserving manner. With a coarse, closed cage, CBD controls the enclosed space's deformation by moving the cage vertices, or *cage handles* around. Let $\mathcal{C}$ denote the cage and H denote the cage vertices (*i.e.*, handles) of $\mathcal{C}$

with $\tilde{\psi}_h$ being the positions of the handles in H in the rest configuration, and let $\mathcal{S}$ be the space enclosed by cage $\mathcal{C}$. For any arbitrary point $\tilde{s} \in \mathcal{S}$, CBD computes a normalized barycentric coordinate $\boldsymbol{w} \in \mathbb{R}^{|H|}$ for the point, called *deformation weights*, which satisfies:

$$\tilde{s} = \sum_{j}^{|H|} \boldsymbol{w}_j \tilde{\psi}_h(j) \quad \text{and} \quad \sum_{j}^{|H|} \boldsymbol{w}_j = 1. \tag{3.1}$$

These deformation weights then define a linear function to transfer the translation of handle vertices to the movement of the associated point at run time through:

$$s = \sum_{j}^{|H|} \boldsymbol{w}_j \psi_h(j), \tag{3.2}$$

where ψ_h is the new positions of the cage handles, and s is the new position of $\tilde{s}$ under deformation. The deformation weights are precomputed for each particular point in the space $\mathcal{S}$ and kept constant at run time. CBD methods preserve various features of the underlying, high-resolution mesh after deformation by carefully designing the weight construction algorithms. Among various CBD methods, we choose the mean value coordinates method [100] for its simplicity, stability, and capability to be extended to deform articulated structures.

Inspired by its power, we apply cage-based deformation to parameterize the shape of each robot component by the positions of a set of cage handles around the shape mesh. In practice, the cage handles of each component can be defined by the users based on their demands. For our purpose in this work, we construct a component database for manipulator construction as shown in Figure 3-8. Each component is represented by a mesh M^i and is associated with a predefined cage $\mathcal{C}^i$ around it. We then use the mean value coordinates method to precompute the deformation weight matrix D^i for each component mesh M^i, and reuse those weights afterwards. Let V^i be the set of vertices of mesh M^i and H^i be the set of handles in cage $\mathcal{C}^i$. Then D^i is a $|V^i|$-by-$|H^i|$ matrix storing the deformation weight for each vertex on the mesh with respect to each handle on the cage. The deformation weights precomputation by mean value coordinates method is cheap, with a computational cost $\mathcal{O}(|V^i| \cdot |H^i|)$. Such

precomputation and the run-time linear combination enable a fast mesh inference given the new cage handle positions. Moreover, by controlling the high-resolution mesh's morphology through a coarse cage, we effectively reduce the morphology space into a relatively low-dimensional space for natural deformed shapes. This provides us a constrained yet expressive morphology design space. Furthermore, the dimension of the deformation is still fully controllable. By adding more handle points, one can deform the underlining mesh with more degrees of freedom, which makes it possible to find a good trade-off between low-dimensional morphology parameter space and fine-grained morphology deformation for different applications. Most importantly, this linear combination gives us a fully differentiable function $\mathcal{M}$ mapping from handle positions ψ_H to the positions of mesh vertices ψ_M.

High-level Morphology Parameterization ($\mathcal{H}$) The CBD method described above works for any *single* mesh with an arbitrary shape. However, parameterizing an articulated robot poses extra challenges, since independently manipulating the cage handles arbitrarily for different robot components will easily lead to a design that is not connected and not manufacturable, as shown in Figure 3-9(c). In order to handle proper articulation, we need to take two extra constraints into consideration: the *fabrication constraint* and the *connectivity constraint*.

The *fabrication constraint* requires us to have different deformation constraints for different components based on their manufacturing methods. For instance, the finger body part can be manufactured using a 3D printer, so it can undergo free-form deformation. In contrast, the joint component is usually composed of some commercially-sourced products (*e.g.*, screws, spring pins, etc.), which come in pre-defined, standard sizes. Thus, components such as joints can only be expanded in directions that maintain the geometric integrity of the model where it interfaces with these standard-sized parts. We achieve this through further parameterizing the cage with a few extra parameters to control the allowed deformation for each component.

Connectivity constraint is a more challenging problem for articulated designs, requiring components to remain connected after deforming each underlying mesh. It is

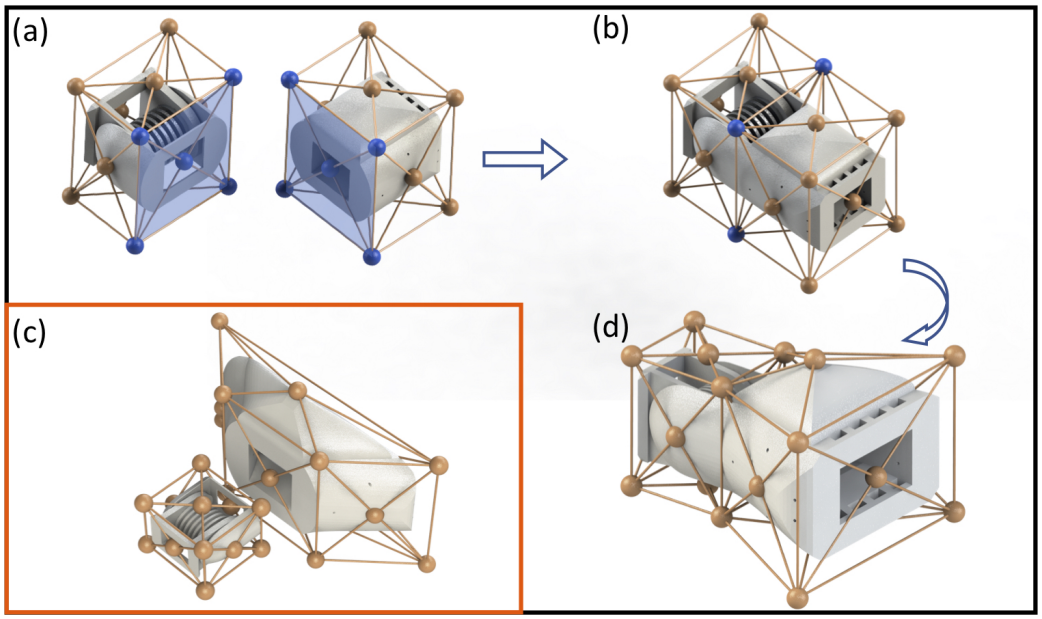


Figure 3-9: **Cage-based deformation for articulated robot morphology parameterization.** A *joint* component and a *body phalanx segment* component are shown in the figures. We parameterize the articulated components into lower-dimensional parameters ψ_c by posing different deformation constraints on each component and merging their handle points on the connection surface (highlighted in blue in (a) and (b)). We can then freely explore the ψ_c space to change the underlining articulated robot shape (d). The two components come apart from each other and become to be not manufacturable if they are deformed individually and arbitrarily by their associated cages (c).

non-trivial for CAD-based and mesh-based parameterizations to satisfy this constraint since they are unable to track the connectedness of the interface surfaces between components while varying the parameters. We instead leverage a special property of mean value coordinates to resolve the connectivity issue. Let $\langle H_1, H_2, H_3 \rangle$ be three handles of a triangle on the cage mesh $\mathcal{C}$, and s be a point in the enclosed space. With mean value coordinates, if s and $\langle H_1, H_2, H_3 \rangle$ are coplanar, and s is inside the triangle $\langle H_1, H_2, H_3 \rangle$, then the deformation weights for s have the following property:

$$\boldsymbol{\omega}_i^s = \begin{cases} 0, & \text{if } i \notin \{H_1, H_2, H_3\} \\ > 0, & \text{if } i \in \{H_1, H_2, H_3\}. \end{cases} \tag{3.3}$$

In other words, s is fully controlled by the triangle that contains it.

With this property in mind, we construct the cage for each component so that, for each component's connection surface (*i.e.*, interface to a neighboring component), we

always construct a cage that is coplanar with and fully contains the connection surface as shown by the blue faces in Figure 3-9(a). We call the handles on the connection surface plane *connection handles*. For two components that can be potentially connected, their connection handles are constructed to be identical, which ensures that the connection surfaces on two components share the same connection handle layout. While connecting two components, we merge their overlapped connection handles (Figure 3-9(b)). Such merge operation is equivalent to adding a constraint on the handles on the connection surface so that their connection handles always have the same motion. Due to the property of mean value coordinates in Eq. 3.3, this placement of connection handles ensures that the connectivity constraint is automatically satisfied.

By considering both the *fabrication constraint* and the *connectivity constraint*, we parameterize the whole cage via a small number of high-level morphology cage parameters ψ_c. This extra layer of parameters implicitly imposes constraints on all cage handle positions to move around in a unified fashion. Specifically, ψ_c consists of: (a) the scale information of the cages (*e.g.*, length of each cage, and width/height of each connection cage, shown as red, blue, and green arrows in Figure 3-7); and (b) any other auxiliary cage points (*e.g.*, yellow vertical arrow in Figure 3-7). The scale parameters are component-dependent and are based on their fabrication constraint. For example, the joint cage can only be scaled along the joint axis direction. Through such cage parameterization, we can map the high-level cage parameters ψ_c to the cage handle positions ψ_h by $\psi_h = \mathcal{H}(\psi_h)$, and effectively construct a high-level morphology design space with only bound constraints. Note that this high-level cage parameterization is not a fixed choice; the parameterization can be modified based on the application and user's demand.

3.2.3 Results

To demonstrate the quality of the morphology design space represented by our deformation-based parameterization, we randomly sample the high-level morphology parameters

ψ_c for two manipulator topologies: *single-finger* and *two-finger gripper* configurations (Figure 3-10). Our deformation-based parameterization method has a compact, yet expressive and rich design space. As shown in Figure 3-10, using only 9 and 17 design parameters respectively for the two configurations, our method is able to generate various natural and fabricable morphology designs of articulated robots while maintaining the connectivity of the articulated manipulators. Furthermore, the represented manipulator designs can be easily manufactured by 3d printing the parts and assembling the parts together as shown in Figure 3-6 (Right).

Figure 3-10: **Morphology design space**. The initial morphology of the single finger and the two-finger gripper designs are shown on the left. We randomly sample different parameters for each configuration and show the deformed morphology on the right.

3.3 Hybrid Shape Representation for Robot Designs

By combining the graph grammar representation for robot shape topology and the deformation based representation for continuous robot morphology shape, we obtained a complete two-stage hybrid representation for robot designs shown in Figure 3-11.

Take the manipulator as an example again. In the first stage, the graph grammar is applied to define the topology structure of the manipulator. Once the manipulator topology is determined, the representation process enters the second stage where the cage-based deformation is utilized to define the continuous morphology shape of each manipulator component.

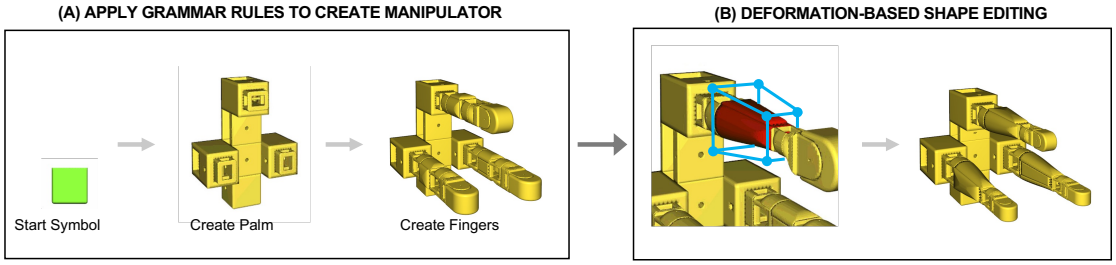


Figure 3-11: **Proposed hybrid representation for manipulator shape designs:** A manipulator topology structure (discrete part) is formed from grammar representation (A), and then the cage-based representation parameterize the continuous geometry shape of each manipulator component (B).

Technically, a deformation cage is predefined for each primitive manipulator component in our database shown in Figure 3-1 and Figure 3-8. To meet the *connectivity constraint* described in Section 3.2.2 and Figure 3-9, the cages are constructed in the way that any pair of connectable components share an identical cage face at the connecting interface plane of the components. To make the hybrid representation computationally seamless, during the grammar rule application, the geometry meshes of the manipulator components, as well as their associated *deformation cages*, are automatically connected. The output articulated geometry meshes and the articulated cages of the robot are served as input to the second stage for further continuous shape changing.

To better illustrate our hybrid shape representation, we developed a design interface (as shown in Fig. 3-11) for our manipulator designs, where the design interface enables the users to explore the enormous manipulator designs spanned by our hybrid shape representation. The constructed designs from our interface can be easily 3d printed and assembled. To further increase the capability of the manipulators, we equipped the manipulators with knitted tactile sensors [101]. The manufactured manipulators can then be controlled to successfully complete various complex manipulation tasks such as picking up an egg, screwing a wing nut, pouring water, and cutting paper with scissors as shown in Figure 3-12.

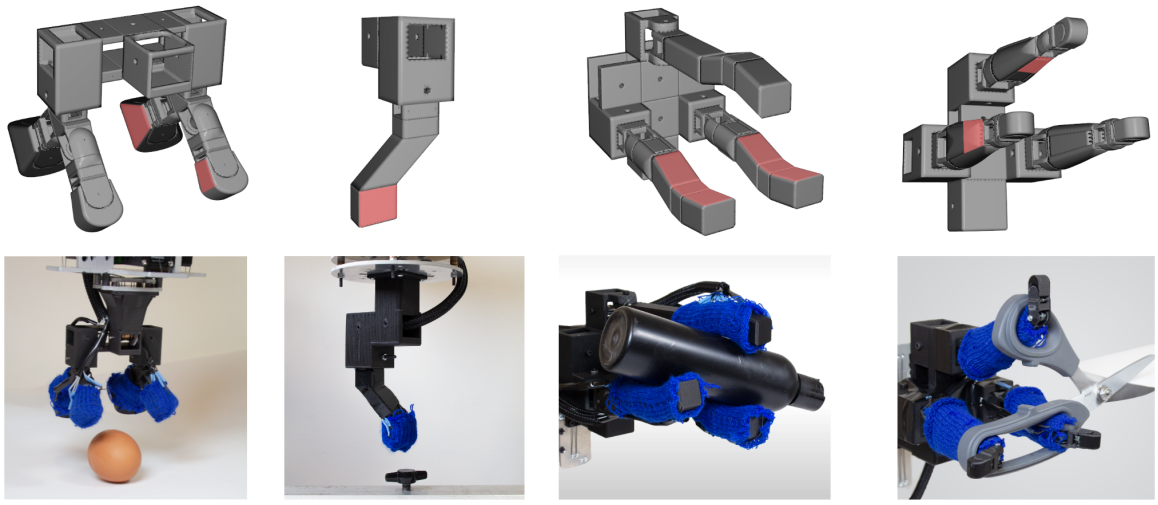


Figure 3-12: **Manipulator designs (combined with tactile sensors) constructed from our developed design interface.** *First row:* The digital designs output from our software with red surfaces representing where to put tactile sensor one. *Second row:* The corresponding 3d printed manipulator designs. When equipped with knitted tactile sensors, they are able to complete various complex manipulation tasks.

Chapter 4

Computational Robot Control Design via Differentiable Physics

Being the brain of a robot, the robotic control strategy plays a crucial role in its task performance. Given a task specification and a predefined robot hardware design, control optimization aims to find the best control strategy for this robot to achieve the optimal performance on this task. In the framework of computational robot co-design, the control strategy serves as an evaluation of a robot shape so as to provide feedback to the robot shape optimization. Therefore, finding the optimal control strategy for a given robot shape is a prerequisite for an unbiased robot shape evaluation, thus being the cornerstone for computational robot design.

Traditionally, researchers compute an open-loop state-action trajectory via numerical trajectory optimization methods [49] and then apply classic feedback control algorithms (*e.g.*, PID, LQR) to control the robot to follow the optimized open-loop trajectory. However, the design of the classical control architecture typically relies on a deep understanding of the task-specific control strategy structure. More recently, neural network policy has become an increasingly popular alternative for controller representation due to the generalizability of the neural network architecture on the complex control problems. Among various policy learning techniques, reinforcement learning (RL) has been a particularly successful tool to learn policies for systems

ranging from robots (*e.g.,* Cheetah, Shadow Hand) [50, 54] to complex animation characters (*e.g.,* muscle-actuated humanoids) [1] using only high-level reward definitions. Despite this success, RL requires large amounts of training data to approximate the policy gradient, making learning expensive and time-consuming, especially for high-dimensional problems (Figure 4-1). The recent development of differentiable simulators opens up new possibilities for accelerating the learning and optimization of control policies. A differentiable simulator may provide accurate first-order gradients of the task performance reward with respect to the control inputs. In this chapter, we argue that such additional gradient information offered by the differentiable simulators allows the use of efficient gradient-based methods to optimize policies.

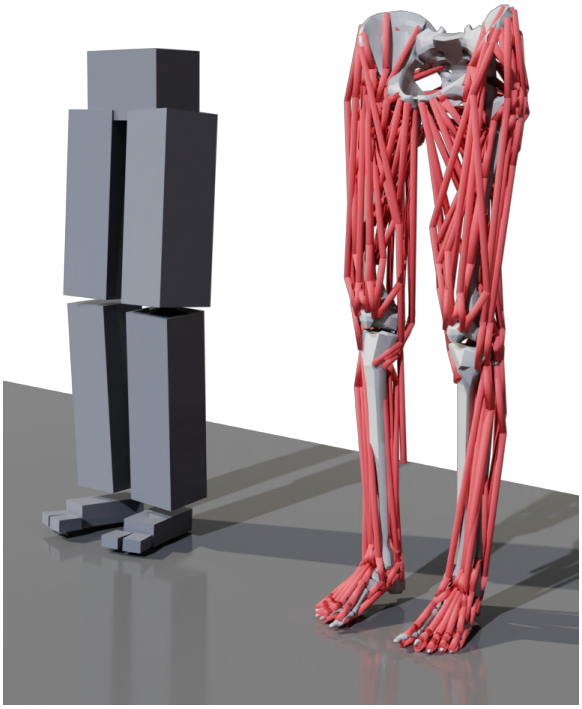


Figure 4-1: **High-dimensional muscle-actuated humanoid control problem.** Reinforcement learning approaches struggle on high-dimensional control problems due to the high demand of stochastic sampled trajectories to estimate the policy gradients.

As evidences, we first introduce a novel differentiable tactile simulator that is able to reliably and efficiently simulate the tactile feedback for articulated rigid robots and provide the analytical first-order gradients of the simulation with respect to the policy parameters. We demonstrate that for basic tactile-based manipulation

tasks, by using straightforward backpropagation through time technique (BPTT) to compute policy gradients via the differentiable simulation, we are able to improve the policy learning efficiency and obtain better-performing policy by large margin over the state-of-the-art reinforcement learning approaches. While BPTT works in basic cases, the numerical gradient based optimization typically poses problems (*e.g.*, prone to local minima) in more complicated task scenarios. In the second part of this chapter, we present our novel policy learning algorithm SHAC (Short-Horizon Actor-Critic) to significantly improve the policy learning efficiency on complex and high-dimensional control tasks by combining differentiable simulation and reinforcement learning.

4.1 Differentiable Articulated Rigid Body Simulation with Tactile Feedback

In this section, we argue that differentiable physics simulator can help us optimize the robotic control policy. Simply applying a gradient descent optimizer (*e.g.*, Adam) with naive backpropagation through time gradient computation can significantly improve the efficiency of the policy training and the performance of the trained policy. To demonstrate that, we present a novel differentiable tactile simulator for articulated rigid robots that can efficiently and reliably simulate both normal and shear tactile force fields covering the entire contact surface. We develop a fast penalty-based tactile model which can run at 1000 frames/s on a single core of Intel i7-9700 CPU. Our tactile model can reasonably approximate the soft contact nature of soft tactile sensor material such as the elastomer used in GelSlim [102], generate dense tactile force fields (*e.g.*, the dense marker array on GelSlim), and is compatible with arbitrary tactile sensor spatial layout (*i.e.*, flat plane, hemisphere, etc.). Most importantly, our compact tactile formulation is differentiable, which allows the simulator to provide fast analytical gradients for the entire dynamics chain.

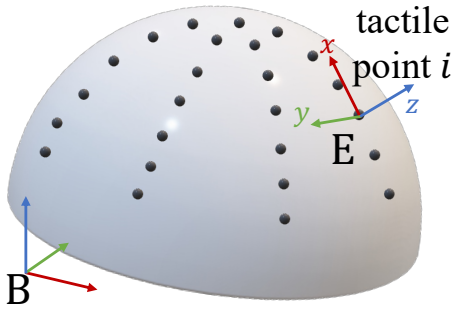


Figure 4-2: **Tactile Sensor Representation.**

4.1.1 Tactile Sensor Representation

Each tactile point i on a sensor pad is represented by a tuple $\langle \mathbf{B}_i, \mathbf{E}_i, \xi_i \rangle$ as shown in Fig. 4-2. $\mathbf{B}_i$ is the rigid body the tactile point is attached to, and $\mathbf{E}_i \in \mathrm{SE}(3)$ is the position/orientation of the point in the local coordinate frame of the body, with the $\boldsymbol{x}_i$ and $\boldsymbol{y}_i$ axes in the shear-direction plane and the $\boldsymbol{z}_i$ axis along the normal tactile direction. (These axes are the same for all points for a *planar* sensor pad.) Finally, ξ_i are the simulation parameters of the penalty-based tactile model, which will be introduced later in §4.1.2. Our representation for tactile points is flexible, allowing us to specify any number of points in arbitrary geometry layouts on a robot, and each tactile sensor can have its individual configuration parameters.

4.1.2 Penalty-based Frictional Contact and Tactile Model

Analytic articulated dynamics simulation can be non-smooth and even discontinuous when contact and joint limits are introduced, and special care must be taken to ensure smooth dynamics. To model contact, we extend the differentiable penalty-based contact model of Geilinger et al. [35], adapted to our reduced coordinate approach. We make two critical changes to make their penalty-based approach work in our manipulation settings.

First, we add support for frictional contact between *two dynamic* bodies, rather than between a single dynamic body and a stationary surface (*e.g.*, floor and walls). To solve this, we use a signed distance field augmented with derivative information.

This distance field is attached to the contact body, which we assume is rigid. For collision detection, we query the signed distance function attached to the collision body, which gives us the following quantities:

d, $\dot{d}$	penetration distance and speed
$\mathbf{n}$	contact normal
$\mathbf{v}_t$	tangential velocity

as well as the *derivatives* of these quantities with respect to the state of the robot $\mathbf{q}$.

Second, we modify the contact damping force so that it is continuous. The original formulation by Geilinger et al. [35] has a discontinuity, which we found can cause convergence problems when there are collisions between two moving dynamic objects. In their formulation, the contact damping force is proportional to the penetration speed, $\dot{d}\mathbf{n}$, which means that there will be a sudden change in the magnitude of this damping force at $d = 0$, since $\dot{d}$ is not necessarily zero when d is zero. To fix this, we instead make the damping force be proportional to both the penetration distance and speed, $d\dot{d}\mathbf{n}$, which ensures that the force remains continuous at the moment of collision. If needed, we can also use a sigmoid function around d to make the damping force match the original formulation when d becomes large.

Mathematically, our penalty-based frictional contact model is as follows:

$$\boldsymbol{f}_n = (-k_n + k_d\dot{d})\min(d, 0)\mathbf{n} \tag{4.1a}$$

$$\boldsymbol{f}_t = -\frac{\mathbf{v}_t}{\|\mathbf{v}_t\|}\min(k_t\|\mathbf{v}_t\|, \mu\|\boldsymbol{f}_c\|) \tag{4.1b}$$

where $\boldsymbol{f}_n, \boldsymbol{f}_t$ are the contact normal force and contact friction force respectively, and k_n, k_d, k_t, μ are contact stiffness, contact damping coefficient, friction stiffness and coefficient of friction respectively.

We use the same penalty-based model in Eq. 4.1a to characterize the tactile force on each tactile point. The coefficients k_n, k_d, k_t, μ are specified for each tactile point to reflect spatially varying tactile material property and they together form the simula-

tion parameters ξ of the tactile point: *i.e.*, for the i^{th} tactile point, $\xi_i = \{k_n^i, k_d^i, k_t^i, \mu^i\}$. After the frictional contact force is computed for each point as $\boldsymbol{f} = \boldsymbol{f}_n + \boldsymbol{f}_t$, we transform this force into the local coordinate frame of the tactile point to acquire the desired *shear* and *normal* tactile force magnitudes:

$$T_{sx} = \boldsymbol{f}^\top \boldsymbol{x}, \qquad T_{sy} = \boldsymbol{f}^\top \boldsymbol{y}, \qquad T_n = \boldsymbol{f}^\top \boldsymbol{z}, \tag{4.2}$$

where $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}$ are the axes of frame $\mathbf{E}$.

4.1.3 Forward Dynamics

Our simulator follows the reduced-coordinate rigid-body dynamics formulation of RedMax [37] so that the equations of motion are expressed compactly using a minimal set of degrees of freedom. The dynamics equations are implicitly integrated in time with the BDF1 scheme [103] with step site h step forward the simulation.

Mathematically, at each time step t, we take a state in reduced coordinate representation $(\boldsymbol{q}_{t-1}, \dot{\boldsymbol{q}}_{t-1})$ and get the state $(\boldsymbol{q}_t, \dot{\boldsymbol{q}}_t)$ through solving the following equation with Newton's Method:

$$\left.\begin{aligned} \boldsymbol{q}_t &= \boldsymbol{q}_{t-1} + h\dot{\boldsymbol{q}}_t \\ \dot{\boldsymbol{q}}_t &= \dot{\boldsymbol{q}}_{t-1} + h\ddot{\boldsymbol{q}}_t(\boldsymbol{q}_t, \dot{\boldsymbol{q}}_t, \boldsymbol{u_t}) \end{aligned}\right\} \Rightarrow \underbrace{\boldsymbol{q}_t - \boldsymbol{q}_{t-1} - h\dot{\boldsymbol{q}}_{t-1} - h^2\ddot{\boldsymbol{q}}_t(\boldsymbol{q}_t, \dot{\boldsymbol{q}}_t, \boldsymbol{u}_t)}_{g(\boldsymbol{q}_{t-1}, \dot{\boldsymbol{q}}_{t-1}, \boldsymbol{u}_t, \boldsymbol{q}_t)} = 0 \tag{4.3}$$

with

$$\ddot{\boldsymbol{q}}_t(\boldsymbol{q}_t, \dot{\boldsymbol{q}}_t, \boldsymbol{u_t}) = \mathsf{M}_r^{-1}(\boldsymbol{q}_t)\Big[\mathsf{f}_r(\boldsymbol{q}_t, \dot{\boldsymbol{q}}_t) + \mathsf{J}^\top(\boldsymbol{q}_t)\mathsf{f}_m(\boldsymbol{q}_t, \dot{\boldsymbol{q}}_t) + \mathsf{f}_{QVV}(\boldsymbol{q}_t, \dot{\boldsymbol{q}}_t) + \boldsymbol{u}_t\Big], \tag{4.4}$$

where $\boldsymbol{q}$ is the simulation state (*i.e.*, joint angles), $\boldsymbol{u}$ is the action (*e.g.*, joint torque), M_r is the generalized mass matrix in reduced coordinates, J is the jacobian, f_r is the generalized force vector generated by joint-space effect (*e.g.*, joint damping), f_m is the maximal wrench (*e.g.*, gravity, Coriolis forces, contact forces, external forces, etc.), f_{QVV} is the quadratic velocity vector. Whenever we need the velocity, we compute it

from the positions: $\dot{\boldsymbol{q}}_t = (\boldsymbol{q}_t - \boldsymbol{q}_{t-1})/h$. We will not go into details of Eq. **??** since they are standard and can be found in any simulation tutorials.

We also developed forward dynamics which is implicitly integrated in time with the BDF2 scheme, with SDIRK2 for the initial step, but we skip the derivation for it in this thesis.

We analytically derive all the derivatives required by these implicit time integration schemes, and we solve the resulting non-linear equations using Newton's Method with line search.

4.1.4 Backward Gradient Computation

Since we use an implicit time integration scheme for forward dynamics, the core step of gradients computation is to differentiate through the nonlinear equations of motion. We start by formulating a finite-horizon tactile-based policy optimization problem:

$$\underset{\boldsymbol{\theta}}{\text{minimize}} \;\; \mathcal{L} = \sum_{t=1}^{H} \mathcal{L}_t\big(\boldsymbol{u}_t, \boldsymbol{q}_t, \mathbf{v}_t(\boldsymbol{q}_t)\big) \tag{4.5a}$$

$$\text{s.t.} \;\; g(\boldsymbol{q}_{t-1}, \dot{\boldsymbol{q}}_{t-1}, \boldsymbol{u}_t, \boldsymbol{q}_t) = 0 \qquad \text{(Equations of Motion)} \tag{4.5b}$$

$$\boldsymbol{u}_t = \pi_\theta\big(\tilde{\boldsymbol{q}}_{t-1}, \tilde{\mathbf{v}}_{t-1}(\boldsymbol{q}_{t-1}), T_{t-1}(\boldsymbol{q}_{t-1}, \dot{\boldsymbol{q}}_{t-1})\big). \qquad \text{(Policy Execution)} \tag{4.5c}$$

Here, H is the task horizon, $\mathcal{L}_t$ is a step-wise task-dependent reward function, $\boldsymbol{u}$ is the action (*e.g.*, joint torque), $\boldsymbol{q}$ is the simulation state (*i.e.*, joint angles), and $\mathbf{v}$ is the derived auxiliary simulation variables (*e.g.*, fingertip positions) which themselves are a function of $\boldsymbol{q}$. Eq. 4.5b describes the nonlinear equations of motion (see Eq. 4.3 and Eq. 4.4). Eq. 4.5c represents the inference of the control policy π_θ to obtain the desired action given the partial observation of the simulation state $\tilde{\boldsymbol{q}}$, partial observation of the simulation computed variables $\tilde{\mathbf{v}}$, and the tactile force values T from Eq. 4.2.

We embed our simulator as a differentiable layer into the PyTorch computation graph and use reverse mode differentiation to backward differentiate through dy-

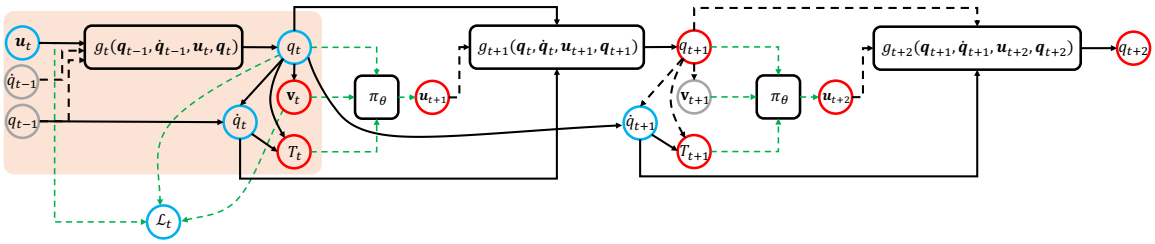


Figure 4-3: **Computation graph of the simulation with BDF1 time stepping around time step t.** We illustrate the computation graph for gradient derivations of $\partial\mathcal{L}/\partial\boldsymbol{q}_t$ and $\partial\mathcal{L}/\partial\boldsymbol{u}_t$. The boxes (*e.g.*, g, π_θ) represent functions, and the circles represent data/values. The grey circles are the data unrelated to the gradient derivation at step t. The red circles are the data $(\cdot)$ that we already have the gradient $\partial\mathcal{L}/\partial(\cdot)$ for when we arrive at step t during backward propagation. The blue circles are the data related to the gradients computation at step t. The green arrows are the data flows completed by PyTorch, and the black arrows are the data flows completed by our simulator. The dashed arrows are the data flows whose gradients computations are not handled by simulator or are not related to the derivation at the current step. The orange shaded part is our simulation layer of step t in the PyTorch computation graph.

namics time integration. To better illustrate the gradients derivation, we draw the computation graph in Fig. 4-3. The computation steps such as loss/reward computation and policy inference (*i.e.*, green arrows in the figure) are completed by PyTorch, and the dynamics-related computation (*i.e.*, black arrows) are processed by our simulator in C++. Each step of our simulation can be regarded as a function (shown in the orange shaded box in Fig. 4-3) in the computation graph:

$$(\boldsymbol{q}_t, \mathbf{v}_t, T_t) = \mathrm{Sim}(\boldsymbol{q}_{t-1}, \dot{\boldsymbol{q}}_{t-1}, \boldsymbol{u}_t). \tag{4.6}$$

We compute the gradients $d\mathcal{L}/d\theta = (d\mathcal{L}/d\boldsymbol{u}_t)(d\boldsymbol{u}_t/d\theta)$ for policy optimization. The first gradient, $d\mathcal{L}/d\boldsymbol{u}_t$, which includes the tactile derivatives, is derived analytically. The second gradient, $d\boldsymbol{u}_t/d\theta$, is computed by PyTorch's auto-differentiation.

Now we show how to compute $d\mathcal{L}/d\boldsymbol{u}_t$. We compute $d\mathcal{L}/d\boldsymbol{u}_t$ in reverse order, starting from the last time step. At the time step t, we assume that we have the following gradients computed: $\partial\mathcal{L}/\partial\mathbf{v}_{t,t+1,\ldots}$, $\partial\mathcal{L}/\partial T_{t,t+1,\ldots}$, $\partial\mathcal{L}/\partial\boldsymbol{q}_{t+1,t+2,\ldots}$ (red circles in Fig. 4-3). To complete the gradients backpropagation at step t, we need to compute

two derivatives $\partial\mathcal{L}/\partial\boldsymbol{q}_t$ and $d\mathcal{L}/d\boldsymbol{u}_t$:

$$\frac{\partial\mathcal{L}}{\partial\boldsymbol{q}_t} = \frac{\partial\mathcal{L}_t}{\partial\boldsymbol{q}_t} + \frac{\partial\mathcal{L}}{\partial\boldsymbol{q}_{t+1}}\left(\frac{\partial\boldsymbol{q}_{t+1}}{\partial\boldsymbol{q}_t} + \frac{\partial\boldsymbol{q}_{t+1}}{\partial\dot{\boldsymbol{q}}_t}\frac{\partial\dot{\boldsymbol{q}}_t}{\partial\boldsymbol{q}_t}\right) + \left(\frac{\partial L}{\partial T_{t+1}}\frac{\partial T_{t+1}}{\partial\dot{\boldsymbol{q}}_{t+1}} + \frac{\partial L}{\partial\boldsymbol{q}_{t+2}}\frac{\partial\boldsymbol{q}_{t+2}}{\partial\dot{\boldsymbol{q}}_{t+1}}\right)\frac{\partial\dot{\boldsymbol{q}}_{t+1}}{\partial\boldsymbol{q}_t}, \tag{4.7a}$$

$$\frac{d\mathcal{L}}{d\boldsymbol{u}_t} = \frac{\partial\mathcal{L}_t}{\partial\boldsymbol{u}_t} + \left(\frac{\partial\mathcal{L}}{\partial\boldsymbol{q}_t} + \frac{\partial\mathcal{L}}{\partial\mathbf{v}_t}\frac{\partial\mathbf{v}_t}{\partial\boldsymbol{q}_t} + \frac{\partial\mathcal{L}}{\partial T_t}\left(\frac{\partial T_t}{\partial\boldsymbol{q}_t} + \frac{\partial T_t}{\partial\dot{\boldsymbol{q}}_t}\frac{\partial\dot{\boldsymbol{q}}_t}{\partial\boldsymbol{q}_t}\right)\right)\frac{\partial\boldsymbol{q}_t}{\partial\boldsymbol{u}_t}. \tag{4.7b}$$

The derivatives $\partial\mathbf{v}_t/\partial\boldsymbol{q}_t$ can be computed from the functions $\mathbf{v}(\boldsymbol{q})$ easily. The derivatives $\partial T_t/\partial\boldsymbol{q}_t$, $\partial T_t/\partial\dot{\boldsymbol{q}}_t$ and $\partial T_{t+1}/\partial\dot{\boldsymbol{q}}_{t+1}$ can be derived from the tactile force formulation 4.2. The derivatives of $\partial\dot{\boldsymbol{q}}_t/\partial\boldsymbol{q}_t$, $\partial\dot{\boldsymbol{q}}_{t+1}/\partial\boldsymbol{q}_t$ and $\partial\dot{\boldsymbol{q}}_{t+2}/\partial\boldsymbol{q}_{t+1}$ can be computed from the BDF1 equations (Eq. 4.3).

$$\frac{\partial\dot{\boldsymbol{q}}_t}{\partial\boldsymbol{q}_t} = \frac{1}{h}\mathrm{I} \tag{4.8a}$$

$$\frac{\partial\dot{\boldsymbol{q}}_{t+1}}{\partial\boldsymbol{q}_t} = -\frac{1}{h}\mathrm{I}. \tag{4.8b}$$

To computing the remaining derivatives $\partial\boldsymbol{q}_{t+1}/\partial\boldsymbol{q}_t$, $\partial\boldsymbol{q}_{t+1}/\partial\dot{\boldsymbol{q}}_t$, $\partial\boldsymbol{q}_{t+2}/\partial\dot{\boldsymbol{q}}_{t+1}$, and $\partial\boldsymbol{q}_t/\partial\boldsymbol{u}_t$ we must differentiate through the implicit function $g(\boldsymbol{q}_{t-1}, \dot{\boldsymbol{q}}_{t-1}, \boldsymbol{u}_t, \boldsymbol{q}_t) = 0$. We show the derivation for $\partial\boldsymbol{q}_t/\partial\boldsymbol{u}_t$ and how to compute Eq. 4.7b efficiently through adjoint method; the same approach can be used for computing others and Eq. 4.7a.

We apply implicit function theorem on $g(\boldsymbol{q}_{t-1}, \dot{\boldsymbol{q}}_{t-1}, \boldsymbol{u}_t, \boldsymbol{q}_t) = 0$,

$$\frac{\mathrm{d}g}{\mathrm{d}\boldsymbol{u}_t} \equiv 0 \Rightarrow \frac{\partial g}{\partial\boldsymbol{q}_t}\frac{\partial\boldsymbol{q}_t}{\partial\boldsymbol{u}_t} + \frac{\partial g}{\partial\boldsymbol{u}_t} \equiv 0 \Rightarrow \frac{\partial\boldsymbol{q}_t}{\partial\boldsymbol{u}_t} = -\left(\frac{\partial g}{\partial\boldsymbol{q}_t}\right)^{-1}\frac{\partial g}{\partial\boldsymbol{u}_t}. \tag{4.9}$$

Plugging this into Eq. (4.7b):

$$\frac{\partial\mathcal{L}}{\partial\boldsymbol{u}_t} = \frac{\partial\mathcal{L}_t}{\partial\boldsymbol{u}_t} - \underbrace{\left(\frac{\partial\mathcal{L}}{\partial\boldsymbol{q}_t} + \frac{\partial\mathcal{L}}{\partial\mathbf{v}_t}\frac{\partial\mathbf{v}_t}{\partial\boldsymbol{q}_t} + \frac{\partial\mathcal{L}}{\partial T_t}\left(\frac{\partial T_t}{\partial\boldsymbol{q}_t} + \frac{\partial T_t}{\partial\dot{\boldsymbol{q}}_t}\frac{\partial\dot{\boldsymbol{q}}_t}{\partial\boldsymbol{q}_t}\right)\right)}_{\boldsymbol{b}}\underbrace{\left(\frac{\partial g}{\partial\boldsymbol{q}_t}\right)^{-1}}_{A}\frac{\partial g}{\partial\boldsymbol{u}_t}. \tag{4.10}$$

To efficiently calculate Eq. (4.10), we apply adjoint method to first solve $\boldsymbol{c}$ from

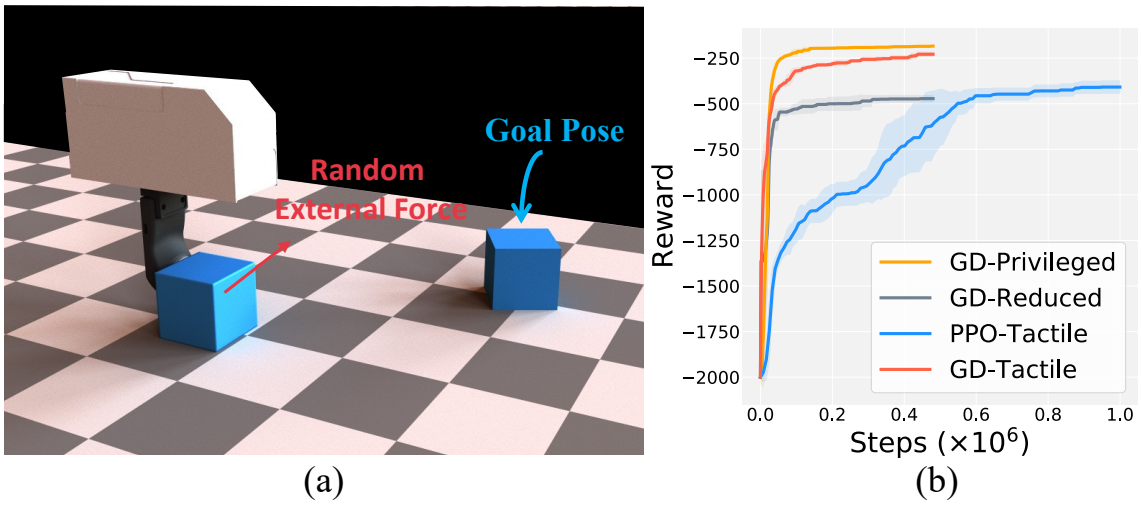


Figure 4-4: **Tactile-based box pushing task.** **(a)** The goal of the gripper policy is to use its tactile feedback to push a box to a randomized target position/orientation. A time-varying external force is randomly applied on the box during the task. **(b)** the training curve for each policy variation is averaged from the five independent runs with different random seeds.

the linear equation $A^\top \boldsymbol{c} = \boldsymbol{b}^\top$ and then compute Eq. (4.7b) as

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{u}_t} = \frac{\partial \mathcal{L}_t}{\partial \boldsymbol{u}_t} - \boldsymbol{c}^\top \frac{\partial g}{\partial \boldsymbol{u}_t}. \tag{4.11}$$

To further speed up the backward pass of $\frac{d\mathcal{L}}{d\boldsymbol{u_t}}$, during the forward pass, we store some auxiliary variables, such as the matrix factors of the final Newton iteration of each time step and the partial derivatives of the loss function. Then during the backward pass, we compute the final derivative with a block banded triangular solver using these auxiliary variables stored during the forward pass without the need to re-compute them.

With the gradient $\frac{d\mathcal{L}}{d\theta}$ computed, we can simply apply any gradient-based numerical optimizer such as Adam [104] to optimize the policy π_θ for the given task.

4.1.5 Experiments

We design a box pushing task similar to [105] to demonstrate how we can leverage the provided analytical gradients from our differentiable simulator to help learn

Figure 4-5: **Visualization of the tactile sensor layouts on a WSG-50 Gripper.**

	Pos. Error	Ori. Error
GD-Privileged	$0.037 \pm 0.002m$	$0.043 \pm 0.003°$
GD-Reduced	$0.126 \pm 0.009m$	$0.255 \pm 0.021°$
PPO-Tactile	$0.123 \pm 0.034m$	$0.241 \pm 0.123°$
GD-Tactile	$\mathbf{0.058 \pm 0.003}m$	$\mathbf{0.074 \pm 0.020}°$

Table 4.1: **Metrics comparison on box pushing task.** We compute the final position/ orientation errors of the best policy in each run and average the metrics from five runs for each policy variation. *GD-Privileged* gives a reference of the best possible metrics, and without the privileged state information of the box, our *GD-Tactile* achieves much better position error and rotation error than other two variations.

tactile-based control policies better and faster.

Task Specification As shown in Fig. 4-4(a), the task here is to use a WSG-50 parallel-jaw gripper (with only one finger kept) to push the box to a randomly sampled goal location and orientation. The gripper has a tactile pad installed at fingertip with a tactile marker resolution of 13×10 as shown in Figure 4-5. The ranges of the goal location coordinates are $x_g \in [0.15 \text{ m}, 0.25 \text{ m}], y_g \in [-0.2 \text{ m}, 0.2 \text{ m}]$. The range of the goal orientation is $\alpha_g \in [y_g\pi - \pi/16, y_g\pi + \pi/16]$. The initial position of the box is randomly disturbed ($[-0.02 \text{ m}, 0.02 \text{ m}]$ along the direction of the finger surface). A random external force f_{ext} (with $f^x_{\text{ext}}, f^y_{\text{ext}} \in [-1, 1]$ N) is applied continually on the box, which changes every 0.25 s. The control frequency is 40 Hz.

Reward Function The reward function is defined at each control step as

$$\mathcal{R}_t = \mathcal{R}_{\text{pos}} + \mathcal{R}_{\text{rot}} + \mathcal{R}_{\text{touch}} + \mathcal{R}_{\text{u}} \tag{4.12a}$$

$$\mathcal{R}_{\text{pos}} = -0.01\big(\frac{\|p_{xy} - [x_g, y_g]\|}{\sigma_{\text{pos}}}\big)^2, \quad \sigma_{\text{pos}} = 0.01 \text{ m} \tag{4.12b}$$

$$\mathcal{R}_{\text{rot}} = -0.1\big(\frac{\alpha - \alpha_g}{\sigma_{\text{rot}}}\big)^2, \quad \sigma_{\text{rot}} = \frac{\pi}{36} \text{ rad} \tag{4.12c}$$

$$\mathcal{R}_{\text{touch}} = -\big(\frac{\|p_{\text{finger}} - p_{\text{box}}\|}{\sigma_{\text{touch}}}\big)^2, \quad \sigma_{\text{touch}} = 0.02 \text{ m} \tag{4.12d}$$

$$\mathcal{R}_{\text{u}} = -0.1\|u\|^2, \tag{4.12e}$$

where p_{xy} is the position of the box in the x-y plane, α is the rotation angle of the box around the vertical axis (z axis), p_{finger} is the position of the center of the gripper finger, p_{box} is the position of the center of the box surface closest to the finger, and u is the policy action (normalized to $[-1, 1]$).

Comparing Policy Learning Algorithms We train the control policies through four different combinations of learning algorithms and observation spaces.

1) *GD-Privileged*: This variation uses the gradient-based optimizer Adam by utilizing the analytical policy gradients computed from our differentiable simulation. The policy observation contains all the privileged state information of the gripper, the box, and the goal. This policy provides an upper-bound performance reference.

2) *GD-Reduced*: Similar to *GD-Privileged*, except that the observation space only contains the state information that can be acquired on a real system such as the gripper state and the goal.

3) *GD-Tactile*: Other than the state information used in *GD-Reduced*, we also include the tactile sensor readings in the policy input. The policy is trained using the analytical policy gradients.

4) *PPO-Tactile*: Similar to *GD-Tactile*, but the policy is trained by PPO.

All the policies are trained to maximize the same reward function. We run each variation five times with different random seeds, and plot their training curves av-

eraged from five runs in Fig. 4-4(b). We also randomly sample 300 goal poses and measure the final position and orientation errors between the box and the goal pose of the best policy from each run, and report the average metrics across five seeds in Table 4.1. The results show that when neither state information of the box or tactile information is available (*i.e.*, *GD-Reduced*), the policy cannot reliably push the box to the target location since the gripper has no any clue when the box goes outside of the control of the gripper due to the random initial box position perturbation and the random external forces. With tactile information feedback (*i.e.*, *PPO-Tactile* and *GD-Tactile*), the gripper has this tactile information to keep the gripper touching the box and allowing it to push the box to the goal effectively. However, the high dimensional tactile observation space results in higher computational cost with PPO which relies on stochastic samples to estimate the policy gradients. In contrast, with the help of our differentiable tactile simulation, *GD-Tactile* makes use of the analytical policy gradients and leads to faster policy learning and better policy performance. This experiment demonstrates the necessity of using the tactile feedback in the pushing task, and shows that with our differentiable simulation, a simple gradient descent optimizer with backpropagation through time gradient computation can significantly improve the efficiency of the policy training and the performance of the trained policy.

4.1.6 Summary

We presented an efficient differentiable simulator that can handle dense tactile force fields with both normal and shear components. Our contact and tactile model are based on penalty-based approach. We demonstrate that by simply applying backpropagation through time to compute the policy gradient, the first-order gradient-based optimizer can achieve significantly faster convergence rate and find better policy than the model-free reinforcement learning approaches on a tactile-based box pushing task. However, how to effectively leverage analytical gradients for more complex tactile-based tasks is still an open question due to the well-known local minima problems of

gradient-based optimization and it require more advanced policy learning algorithms such as the one we will describe in the next section.

4.2 Accelerated Policy Learning with Parallel Differentiable Simulation

4.2.1 Motivation

Learning control policies is an important task in robotics and computer animation. Among various policy learning techniques, reinforcement learning (RL) has been a particularly successful tool to learn policies for systems ranging from robots (*e.g.,* Cheetah, Shadow Hand) [50, 54] to complex animation characters (*e.g.,* muscle-actuated humanoids) [1] using only high-level reward definitions. Despite this success, RL requires large amounts of training data to approximate the policy gradient, making learning expensive and time-consuming, especially for high-dimensional problems (Figure 4-6, Right). The recent development of differentiable simulators opens up new possibilities for accelerating the learning and optimization of control policies. A differentiable simulator may provide accurate first-order gradients of the task performance reward with respect to the control inputs. As shown in the previous section, such additional information potentially allows the use of efficient gradient-based methods to optimize policies. As recently [106] show, however, despite the availability of differentiable simulators, it has not yet been convincingly demonstrated that they can effectively accelerate policy learning in complex high-dimensional and contact-rich tasks, such as some traditional RL benchmarks. There are several reasons for this:

1) Local minima may cause gradient-based optimization methods to stall.
2) Numerical gradients may vanish/explode along the backward path for long trajectories.
3) Discontinuous optimization landscapes can occur during policy failures/early termination.

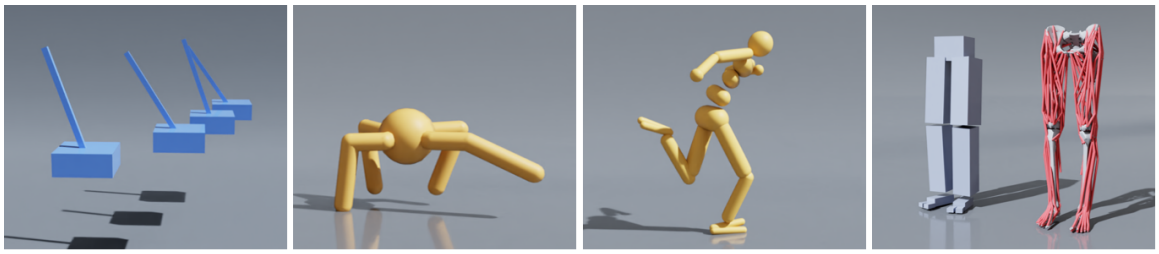


Figure 4-6: **Environments**: Here are some of our environments for evaluation. Three classical physical control RL benchmarks of increasing difficulty, from left: Cartpole Swing Up + Balance, Ant, and Humanoid. In addition, we train the policy for the high-dimensional muscle-tendon driven Humanoid MTU model from [1]. Whereas model-free reinforcement learning (PPO, SAC) needs many samples for such high-dimensional control problems, SHAC scales efficiently through the use of analytic gradients from differentiable simulation with a parallelized implementation, both in sample complexity and wall-clock time.

Because of these challenges, previous work has been limited to the optimization of open-loop control policies with short task horizons [40, 42], or the optimization of policies for relatively simple tasks (*e.g.,* contact-free environments) [71, 38]. In this work, we explore the question: *Can differentiable simulation accelerate policy learning in tasks with continuous closed-loop control and complex contact-rich dynamics?*

Inspired by actor-critic RL algorithms [107], we propose an approach to effectively leverage differentiable simulators for policy learning. We alleviate the problem of local minima by using a critic network that acts as a smooth surrogate to approximate the underlying noisy reward landscape resulted by complex dynamics and occurrences of policy failures (Figure 4-7). In addition, we use a truncated learning window to shorten the backpropagation path to address problems with vanishing/exploding gradients, reduce memory requirements, and improve learning efficiency.

A further challenge with differentiable simulators is that the backward pass typically introduces some computational overhead compared to optimized forward-dynamics physics engines. To ensure meaningful comparisons, we must ensure that our learning method not only improves sample-efficiency, but also wall-clock time. GPU-based physics simulation has shown remarkable effectiveness for accelerating model-free RL algorithms [108, 109], given this, we develop a GPU-based *differentiable* simulator that can compute gradients of standard robotics models over many environments in parallel. Our PyTorch-based simulator allows us to connect high-quality simulation

with existing algorithms and tools.

To the best of our knowledge, this work is the first to provide a fair and comprehensive comparison between gradient-based and RL-based policy learning methods, where fairness is defined as (a) benchmarking on both RL-favored tasks and differentiable-simulation-favored tasks, (b) testing complex tasks (*i.e.*, contact-rich tasks with long task horizons), (c) comparing to the state-of-the-art implementation of RL algorithms, and (d) comparing both sample efficiency and wall-clock time. We evaluate our method on standard RL benchmark tasks, as well as a high-dimensional character control task with over 150 actuated degrees of freedom (some of tasks are shown in fig:envs). We refer to our method as Short-Horizon Actor-Critic (SHAC), and our experiments show that SHAC outperforms state-of-the-art policy learning methods in both sample-efficiency and wall-clock time.

4.2.2 GPU-Based Differentiable Dynamics Simulation

We introduced our differentiable rigid body simulator in Section 4.1. Here we introduce our another differentiable rigid body simulator which parallelizes the simulation with GPU. Briefly recap the simulation dynamics here. Conceptually, we treat the simulator as an abstract function $\mathbf{s}_{t+1} = \mathcal{F}(\mathbf{s}_t, \mathbf{a}_t)$ that takes a state $\mathbf{s}$ from a time $t \rightarrow t+1$, where $\mathbf{a}$ is a vector of actuation controls applied during that time-step (may represent joint torques, or muscle contraction signals depending on the problem).

To model the dynamics function $\mathcal{F}$, our physics simulator solves the forward dynamics equations

$$\mathbf{M}\ddot{\mathbf{q}} = \mathbf{J}^T\mathbf{f}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{c}(\mathbf{q}, \dot{\mathbf{q}}) + \boldsymbol{\tau}(\mathbf{q}, \dot{\mathbf{q}}, \mathbf{a}), \tag{4.13}$$

where $\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}$ are joint coordinates, velocities and accelerations respectively, $\mathbf{f}$ represents external forces, $\mathbf{c}$ includes Coriolis forces, and $\boldsymbol{\tau}$ represents joint-space actuation. The mass matrix $\mathbf{M}$, and Jacobian $\mathbf{J}$, are computed in parallel using one thread per-environment. We use the composite rigid body algorithm (CRBA) to compute

articulation dynamics which allows us to cache the resulting matrix factorization at each step (obtained using parallel Cholesky decomposition) for re-use in the backwards pass. After determining joint accelerations $\ddot{\mathbf{q}}$ we perform a semi-implicit Euler integration step to obtain the updated system state $\mathbf{s} = (\mathbf{q}, \dot{\mathbf{q}})$.

We use the same penalty-based frictional contact model as described in Section 4.1.2. To model joint limits, a similar continuous penalty-based force is applied:

$$\boldsymbol{f}_{\text{limit}} = \begin{cases} k_{\text{limit}}(q_{\text{lower}} - q), & q < q_{\text{lower}} \\ k_{\text{limit}}(q_{\text{upper}} - q), & q > q_{\text{upper}} \end{cases} \tag{4.14}$$

where k_{limit} is the joint limit stiffness, and $[q_{\text{lower}}, q_{\text{upper}}]$ is the bound for the joint angle q.

For simple environments we actuate our agents using torque-based control, in which the policy outputs $\boldsymbol{\tau}$ at each time-step. For the more complex case of muscle-actuation, each muscle consists of a list of attachment sites that may pass through multiple rigid bodies, and the policy outputs activation values for each muscle. A muscle activation signal generates purely contractive forces (mimicking biological muscle) with maximum force prescribed in advance [1].

We build our differentiable simulator on PyTorch [110] and use a source-code transformation approach to generate forward and backward versions of our simulation kernels [111, 41]. We parallelize the simulator over environments using distributed GPU kernels for the dense matrix routines and evaluation of contact and joint forces.

4.2.3 Optimization Landscape Analysis

Although smoothed physical models improve the local optimization landscape, the combination of forward dynamics and the neural network control policy renders each simulation step non-linear and non-convex. This problem is exacerbated when thousands of simulation steps are concatenated and the actions in each step are coupled by a feedback control policy. The complexity of the resulting reward landscape leads simple gradient-based methods to easily become trapped in local optima.

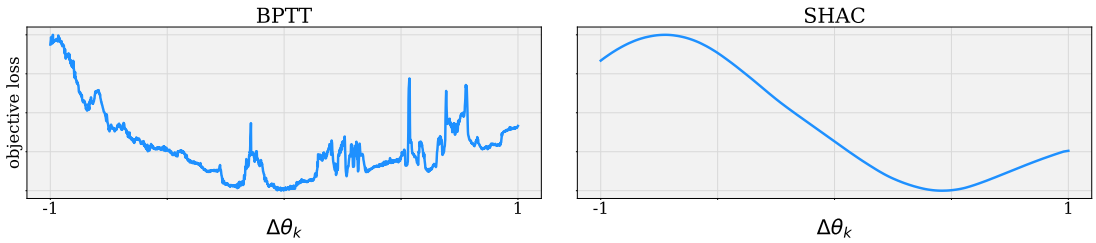


Figure 4-7: **Landscape comparison between BPTT and SHAC.** We select one single weight from a policy and change its value by $\Delta\theta_k \in [-1, 1]$ to plot the task loss landscapes of BPTT and SHAC w.r.t. one policy parameter. The task horizon is $H = 1000$ for BPTT, and the short horizon length for our method is $h = 32$. As we can see, longer optimization horizons lead to noisy loss landscape that are difficult to optimize, and the landscape of our method can be regarded as a smooth approximation of the real landscape.

Furthermore, to handle agent failure (*e.g.,* a humanoid falling down) and improve sample efficiency, early termination techniques are widely used in policy learning algorithms [112]. Although these have proven effective for model-free algorithms, early termination introduces additional discontinuities to the optimization problem, which makes methods based on analytical gradients less successful.

To analyze this problem, inspired by previous work [113], we plot the optimization landscape in Figure 4-7 (Left) for a humanoid locomotion problem with a 1000-step task horizon. Specifically, we take a trained policy, perturb the value of a single parameter θ_k in the neural network, and evaluate performance for the policy variations. As shown in the figure, with long task horizons and early termination, the landscape of the humanoid problem is highly non-convex and discontinuous. In addition, the norm of the gradient $\frac{\partial \mathcal{L}}{\partial \theta}$ computed from backpropagation can be larger than 10^6. Thus, due to the noisy landscape and exploded gradient problems of BPTT method, most previous works based on differentiable simulation focus on short-horizon tasks with contact-free dynamics and no early termination, where pure gradient-based optimization (*e.g.*, BPTT) can work successfully.

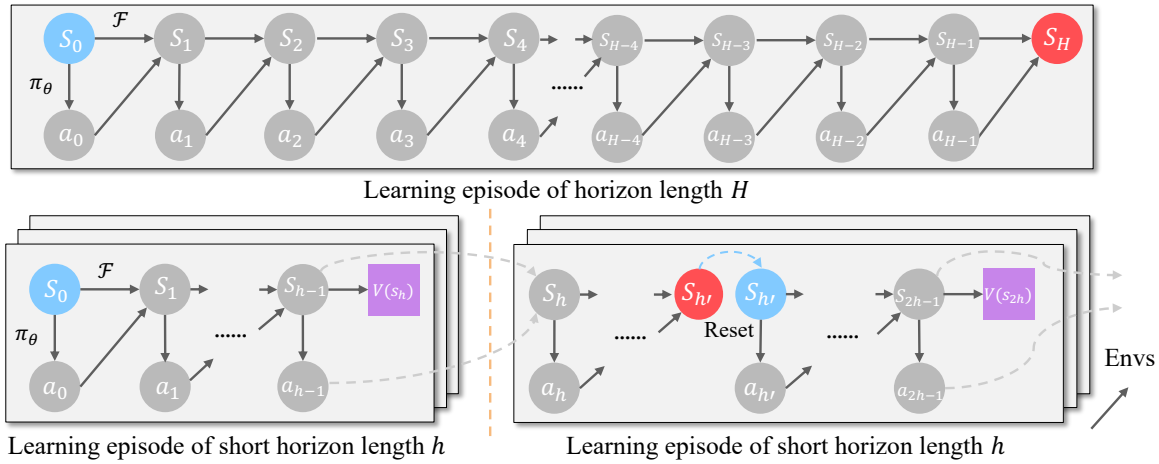


Figure 4-8: **Computation graph of BPTT and SHAC**. **Top**: BPTT propagates gradients through an entire trajectory in each learning episode. This leads to noisy loss landscapes, increased memory, and numerical gradient problems. **Bottom**: SHAC subdivides the trajectory into short optimization windows across learning episodes. This results in a smoother surrogate reward function and reduces memory requirements, enabling parallel sampling of many trajectories. The environment is reset upon early termination happens. Solid arrows denote gradient-preserving computations; dashed arrows denote locations at which the gradients are cut off.

4.2.4 Short-Horizon Actor-Critic (SHAC)

To resolve the aforementioned issues of gradient-based policy learning, we propose the Short-Horizon Actor-Critic method (SHAC). Our method concurrently learns a policy network (*i.e.*, actor) π_θ and a value network (*i.e.*, critic) V_ϕ during task execution, and splits the entire task horizon into several sub-windows of smaller horizons across learning episodes (Figure 4-8). A multi-step reward in the sub-window plus a terminal value estimation from the learned critic is used to improve the policy network. The differentiable simulation is used to backpropagate the gradient through the states and actions inside the sub-windows to provide an accurate policy gradient. The trajectory rollouts are then collected and used to learn the critic network in each learning episode.

Specifically, we model each of our control problems as a finite-horizon Markov decision process (MDP) with state space $\mathcal{S}$, action space $\mathcal{A}$, reward function $\mathcal{R}$, transition function $\mathcal{F}$, initial state distribution $\mathcal{D}_{\mathbf{s}_0}$, and task horizon H. At each step, an action vector $\mathbf{a}_t$ is computed by a feedback policy $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$. While our

method does not constrain the policy to be deterministic or stochastic, we use the stochastic policy in our experiments to facilitate extra exploration. Specifically, the action is sampled by $\mathbf{a}_t \sim \mathcal{N}(\mu_\theta(\mathbf{s}_t), \sigma_\theta(\mathbf{s}_t))$. The transition function $\mathcal{F}$ is modeled by our differentiable simulation (Section 4.2.2). A single-step reward $r_t = \mathcal{R}(\mathbf{s}_t, \mathbf{a}_t)$ is received at each step. The goal of the problem is to find the policy parameters θ that maximize the expected finite-horizon reward.

Our method works in an on-policy mode as follows. In each learning episode, the algorithm samples N trajectories $\{\tau_i\}$ of short horizon $h \ll H$ in parallel from the simulation, which continue from the end states of the trajectories in the previous learning episode. The following policy loss is then computed:

$$\mathcal{L}_\theta = -\frac{1}{Nh} \sum_{i=1}^{N} \left[\Big(\sum_{t=t_0}^{t_0+h-1} \gamma^{t-t_0} \mathcal{R}(\mathbf{s}_t^i, \mathbf{a}_t^i) \Big) + \gamma^h V_\phi(\mathbf{s}_{t_0+h}^i) \right], \tag{4.15}$$

where $\mathbf{s}_t^i$ and $\mathbf{a}_t^i$ are the state and action at step t of the i-th trajectory, and $\gamma < 1$ is a discount factor introduced to stabilize the training. Special handling such as resetting the discount ratio is conducted when task termination happens during trajectory sampling.

To compute the gradient of the policy loss $\frac{\partial \mathcal{L}_\theta}{\partial \theta}$, we treat the simulator as a differentiable layer in the PyTorch computation graph and perform regular backpropagation. We apply reparameterization sampling method to compute the gradient for the stochastic policy. Our algorithm then updates the policy using one step of Adam [104]. The differentiable simulator plays a critical role here, as it allows us to fully utilize the underlying dynamics linking states and actions, as well as optimize the policy, producing better short-horizon reward inside the trajectory and a more promising terminal state for the sake of long-term performance. We note that the gradients are cut off between learning episodes to prevent unstable gradients during long-horizon backpropagation.

After we update the policy π_θ, we use the trajectories collected in the current learning episode to train the value function V_ϕ. The value function network is trained

by the following MSE loss:

$$\mathcal{L}_\phi = \mathop{\mathbb{E}}_{\mathbf{s}\in\{\tau_i\}}\Big[\|V_\phi(\mathbf{s}) - \tilde{V}(\mathbf{s})\|^2\Big], \tag{4.16}$$

where $\tilde{V}(\mathbf{s})$ is the estimated value of state $\mathbf{s}$, and is computed from the sampled short-horizon trajectories through a td-λ formulation [114], which computes the estimated value by exponentially averaging different k-step returns to balance the variance and bias of the estimation:

$$\tilde{V}(\mathbf{s}_t) = (1-\lambda)\Bigg(\sum_{k=1}^{h-t-1} \lambda^{k-1} G_t^k\Bigg) + \lambda^{h-t-1} G_t^{h-t}, \tag{4.17}$$

where $G_t^k = \Big(\sum_{l=0}^{k-1} \gamma^l r_{t+l}\Big) + \gamma^k V_\phi(\mathbf{s}_{t+k})$ is the k-step return from time t. The estimated value $\tilde{V}(\mathbf{s})$ is treated as constant during critic training, as in regular actor-critic RL methods. In other words, the gradient of Eq. 4.16 does not flow through the states and actions in Eq. 4.17.

We further utilize the target value function technique [115] to stabilize the training by smoothly transitioning from the previous value function to the newly fitted one, and use the target value function $V_{\phi'}$ to compute the policy loss (Eq. 4.15) and to estimate state values (Eq. 4.17). In addition, we apply observation normalization as is common in RL algorithms, which normalizes the state observation by a running mean and standard deviation calculated from the state observations in previous learning episodes. The pseudo code of our method is provided in Algorithm 1.

Our actor-critic formulation has several advantages that enable it to leverage simulation gradients effectively and efficiently. First, the terminal value function absorbs noisy landscape over long dynamics horizons and discontinuity introduced by early termination into a smooth function, as shown in Figure 4-7 (Right). This smooth surrogate formulation helps reduce the number of local spikes and alleviates the problem of easily getting stuck in local optima. Second, the short-horizon episodes avoid numerical problems when backpropagating the gradient through deeply nested update chains. Finally, the use of short-horizon episodes allows us to update the actor more

Algorithm 1: SHAC (Short-Horizon Actor-Critic) Policy Learning

Initialize policy π_θ, value function V_ϕ, and target value function $V_{\phi'} \leftarrow V_\phi$.
for *learning episode* $\leftarrow 1, 2, ..., M$ **do**
  Sample N short-horizon trajectories of length h by the parallel differentiable simulation from the end states of the previous trajectories.
  Compute the policy loss $\mathcal{L}_\theta$ defined in Eq. 4.15 from the sampled trajectories and $V_{\phi'}$.
  Compute the analytical gradient $\frac{\partial \mathcal{L}_\theta}{\partial \theta}$ and update the policy π_θ one step with Adam.
  Compute estimated values for all the states in sampled trajectories with Eq. 4.17.
  Fit the value function V_ϕ using the critic loss defined in Eq. 4.16.
  Update target value function: $V_{\phi'} \leftarrow \alpha V_{\phi'} + (1 - \alpha) V_\phi$.
end for

frequently, which, when combined with parallel differentiable simulation, results in a significant speed up of training time.

4.2.5 Experiments

We design experiments to investigate five questions: (1) How does our method compare to the state-of-the-art RL algorithms on classical RL control tasks, in terms of both sample efficiency and wall-clock time efficiency? (2) How does our method compare to the previous differentiable simulation-based policy learning methods? (3) Does our method scale to high-dimensional problems? (4) Is the terminal critic necessary? (5) How important is the choice of short horizon length h for our method?

4.2.5.1 Experiment Setup

To ensure a fair comparison for wall-clock time performance, we run all algorithms on the same GPU model (TITAN X) and CPU model (Intel Xeon(R) E5-2620). Furthermore, we conduct extensive hyperparameter searches for all algorithms and report the performance of the best hyperparameters settings for each problem. For our method, we run for 500 learning episodes for *CartPole Swing Up* problem and for 2000 learning episodes for other five problems. For baseline algorithms, we run

each of them for sufficiently long time in order to acquire the policies with the highest rewards. We run each algorithm with five individual random seeds, and report the average performance.

4.2.5.2 Benchmark Control Problems

For comprehensive evaluations, we select six broad control tasks, including five classical RL tasks across different complexity levels (*CartPole Swing Up*, *HalfCheetah*, *Hopper*, *Ant* and *Humanoid*) as well as one high-dimensional control task with a large action space (*Humanoid MTU*). All tasks have stochastic initial states to further improve the robustness of the learned policy. We introduce four representative tasks in Figure 4-6 in this thesis.

CartPole Swing Up: CartPole Swing Up is one of the simplest classical RL control tasks. In this problem, the control policy has to swing up the pole to the upward direction and keeps it in that direction as long as possible. We have 5-dimensional observation space as shown in Table 4.2 and 1-dimensional action to control the torque applied to the prismatic joint of the cart base.

Table 4.2: **Observation vector of CartPole Swing Up problem**

Observation	**Degrees of Freedom**
position of cart base: x	1
velocity of the cart base: $\dot{x}$	1
sine and cosine of the pole angle: $\sin(\theta), \cos(\theta)$	2
angular velocity of pole: $\dot{\theta}$	1

The single-step reward is defined as:

$$\mathcal{R} = -\theta^2 - 0.1\dot{\theta}^2 - 0.05x^2 - 0.1\dot{x}^2 \tag{4.18}$$

The initial state is randomly sampled. The task horizon is 240 steps and there is no early termination in this environment so that it can be used to test the algorithms which are not compatible with early termination strategy.

Ant: In this problem, a four-legged ant is controlled to run forward as fast as possible. We have 37-dimensional observation space as shown in Table 4.3 and 8-dimensional action to control the torque applied to each joint.

Table 4.3: **Observation vector of Ant problem**

Observation	**Degrees of Freedom**
height of the base: h	1
rotation quaternion of the base	4
linear velocity of the base: v	3
angular velocity of the base	3
joint angles	8
joint angle velocities	8
up and heading vectors projections	2
actions in last time step	8

The single-step reward is defined as:

$$\mathcal{R} = \mathcal{R}_v + 0.1\mathcal{R}_{up} + \mathcal{R}_{heading} + \mathcal{R}_{height}, \tag{4.19}$$

where $\mathcal{R}_v = v_x$ is the forward velocity, $\mathcal{R}_{up}$ = projection(upward direction) encourages the agent to be vertically stable, $\mathcal{R}_{heading}$ = projection(forward direction) encourages the agent to run straight forward, and $\mathcal{R}_{height} = h - 0.27$ is the height reward.

The initial state is randomly sampled. The task horizon is 1000 steps and early termination is triggered when the height of the ant is lower than 0.27m.

Humanoid: In this problem, a humanoid robot is controlled to run forward as

fast as possible. We have a 76-dimensional observation space as shown in Table 4.4 and a 21-dimensional action vector to control the torque applied to each joint.

Table 4.4: **Observation vector of Humanoid problem**

Observation	**Degrees of Freedom**
height of the torso: h	1
rotation quaternion of the torso	4
linear velocity of the torso: v	3
angular velocity of the torso	3
joint angles	21
joint angle velocities	21
up and heading vectors projections	2
actions in last time step	21

The single-step reward is defined as:

$$\mathcal{R} = \mathcal{R}_v + 0.1\mathcal{R}_{up} + \mathcal{R}_{heading} + \mathcal{R}_{height} - 0.002\|\mathbf{a}\|^2, \tag{4.20}$$

where $\mathcal{R}_v = v_x$ is the forward velocity, $\mathcal{R}_{up}$ is the projection of torso on the upward direction encouraging the agent to be vertically stable, $\mathcal{R}_{heading}$ is the projection of torso on the forward direction encouraging the agent to run straight forward, and $\mathcal{R}_{height}$ is defined by:

$$\mathcal{R}_{height} = \begin{cases} -200\Delta_h^2, & \Delta_h < 0 \\ 10\Delta_h, & \Delta_h \geq 0 \end{cases} \tag{4.21}$$

$$\Delta_h = \text{clip}(h - 0.84, -1, 0.1) \tag{4.22}$$

The initial state is randomly sampled. The task horizon is 1000 steps and early termination is triggered when the height of torso is lower than 0.74m.

Humanoid MTU: To assess how our method scales to high-dimensional tasks,

we examine the challenging problem of muscle-actuated humanoid control (Figure 4-6, Right). In this problem, a lower body of the humanoid model from [1] is actuated by 152 muscle-tendon units (MTUs). Each MTU contributes one actation degree of freedom that controls the contractile force applied to the attachment sites on the connected bodies. We have a 53-dimensional observation space as shown in Table 4.5 and a 152-dimensional action vector.

Table 4.5: **Observation vector of Humanoid MTU problem**

Observation	**Degrees of Freedom**
height of the pelvis: h	1
rotation quaternion of the pelvis	4
linear velocity of the pelvis: v	3
angular velocity of the pelvis	3
joint angles	22
joint angle velocities	18
up and heading vectors projections	2

The single-step reward is defined as:

$$\mathcal{R} = \mathcal{R}_v + 0.1\mathcal{R}_{up} + \mathcal{R}_{heading} + \mathcal{R}_{height} - 0.001\|\mathbf{a}\|^2, \tag{4.23}$$

where $\mathcal{R}_v = v_x$ is the forward velocity, $\mathcal{R}_{up}$ = projection(upward direction) encourages the agent to be vertically stable, $\mathcal{R}_{heading}$ = projection(forward direction) encourages the agent to run straight forward, and $\mathcal{R}_{height}$ is defined by:

$$\mathcal{R}_{height} = \begin{cases} -200\Delta_h^2, & \Delta_h < 0 \\ 4\Delta_h, & \Delta_h \geq 0 \end{cases} \tag{4.24}$$

$$\Delta_h = \text{clip}(h - 0.51, -1, 0.05) \tag{4.25}$$

The initial state is randomly sampled. The task horizon is 1000 steps and early

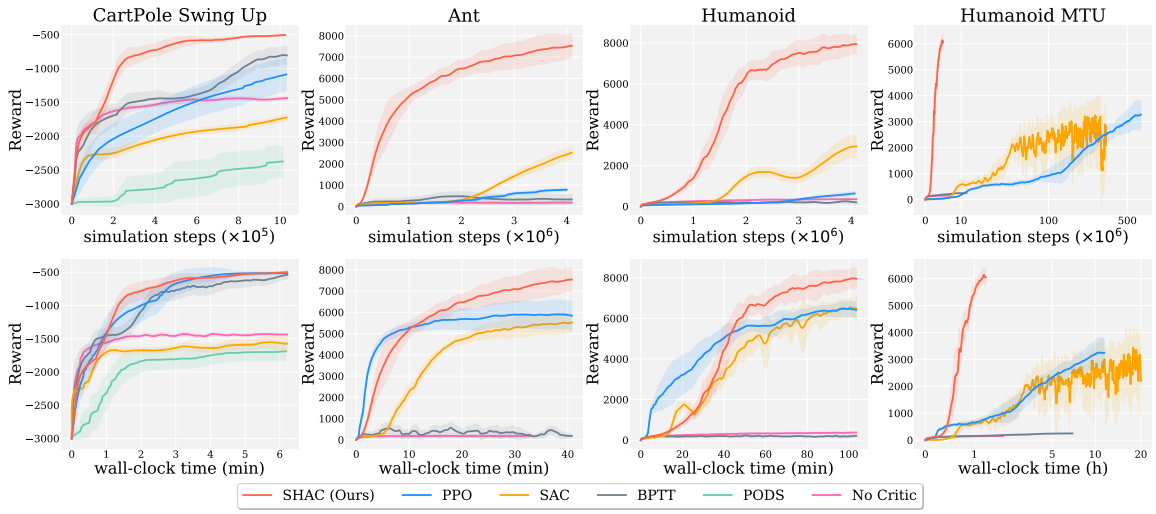


Figure 4-9: **Learning curves comparison on four benchmark problems.** Each column corresponds to a particular problem, with the top plot evaluating sample efficiency and the bottom plot evaluating wall-clock time efficiency. For better visualization, we truncate all the curves up to the maximal simulation steps/wall-clock time of our method (except for Humanoid MTU), and we provide the full plots in Appendix **??**. Each curve is averaged from five random seeds, and the shaded area shows the standard deviation. SHAC is more sample efficient than all baselines. Model-free baselines are competitive on wall-clock time on pedagogical environments such as the cartpole, but are much less effective as the problem complexity scales.

termination is triggered when the height of pelvis is lower than 0.46m.

4.2.5.3 Results

Comparison to model-free RL. We compare SHAC with Proximal Policy Optimization (PPO) [60] (on-policy) & Soft Actor-Critic (SAC) [64] (off-policy). We use high-performance implementations from *RL games* [116]. To achieve state-of-the-art performance, we follow [117]: all simulation, reward and observation data remain on the GPU and are shared as PyTorch tensors between the RL algorithm and simulator. The PPO and SAC implementations are parallelized and operate on vectorized states and actions. With PPO we used short episode lengths, an adaptive learning rate, and large mini-batches during training to achieve the best possible performance.

As shown in the first row of Figure 4-9, our method shows significant improvements in sample efficiency over PPO and SAC in three classical RL problems, especially when the dimension of the problem increases (*e.g.*, *Humanoid*). The analytical gradients

provided by the differentiable simulation allow us to efficiently acquire the expected policy gradient through a small number of samples. In contrast, PPO and SAC have to collect many Monte-Carlo samples to estimate the policy gradient.

Model-free algorithms typically have a lower per-iteration cost than methods based on differentiable simulation; thus, it makes sense to also evaluate wall-clock time efficiency instead of sample-efficiency alone. As shown in the second row of Figure 4-9, the wall-clock time performance of PPO, SAC, and our method are much closer than the sample efficiency plot. Interestingly, the training speed of our method is slower than PPO at the start of training. We hypothesize that the target value network in our method is initially requiring sufficient episodes to warm up.

We observe that our method achieves better policies than RL methods in all problems. We hypothesize that, while RL methods are effective at exploration far from the solution, they struggle to accurately estimate the policy gradient near the optimum, especially in complex problems.

To better analyze the wall-clock time performance of our method, we provide the detailed wall-clock time performance breakdown of a single learning episode of our method in Table 4.6. As shown in the table, the forward simulation and backward simulation time scales up when the dimensionality of the problem increases, while the time spent in critic value function training is almost constant across problems. As expected, the differentiable simulation brings extra overhead for its backward gradient computation. Specifically in our differentiable simulation, the backward computation time is roughly $2\times$ of the forward time. This indicates that our method still has room to improve its overall wall-clock time efficiency through the development of more optimized differentiable simulators with fast backward computation.

Comparison with previous gradient-based methods. We compare our approach to three gradient-based learning methods: (1) Backpropagation Through Time (BPTT), which has been widely used in the differentiable simulation literature [40, 38], (2) PODS [71], and (3) Sample Enhanced Model-Based Policy Optimization (SE-MBPO) [70].

	Forward (s)	**Backward** (s)	**Critic Training** (s)
CartPole SwingUp	0.15	0.23	0.34
Ant	0.25	0.37	0.32
Humanoid	0.91	1.97	0.32
SNU Humanoid	0.82	1.66	0.33

Table 4.6: **Wall-clock performance breakdown of a single training episode.** The forward stage includes simulation, reward calculation, and observations. Backward includes the simulation gradient calculation and actor update. Critic training, which is specific to our method, is listed individually, and is generally a small proportion of the overall training time.

1) *BPTT*: The original BPTT method backpropagates gradients over the entire trajectory, which results in exploding gradients as shown in Section 4.2.3. We modify BPTT to work on a shorter window of the tasks (64 steps for *CartPole* and 128 steps for other tasks), and also leverage parallel differentiable simulation to sample multiple trajectories concurrently to improve its time efficiency. As shown in Figure 4-9, BPTT successfully optimizes the policy for the contact-free *CartPole Swing Up* task, whereas it falls into local minima quickly in all other tasks involving contact. For example, the policy that BPTT learns for Ant is a stationary position leaning forward, which is a local minimum.

2) *PODS*: We compare to the first-order version of PODS, as the second-order version requires the full Jacobian of the state with respect to the whole action trajectory, which is not efficiently available in a reverse-mode differentiable simulator (including ours). Since PODS relies on a trajectory optimization step to optimize an open-loop action sequence, it is not clear how to accommodate early termination where the trajectory length can vary during optimization. Therefore, we test PODS performance only on the *CartPole Swing Up* problem. As shown in Figure 4-9, PODS quickly converges to a local optimum and is unable to improve further. This is because PODS is designed to be a method with high gradient exploitation but little exploration. Specifically, the line search applied in the trajectory optimization stage helps it converge quickly, but also prevents

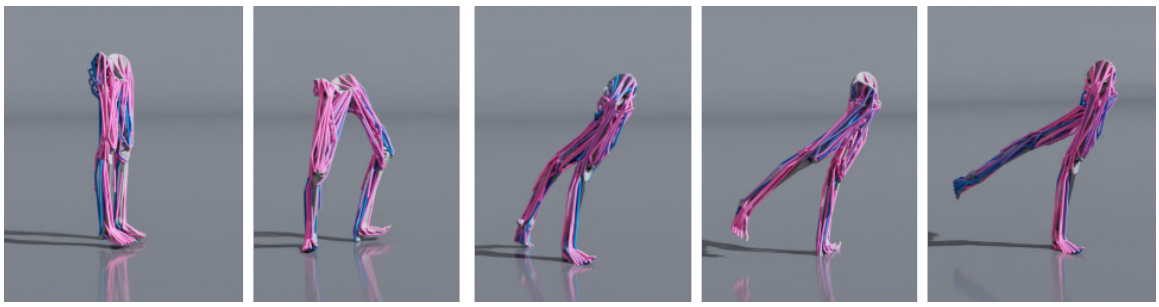


Figure 4-10: **Humanoid MTU**: A sequence of frames from a learned running gait. The muscle unit color indicates the activation level at the current state.

it from exploring more surrounding space. Furthermore, the extra simulation calls introduced by the line search and the slow imitation learning stage make it less competitive in either sample or wall-clock time efficiency.

3) *SE-MBPO*: [70] propose to improve a model-based RL method MBPO [66] by augmenting the rollout samples using data augmentation that relies on the Jacobian from the differentiable simulator. Although SE-MBPO shows high sample efficiency, the underlying model-based RL algorithm and off-policy training lead to a higher wall-clock time. As a comparison, the officially released code for SE-MBPO takes 8 hours to achieve a reasonable policy in the *Ant* problem used by [70], whereas our algorithm takes less than 15 minutes to acquire a policy with the same gait level in our *Ant* problem. Aiming for a more fair comparison, we adapt their implementation to work on our *Ant* problem in our simulator. However, we found that it could not successfully optimize the policy even after considerable hyperparameter tuning. Regardless, the difference in wall-clock time between two algorithms is obvious, and the training time of SE-MBPO is unlikely to be improved significantly by integrating it into our simulation environment. Furthermore, as suggested by [70], SE-MBPO does not generalize well to other tasks, whereas our method can be successfully applied to various complexity levels of tasks.

Scalability to high-dimensional problems. We test our algorithm and RL baselines on the *Humanoid MTU* example to compare their scalability to high-dimensional

problems. With the large 152-dimensional action space, both PPO and SAC struggle to learn the policy as shown in Figure 4-9 (Right). Specifically, PPO and SAC learn significantly worse policies after more than 10 hours of training and with hundreds of millions of samples. This is because the amount of data required to accurately estimate the policy gradient significantly increases as the state and action spaces become large. In contrast, our method scales well due to direct access to the more accurate gradients provided by the differentiable simulation with the reparameterization techniques. To achieve the same reward level as PPO, our approach only takes around 35 minutes of training and 1.7M simulation steps. This results in over 17× and 30× wall-clock time improvement over PPO and SAC, respectively, and 382× and 170× more sample efficiency. Furthermore, after training for only 1.5 hours, our method is able to find a policy that has twice the reward of the best-performing policy from the RL methods. A learned running gait is visualized in Figure 4-10. Such scalability to high-dimensional control problems opens up new possibilities for applying differentiable simulation in computer animation, where complex character models are widely used to provide more natural motion.

Ablation study on the terminal critic. We introduce a terminal critic value in Eq. 4.15 to account for the long-term performance of the policy after the short episode horizon. In this experiment, we evaluate the importance of this term. By removing the terminal critic from Eq. 4.15, we get an algorithmic equivalent to BPTT with a short-horizon window and discounted reward calculation. We apply this no-critic variation on all four problems and plot the training curve in Figure 4-9, denoted by "No Critic". Without a terminal critic function, the algorithm is not able to learn a reasonable policy, as it only optimizes a short-horizon reward of the policy regardless of its long-term behavior.

Robustness of SHAC to hyperparameters. In the previous results, we fine tune the network architectures of policy and value function for each algorithm on each problem to get their maximal performance for a fair comparison. In this section,

we test the robustness of our method by using a fixed network architecture and fixed set of hyperparameters (*e.g.* learning rates) to train on all problems. Specifically, we use a same setting of network architecture and the hyperparameters reported in Table 4.7 in all problems, and plot the training curves in Figure 4-11.

Table 4.7: **A general setting of hyperparameters of SHAC.**

Hyperparameter names	**Values**
short horizon length h	32
number of parallel environments N	64
policy learning rate	0.002
critic learning rate	0.0005
discount factor γ	0.99
value estimation λ	0.95
target value network α	0.995
number of critic training iterations	16
number of critic training minibatches	4

Experiment SHAC with deterministic policy Our method does not constrain the policy to be stochastic or deterministic. We choose to use the stochastic policy in the most experiments for the extra exploration that it provides. In this experiment, we test our method with the deterministic policy choice. Specifically, we change the policy of our method in each problem from stochastic policy to deterministic policy while keeping all other hyperparameters such as network dimensions and learning rates the same. The training curves of the deterministic policy is plotted in Figure 4-12. The results show that our method works reasonably well with deterministic policy, and sometimes the deterministic policy even outperforms the stochastic policy (*e.g. Humanoid*). The small performance drop on the *Ant* problem comes from one single random seed (out of five) which results in a sub-optimal policy.

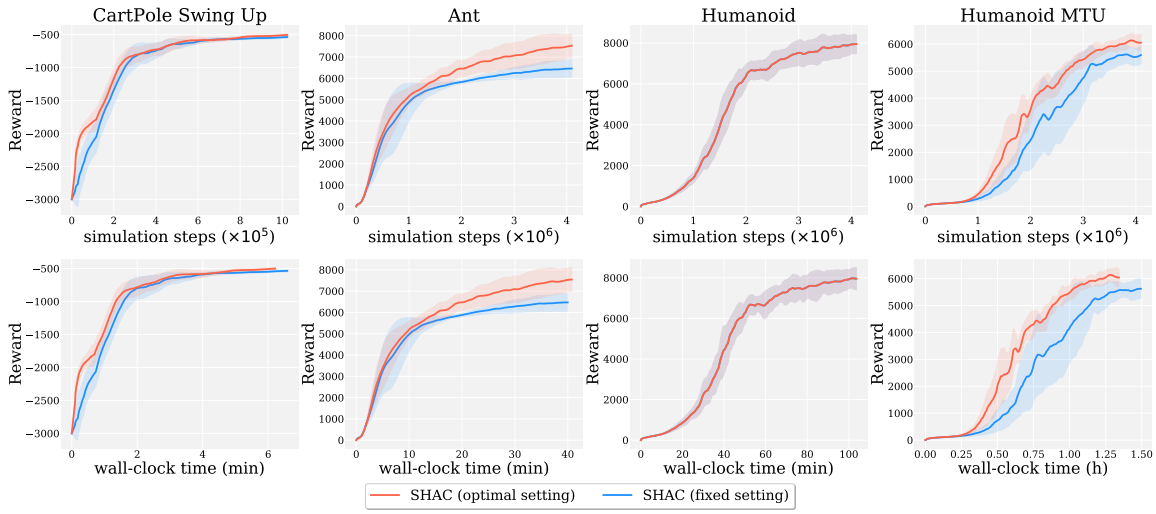


Figure 4-11: **Learning curves of our method with fixed network architectures and learning rates.** We use the same network architectures and learning rates used in *Humanoid* problem on all other problems, and plot the training curves comparison with the ones using optimal settings. The plot shows that our method still performs reasonably well with the fixed network and learning rates settings.

Study of short horizon length h. The choice of horizon length h is important for the performance of our method. h cannot be too small, as it will result in worse value estimation by td-λ (Eq. 4.17) and underutilize the power of the differentiable simulator to predict the sensitivity of future performance to the policy weights. On the other hand, a horizon length that is too long will lead to a noisy optimization landscape and less-frequent policy updates. Empirically, we find that a short horizon length $h = 32$ with $N = 64$ parallel trajectories works well for all tasks in our experiments. We conduct a study of short horizon length on the *Ant* task to show the influence of this hyperparameter. We run our algorithm with six short horizon lengths $h = 4, 8, 16, 32, 64, 128$. We set the corresponding number of parallel trajectories $N = 512, 256, 128, 64, 32, 16$ for the variant, such that each one generates the same amount of samples in single learning episode. We run each variant for the same number of episodes $M = 2000$ with 5 individual random seeds. In Figure 4-13, we report the average reward of the best policies from 5 runs for each variant, as well as the total training time. As expected, the best reward is achieved when $h = 16$ or 32, and the training time scales linearly as h increases.

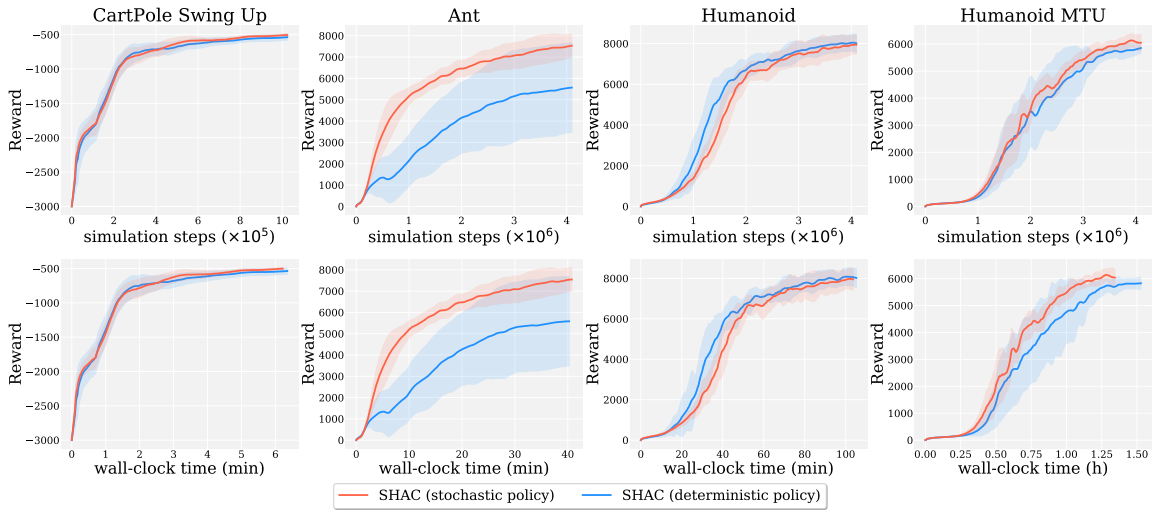


Figure 4-12: **Learning curves of our method with deterministic policy.** We test our method with deterministic policy choice. We use the same network sizes and the hyperparameters as used in the stochastic policy and remove the policy output stochasticity. We run our method on each problem with five individual random seeds. The results show that our method with deterministic policy works reasonably well on all problems, and sometimes the deterministic policy even outperforms the stochastic policy (*e.g.*, *Humanoid*). The small performance drop on the *Ant* problem comes from one single seed (out of five) which results in a sub-optimal policy.

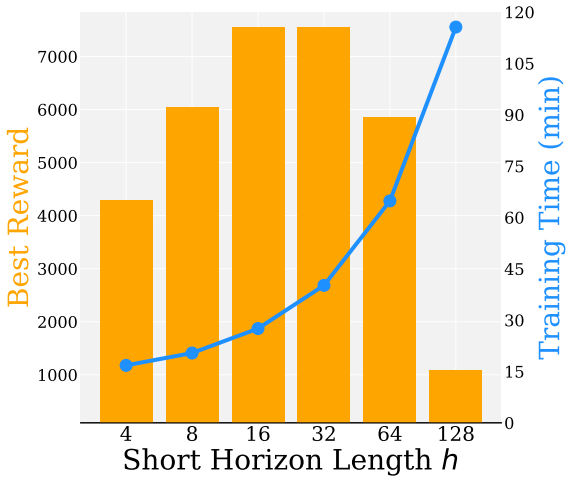


Figure 4-13: **Study of short horizon length h on Ant problem**. A small h results in worse value estimation. A too large h leads to an ill-posed optimization landscape and longer training time.

4.2.6 Summary

In this work, we propose an approach to effectively leverage differentiable simulation for policy learning. At the core is the use of a critic network that acts as a

smooth surrogate to approximate the underlying noisy optimization landscape. In addition, a truncated learning window is adopted to alleviate the problem of exploding/vanishing gradients during deep backward paths. Equipped with the developed parallel differentiable simulation, our method shows significantly higher sample efficiency and wall-clock time efficiency over state-of-the-art RL and gradient-based methods, especially when the problem complexity increases. As shown in our experiments, model-free methods demonstrate efficient learning at the start of training, but SHAC is able to achieve superior performance after a sufficient number of episodes. A compelling future direction for research is how to combine model-free methods with our gradient-based method in order to leverage the strengths of both. Furthermore, in our method, we use a fixed and predetermined short horizon length h throughout the learning process; however, future work may focus on implementing an adaptive short horizon schedule that varies with the status of the optimization landscape.

Chapter 5

Computational Robot Shape and Control Co-Design

While the controller serves as a brain of a robot and a significant amount of research effort has been paid to optimize the control for a fixed robot hardware design, robot hardware actually plays an equally important role as its control algorithm in its task performance. As an example, a quadruped with longer legs can typically run faster than a quadruped with shorter legs no matter how optimal the control algorithm of the short-leg quadruped is. Despite of its importance, co-optimizing both shape and control of a robotic system is still under-explored. Today this process is labor-intensive and time-consuming. For instance, hardware components and control algorithms are typically constructed sequentially making the integration of different modules difficult which necessitates many design iterations. A computational automated algorithm for this co-design purpose is still an active and challenging research question. This is because the hardware shape of a robot introduces a mixed optimization space involving both discrete parameters and continuous parameters as we discussed in Chapter 3. Though we have paced one step forward by proposing effective representation for those different categories of shape parameters, how to efficiently optimize those shape parameters is still questionable. Furthermore, the underlying coupling of the hardware shape and the software control of a robot results in an

enormous optimization space. As different robot shape designs have different optimal control strategies, control optimization, which alone is already a hard enough research topic, now becomes to a sub-problem of a control-shape optimization problem.

Co-optimizing the shape and control typically follows two strategies: (a) a bi-level optimization scheme where a shape optimization serves as an outer loop while a control optimization serves as the inner loop to evaluate each proposed shape design from the outer loop; (b) a joint optimization scheme where the shape parameters and the control parameters are optimized simultaneously. In this chapter, we focus on the robot shape and control co-optimization problems, and argue that different co-optimization schemes are appropriate for different categories of shape parameters. Specifically, we present a learning-based bi-level optimization scheme for optimizing the robot control and discrete shape topology in Section 5.1, and further present a joint optimization approach enhanced by differentiable physics simulation for optimizing the robot control and continuous shape morphology in Section 5.2.

5.1 Co-Optimizing Robot Control and Discrete Shape Topology: Graph Heuristic Search

5.1.1 Motivation

Automating the discovery of the novel robot structure and the controller for given tasks has long been a key research question. This is a particularly challenging research problem as the design space is vast and intractable and there are limited tools for automatically and efficiently exploring it. To enable large scale search and optimization of robots, we have introduced in Chapter 3.1 our novel graph grammar based representation for the manipulator topological structure design space. However, the graph grammar space provides a large combinatorial search space for the optimization algorithm.

To address this challenge, we introduce a novel computational algorithm for simultaneously optimizing the physical structures and controllers of robots. We take the terrestrial robot as a case study for this purpose. Similarly as manipulators, we construct a recursive graph grammar for terrestrial robots that emphasizes mobility and fabricability. The goal of the search algorithm is to take the graph grammar robot structure space as input and generate an optimal robot structure and controller for traversing a given terrain. To provide the performance evaluation for a robot structure design, we use model predictive control (MPC) to provide a stochastic approach to controller learning. To efficiently search the design space of robot graphs, we introduce a novel Graph Heuristic Search algorithm (GHS) which generalizes the knowledge of explored designs to predict performance of unexplored branches of the search space. Specifically, our search algorithm takes a learning-based approach inspired by reinforcement learning, iteratively exploring a large space of robot designs for a given task, and learning a heuristic function to gradually steer that search toward optimal designs. Our learning model takes a neural-based approach, exploiting a graph neural network architecture to provide a fast method for approximating the performance metrics of best designs.

In summary, we present the following key contributions:

- A Graph Heuristic Search method for efficiently searching the design space described with the terrestrial robot grammar. This is bench-marked against alternatives including Monte Carlo tree search and random search.
- A demonstration of terrain-driven optimization using MPC based stochastic evaluation of each proposed design. We show the variety of innovative robot designs that are obtained across six different terrains. In addition, our approach identifies a number of high-performing robots for a single terrain.

The following section of this paper reviews related work presenting current state-of-the-art research. In Section 5.1.2 we provide an overview of the different systems and algorithms used to enable the grammar-driven exploration and optimization of robots. Section 5.1.3 briefly describes the graph grammar which is developed to

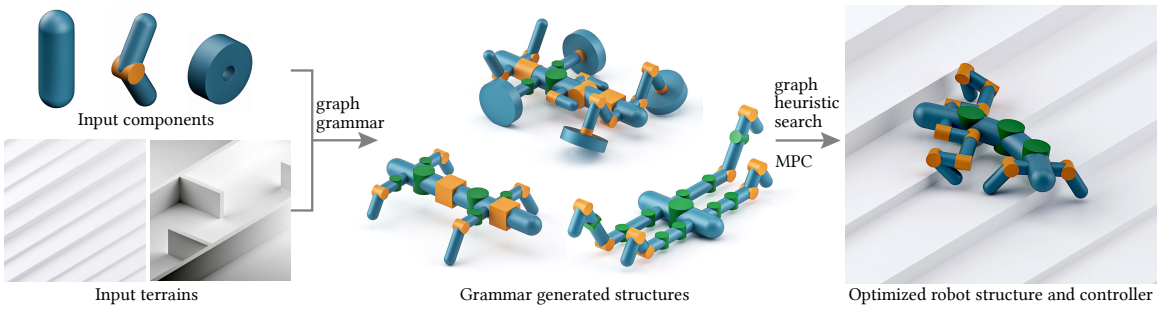


Figure 5-1: **Overview of the computational terrestrial robot design system.** The input to our system is a set of base robot components, such as links, joints, and end structures, and at least one terrain, such as stepped terrain or terrain with wall obstacles. A recursive graph grammar is constructed to efficiently generate hundreds of thousands of robot structures built with the given components. We then use Graph Heuristic Search coupled with model predictive control (MPC) to facilitate exploration of the large design space, and identify high performing examples for a given terrain. Our approach enables co-optimization of both robot structures and controllers.

express a wide range of different terrestrial robots. In Section 5.1.4 we present the optimization process, providing details of the Graph Heuristic Search. The results are presented in Section 5.1.5, showing best-performing designs generated using *Graph Heuristic Search* for six different terrains. We conclude with a discussion of the limitations of our approach and identify avenues for future work.

5.1.2 System Overview

Our whole computational terrestrial robot design system consists of three main components listed below. Figure 5-1 provides a graphical overview.

First, a recursive graph grammar (named *RoboGrammar*) is constructed to represent the terrestrial robot structure design space (Section 5.1.3). We use a graph representation for robot structure and define the set of components and grammar rules which can be used to assemble robots.

The second and the core component of our system is the novel Graph Heuristic Search (GHS) algorithm described in Section 5.1.4. GHS searches over the design space defined by the grammar to efficiently identify optimal robots and controllers. The algorithm exploits a graph neural network-based heuristic function, whose archi-

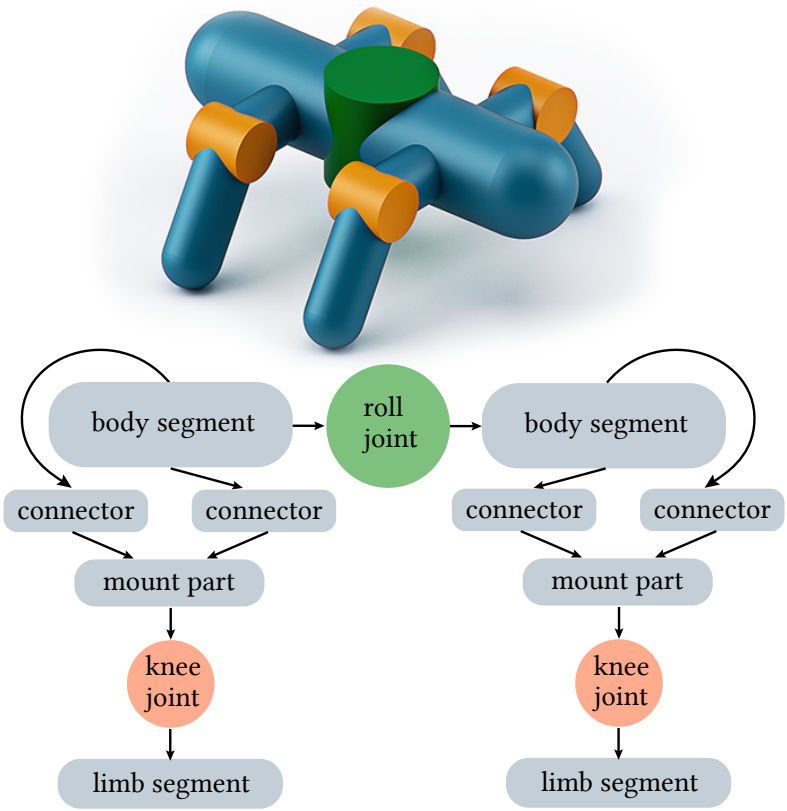


Figure 5-2: **An example of a kinematic tree (top) with the corresponding robot graph (bottom).** To enforce symmetry in leg pairs, after adding nodes for connectors on both sides of the body, both legs of one pair are defined in one branch of the graph.

tecture is analogous to the graph-like structure of rigid robots. The heuristic function is learned as the search progresses using ground-truth data from the MPC-based evaluations.

5.1.3 Graph Grammar for Terrestrial Robot Topology Design

Robot Graph Representation We represent robots in the form of directed acyclic graphs. Each node of the graph represents a physically realizable component. We consider robot structures to consist of body segments and limbs, with optional head and tail. The structures are composed of rigid links and rigid or articulated joints. Each body segment can have at most one pair of legs attached to it.

As our design space is based on arthropods, we impose symmetry on the robot structure. Legs are always added in pairs and each pair has identical leg structure

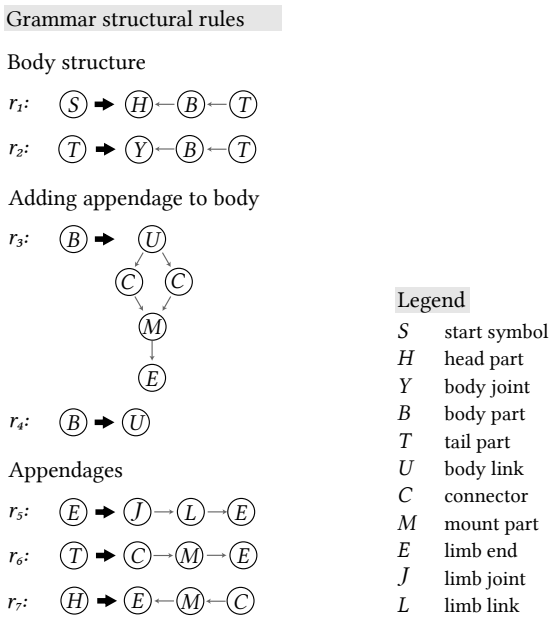


Figure 5-3: **Structural rules of our robot grammar**. Here $S, H, Y, B, T, U, E, J, L$ are non-terminal symbols. Rule r_1 initializes the body structure, while r_2 can be used to extend the body. Note that each body segment U can have at most one pair of limbs attached to it. Rule r_3 enforces symmetry of the limb pairs, and rule r_4 allows body segments without limbs. Rule r_5 serves for extending the limbs, and r_6 and r_7 for adding back and front limbs.

on both sides. We encode symmetry and repetition by representing each pair of legs with a single branch in the robot graph (see Figure 5-2). After deriving a full robot graph, we convert it to a kinematic tree for efficient simulation.

Graph Grammar for Terrestrial Robot We use a context-free graph grammar for terrestrial robot design space. Let us revisit the graph grammar. A graph grammar is a collection of non-terminal symbols, terminal symbols, and expansion/production rules. Non-terminal symbols are temporary graph nodes that help us construct different body and leg parts. Terminal symbols are final graph nodes which represent physical robot components (e.g., links, joints, wheels, etc.). We refer to graphs with only terminal symbols as "complete" robot designs, and all other graphs as "partial" robot designs. In addition, we assign attributes to several terminal symbols. The attributes define the initial state of the robot by determining initial positions and angles between robot parts.

Each production rule detects and replaces a non-terminal symbol in a "partial"

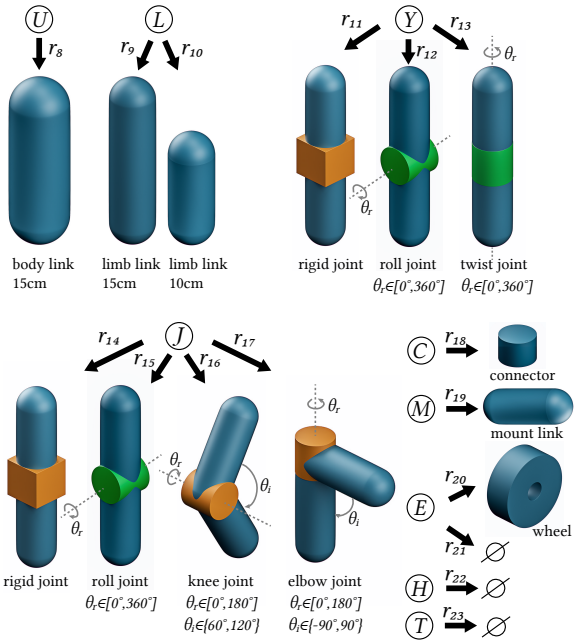


Figure 5-4: **Component-based rules of the robot grammar.** Initial-pose angle θ_i and rotational range angle θ_r are attributes of joints.

robot graph with a sub-graph. We construct two categories of production rules: *structural rules* (Figure 5-3) and *component-based rules*. Structural rules serve to construct a physically realistic topology for the robot and define the number of body and limb segments. Component-based rules replace non-terminal symbols with terminal symbols representing robot components.

An example robot derived from grammar $\mathcal{G}$, along with the sequence of production rules applied, is shown in Figure 5-5. Our recursive grammar is designed in a way that allows for a potentially infinite number of legs and body segments. In order to limit the design space, our implementation uses a recursion counter. The recursion counter counts the total number of derivation steps. We set the maximum as 40 in our experiments. Increasing this parameter would allow for creation of more complex designs, while also exponentially enlarging the design space.

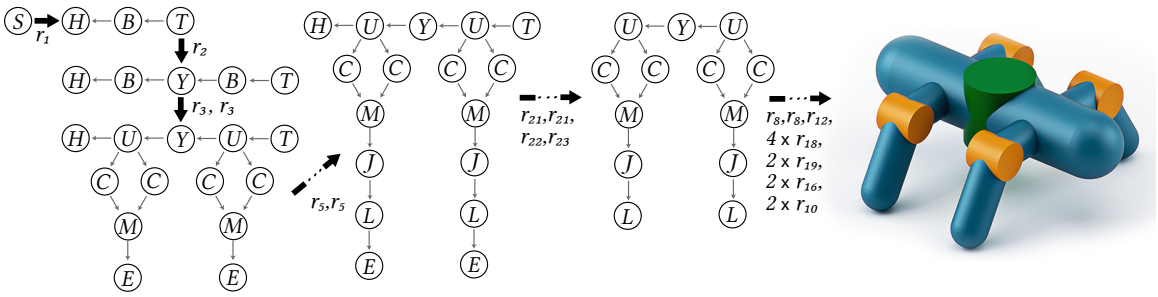


Figure 5-5: **A derivation sequence for a Simple Walker robot generated with our grammar.** Derivation begins with the start symbol S, then creates the body and extends it with rules r_1,r_2 respectively. Legs are added on both body segments with rule r_3 applied twice. Both pairs of legs are extended, adding additional sets of joints and links, with r_5. For the Simple Walker, end structures are not used, hence we remove them with r_{21},r_{22}, and r_{23}. Finally, terminal components are added for each segment of the robot, following the rules from Figure 5-4.

5.1.4 Graph Heuristic Search

While our grammar constrains the search space of all robotic structures to a tractable and meaningful subset, the robot design space spanned by our grammar is still incredibly enormous. Intuitively, we can apply any valid rule from total 23 rules in each of 40 derivation steps, thus the number of possible rule sequences with length 40 is exponentially proportional to the number of derivation steps. To tackle such an enormous search space, we describe our novel search and optimization algorithms named Graph Heuristic Search that efficiently search for high-performing designs and controllers, exploiting the search space this grammar provides.

With our robot grammar, we convert the task of searching the optimal robot topology structure into searching the optimal grammar rule application sequence, that results in a robot graph whose corresponding robot design achieves the optimal performance on the given terrain type. From another perspective, we treat the robot design as a sequential decision making problem, that given a current partial design of the robot after applying k grammar rules, we need to make the decision at $(k+1)$-th step about what is the next optimal rule to apply towards the optimal performing robot.

Starting from this angle, we then draw inspirations from reinforcement learning

approaches to solve our robot design optimization problem and invent a "heuristic" function guided search algorithm. Our heuristic search algorithm is learning-based, using a learned heuristic function to inform and guide the search of the design space, which plays the same role as the value function in a value-based reinforcement learning algorithm. This heuristic takes the form $V(g) : (\mathcal{V}, \mathcal{E}) \rightarrow \mathbb{R}$. The input to the function is a graph representing a partial robot design, where some nodes correspond to non-terminal symbols. The partial design may be expanded into one of many complete designs which have only terminal symbols. The function $V(g)$ aims to outputs the highest achievable performance across all of these complete designs. Our search algorithm is agnostic to the model used for the heuristic, and thus we describe it in general terms. In practice, we take a deep-learning-based approach and use graph neural networks to create our learnable heuristic.

5.1.4.1 Search Algorithm

Our Graph Heuristic Search algorithm works by interleaving a design phase (in which a candidate robot is sampled, guided by our heuristic function), an evaluation phase (in which the candidate robot is evaluated in simulation), and a learning phase (in which the heuristic function is improved based on the simulated data). These three phases are repeated over N episodes, or until they converge on an optimal design. The algorithm is described in Alg. 2.

Design Phase During the design phase, K possible candidate robot designs are generated and one of them is selected for evaluation. Each design is generated by the following procedure. Starting from a partial robot design s_0, composed solely of the initial start symbol ($s_0 := S$), production rules of the grammar are iteratively applied to the partial robot design until it contains only terminal symbols. The selection of production rules is inspired by Q-learning [118] and follows an ϵ-greedy approach. Given a partial robot design s_l after l production rules have been applied, the $l + 1^{\text{th}}$ rule a_{l+1} is selected as follows. With probability ϵ, a random rule is applied from the

Algorithm 2: Graph Heuristic Search

Inputs: Number of iterations N, number of candidate designs M, Adam optimization steps opt_iter and batch size M.
Output: The best design s^*.
Initialize the look up table $\hat{V} \leftarrow \{\}$.
Initialize the graph neural network $V_\theta(s)$ with random parameters θ.
Initialize the best design $s^* \leftarrow$ None, $r^* \leftarrow 0$.
for episode $j \leftarrow 1$ **to** N **do**
  ▷ **Design Phase:** Generate a candidate design
  $\mathcal{P} \leftarrow \{\}$ ▷ Initialize possible design candidates
  ▷ Sample K designs by ϵ-greedy approach
  for $k \leftarrow 1$ **to** K **do**
    $s \leftarrow$ initial design graph
    while s has non-terminals **do**
      With probability ϵ select a random rule a from available rules
      otherwise select $a = \arg\max_a V_\theta(P(s, a))$.
      $s \leftarrow P(s, a)$
    end while
    Add possible candidate s to $\mathcal{P}$.
  end for
  ▷ Choose one to be the candidate
  With probability ϵ select a random sampled design from $\mathcal{P}$ as the candidate design d, otherwise select $d =_{d \in \mathcal{P}} V_\theta(d)$ by heuristic function V_θ.
  ▷ **Evaluation Phase:** Compute the average reward for the design
  Run MPC to evaluate d and get average reward r.
  ▷ Update the best design and $\hat{V}$
  if $r > r^*$ **then**
    $s^* \leftarrow d$
    $r^* \leftarrow r$
  end if
  for Each partial ancestor design d_p of d **do**
    Update $\hat{V}(d_p) \leftarrow \max(\hat{V}(d_p), r)$.
  end for
  ▷ **Learning Phase:** train heuristic value function V_θ
  for $i \leftarrow 1$ **to** opt_iter **do**
    Sample a minibatch S of seen designs (partial or complete) of size M.
    Update $V_\theta(s)$ one step by Adam with the loss:

$$\sum_{s \in S} \|V_\theta(s) - \hat{V}(s)\|^2$$

  end for
end for

set of possible rules. Otherwise, with probability $(1-\epsilon)$, the rule that leads to the design with the highest heuristic score is applied, *i.e.* $a_{l+1} \leftarrow \arg\max_a V_\theta(P(s_l, a))$, where $P(\mathcal{S}, \mathcal{A})$ is a function which applies production rule $\mathcal{A}$ to partial design $\mathcal{S}$. Once a candidate design with only terminal symbols is produced, it is added to the list of possible candidate designs. From the final list of K candidates, a random robot is selected with probability ϵ; with probability $1-\epsilon$, the design with the highest heuristic score is chosen as the candidate to continue to the evaluation phase.

Two ϵ-greedy selection steps are applied during candidate robot generation. This strategy is necessary to ensure that the space of possible robot designs is sufficiently explored; if one begins with a pure greedy strategy, the algorithm quickly converges to a suboptimal design. This is because we are taking a learning-based approach; our heuristic function is inaccurate at the beginning (and not strictly admissible) and improves in accuracy as the algorithm progresses. Until the heuristic function converges to an accurate estimator mapping robot design to performance, it is necessary to generate a diverse collection of robot designs from which to learn from (beginning with $\epsilon = 1$). The first ϵ-greedy exploration rule, within a given design generation, helps guarantee a diverse collection of possible candidate designs (with the variance of that set parameterized by ϵ). Since K applications of this process (with large K and small ϵ) makes it likely that at least one robot resembling the pure greedy-strategy will be generated, the second ϵ-greedy exploration guarantees that the same best candidate is not chosen each time. ϵ is decreased with each episode toward 0 as the heuristic's accuracy increases, according to an exponential decay schedule (as in Q-learning); this is made possible by a fast, accurate learned heuristic function, and shifts the algorithm from exploration to exploitation.

Evaluation Phase After a candidate robot has been decided on, its performance must be evaluated. We simulate the candidate robot with actuation inputs generated by the MPC algorithm (specifically MPPI). It is possible for the same design candidate to be proposed multiple times. Because our MPC algorithm is sampling-based and stochastic, different average rewards $\hat{V}$ may be seen for the same design between

episodes. We consider the $\hat{V}$ of a design to be the best average reward over all evaluations of the design. If this average reward is the best seen so far, the candidate is stored as the current best design, along with $\hat{V}$. Regardless, the candidate robot design and its corresponding $\hat{V}$ are stored (or updated) in a lookup table, and the $\hat{v}$ label of all of that design's partial design ancestors are updated to be the maximum of their current value and the candidate robot's average reward. This is important for the upcoming learning phase, which must learn a heuristic function that is valid for both complete and partial designs.

The number of candidate designs evaluated in each iteration is an algorithm design trade-off. Evaluating more candidates will collect more data, helping to train a more accurate prediction function. It will also significantly increase the computation time, however, since evaluation is the time bottleneck in our algorithm. We therefore choose to evaluate only one design per iteration.

Learning Phase The heuristic is trained using the data stored in the lookup table. For `opt_iter` epochs, minibatches of $(s_i, \hat{V}_i)$ pairs are sampled, and the loss $L = \frac{1}{2}\|V_\theta(s_i) - \hat{V}\|_2^2$ is minimized using Adam [119].

5.1.4.2 Heuristic Function Model

To implement our heuristic function, we choose to leverage the expressive nature of graph neural networks. Graph neural networks (GNNs) are neural network architectures which aim to extend the benefits of deep learning to a graphical setting. Unlike other neural network types such as CNNs, which operate on images and data with fixed grid-like topologies, graph neural networks aim to be flexible and operate on structures with arbitrary topologies. The input to a GNN consists of a graph topology (*e.g.* an adjacency matrix), and values associated with nodes (*e.g.* feature vectors). While many GNN models have been proposed in recent years, our architecture is based on the differentiable-pooling model. This model was designed for inference tasks involving graphs with a *hierarchical nature*, by iteratively reducing the graph to

a "lower resolution" graph in a manner similar to hierarchical clustering. Please see [120] for more details. This model is well-suited for our scenario, where each robot is itself created through a hierarchical substitution of grammar rules.

The differentiable-pooling GNN extends the GraphSage framework from [121], which in turn is based on graph convolutional networks (GCN) [122]. Analogously to CNNs, GCNs apply a generalized convolution operator that operates on graphs rather than grids. We adopt a similar model as [120]. In this model, two "mean" GraphSage+BatchNormalization layers are applied, followed by the hierarchical clustering DiffPool layer, followed by three layers of graph convolutions. This process is repeated one additional time, followed by a final GraphSage layer, a mean pooling layer, and a final ReLU. The output is a positive real-valued scalar representing predicted robot performance. Each DiffPool layer reduces the node set's cardinality by 75%.

An important property of this GNN model is that it is isomorphism-invariant, meaning any two isomorphic graphs will have the same value and gradient without the need for explicit transposition tables. This greatly simplifies the bookkeeping in Graph Heuristic Search.

We convert robot design graphs into inputs for the GNN model as follows. The graph is first converted to a kinematic tree, so that each link has a unique joint connecting it to its parent link. Each link and its parent joint is considered a node in this new graph. Note that this representation differs from the one described in Section 5.1.3. Next, an m-dimensional feature vector is extracted from each node. If the link associated with the node is a terminal symbol, the feature vector encodes the link's initial position, orientation, and geometric description. The parent joint's rotation and servo parameters are included similarly, if present. If either the link or joint are non-terminal symbols, the feature vector one-hot encodes the non-terminal type.

5.1.5 Experiments

5.1.5.1 Implementation Details

Hyperparameters We run all experiments with the same hyperparameters unless otherwise specified. The exploration parameter ϵ in GHS follows an exponential decay schedule: $\epsilon(i) = \epsilon_1 + (\epsilon_0 - \epsilon_1)\exp\left(-\frac{i/N}{\epsilon_{decay}}\right)$, where i is the current iteration, N is the total number of iterations, $\epsilon_0 = 1$, $\epsilon_1 = 0.1$ and $\epsilon_{decay} = 0.3$. We run GHS for 2,000 iterations ($N = 2000$). In the design phase of each iteration, GHS uses the two ϵ-greedy steps to select one design to be tested by MPC from 16 sampled possible candidate robots. In the learning phase, the Adam optimizer runs for 25 steps with batch size 32 and learning rate 1×10^{-4}. A summary of hyperparameters is provided in Table 5.1.

Experiment Setup & Computational Time We implemented our Graph Heuristic Search algorithm in Python, and the simulation and MPC in C++. Experiments were run on VM instances with either 32 or 64 Intel Cascade Lake vCPUs on Google Cloud Platform. Each iteration of GHS spends less than 1 second per possible candidate robot in the design phase, 40-60 seconds in the MPC evaluation phase, and 6-8 seconds in the learning phase. Since each possible candidate robot is sampled independently, the design phase can be fully parallelized for further speedup. The time bottleneck of the MPC evaluation phase shows the necessity of our Graph Heuristic Search algorithm, which is able to find the best-performing robots while evaluating a significantly fewer number of robot designs.

5.1.5.2 Task Specification

Here, we describe the remaining (user-specified) components needed to define a co-optimization problem with RoboGrammar.

Table 5.1: **Graph Heuristic Search hyperparameter values**

Hyperparameter	Value
Number of iterations (N)	2000
Initial ϵ (ϵ_0)	1.0
Final ϵ (ϵ_1)	0.1
ϵ exponential decaying factor (ϵ_{decay})	0.3
Number of possible candidate robots (K)	16
Optimization steps (opt_iter)	25
Optimization batch size (M)	32
Adam learning rate	1×10^{-4}

Reward Function A single reward function (Equation 5.1) is used to evaluate designs on every terrain, and is computed at every time step of the simulation. For the purposes of design optimization we use the average reward across all time steps.

$$r(t) = \vec{w_x} \cdot \vec{d_x}(t) + \vec{w_y} \cdot \vec{d_y}(t) + \vec{w_v} \cdot \vec{v}(t) \tag{5.1}$$

We consider the base link of the robot, or the forwardmost wide body segment, to be representative of the robot's motion. $\vec{d_x}$ and $\vec{d_y}$ are unit vectors pointing forward and upward in the base link's reference frame, respectively, and $\vec{v}$ is the base link's velocity. All quantities are expressed in world coordinates.

$\vec{w_x} = [-2, 0, 0]^T$, $\vec{w_y} = [0, 2, 0]^T$, and $\vec{w_v} = [2, 0, 0]^T$ are weighting vectors which set the relative importance of each term. They also scale the reward function's magnitude to the range expected by the design search algorithm.

The first two terms encourage maintaining the initial orientation, and the last term rewards forward progress. The robot starts with its local x-axis pointing in the negative x direction in world coordinates.

Terrains Terrains, in conjunction with the reward function, define tasks to optimize robot structures for. Each terrain is intended to result in a different set of optimal designs.

1) *Flat terrain*: A featureless surface with a friction coefficient of 0.9, the flat terrain accommodates the greatest variety of locomotion styles.

2) *Frozen lake terrain*: A flat surface with a low friction coefficient of 0.05, the frozen lake terrain encourages designs which maximize traction or use the low friction to their advantage.

3) *Ridged terrain*: Ridges or hurdles spaced an average of one meter apart span the entire width of the ridged terrain, requiring designs to jump or crawl in order to make progress.

4) *Wall terrain*: Walls which are too high to traverse directly are placed in a slalom-like arrangement. Designs must move around the walls, requiring them to change their direction of motion rapidly.

5) *Gap terrain*: A series of platforms separated by gaps require designs to tread carefully. As the gaps become progressively wider, designs with the ability to take larger steps are favored.

6) *Upward stepped terrain*: A series of steps resembling a flight of stairs test the ability of designs to climb. The steps are of varying height, producing different gait variations over time.

5.1.5.3 Results

Here we demonstrate our co-design approach on a collection of different problems, examining how terrain affects which designs and controllers are optimal. Furthermore, we quantitatively analyze the efficiency of our Graph Heuristic Search algorithm compared to the baselines.

Terrain Driven Optimization As an end-to-end demonstration of our co-design approach, we run Graph Heuristic Search on several different terrains. Each search run consists of 2,000 iterations. A selection of best-performing designs is shown in Figure 5-6.

Optimal designs for the ridged terrain are characterized by long limbs which are able to swing upwards and clear obstacles. Although the set of optimal designs

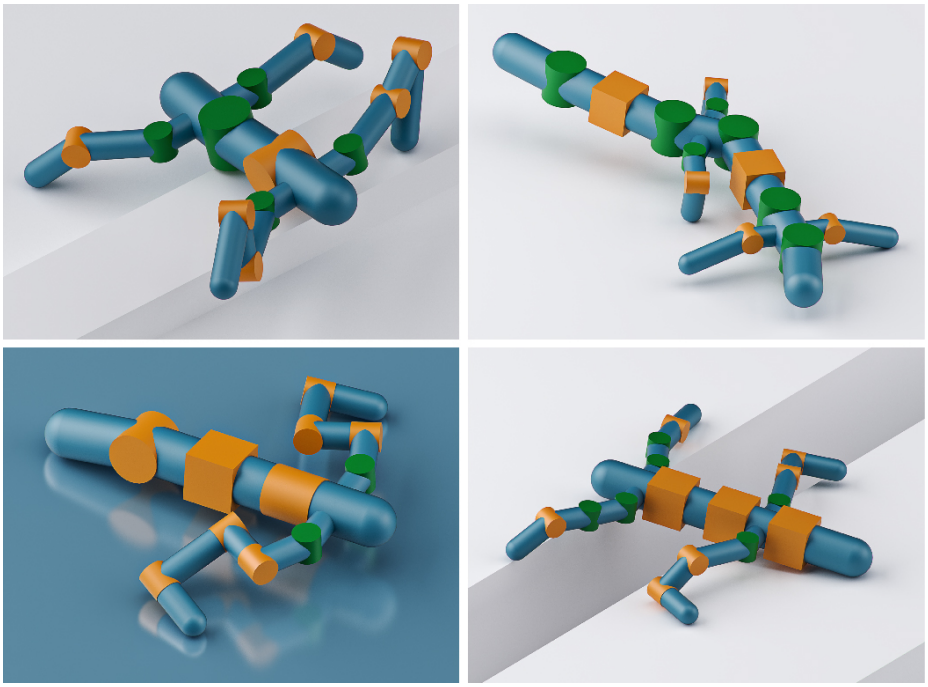


Figure 5-6: **Selection of best-performing designs optimized with Graph Heuristic Search for ridged, flat, frozen lake, and gapped terrain respectively.**

consists mainly of quadrupeds, a few tripedal designs emerge. These designs use their body as a third point of contact with the ground.

The flat terrain, despite having no obstacles, still produces specialized designs. One successful strategy is to place short limbs spaced far apart on the body, giving them a full range of motion. Although short limbs would be unable to clear obstacles on most of the other terrains, their low inertia enables quick movement.

The frozen lake is superficially similar to the flat terrain, but its low friction coefficient requires a different strategy. Successful designs can both overcome the low friction and use it to their advantage. The bottom-left design in Figure 5-6 serves as an example. Highly articulated yet compact arms maintain contact with the ground during the stance phase, while the rear body segment slides freely.

The gap terrain tends to produce designs with long limbs, much like the ridged terrain. However, designs for gap terrain tend to have limbs that are optimized for forward reaching instead of climbing. Green joints, which enable limbs to move horizontally, are more prevalent than orange joints, which enable vertical motion.

Efficiency of Graph Heuristic Search To show the efficiency of the proposed Graph Heuristic Search algorithm, we compare our algorithm with two baselines on four different terrain tasks (flat terrain, frozen lake terrain, ridged terrain, and wall terrain). Specifically, the first baseline is an adapted Monte Carlo tree search (MCTS) algorithm. The second baseline algorithm is a random search algorithm, where in each iteration, one candidate design is selected by applying random rules. Due to the stochasticity of the search algorithms, we run each algorithm on each task at least three times with different random seeds. Note that we run our Graph Heuristic Search algorithm for only 2,000 iterations, as opposed to 5,000 iterations for each baseline algorithm. The results in Figure 5-7 show that Graph Heuristic Search consistently finds better designs (achieves greater reward) than the baseline algorithms in much fewer iterations. Due to the expensive MPC-based evaluation of designs and the combinatorial nature of our design space, this sample efficiency is key to finding high-performing designs in a reasonable amount of time. The efficiency comes from the ability of the learned heuristic function to generalize knowledge from explored designs to predict the performance of untested designs and effectively prune the search space.

Running 2,000 iterations of Graph Heuristic Search requires approximately 31 hours on a 32-core Google Cloud machine (instance type `n2-highcpu-32`). This computational intensity is comparable to other state-of-the-art robot design methods. For example, Neural Graph Evolution [11] is evaluated using 12 hours on a 64-core machine. Note that we simulate designs with greater numbers of joints on more complex terrains, resulting in more expensive evaluations. Evaluations account for approximately 20 hours out of our total run time.

Convergence of Graph Heuristic Search To demonstrate that Graph Heuristic Search is agnostic to the specific grammar described (the *standard* grammar), we optimize robots for flat terrain using two modified grammars. The *simple* grammar removes the rules for long limb links and elbow joints (r_9 and r_{17} in Figure 5-4 respectively), whose functionality is provided by other rules. The *asymmetric* grammar

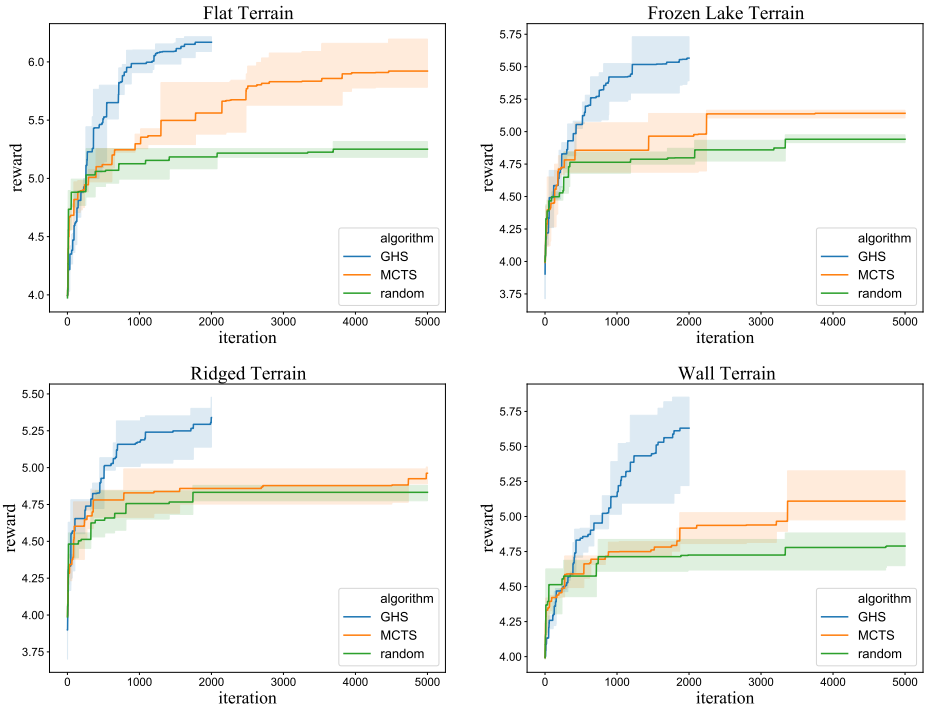


Figure 5-7: **Training progress comparison with baselines.** Cumulative maximum reward versus iteration for Graph Heuristic Search, Monte Carlo tree search, and random search on four different terrains. Each solid line is the mean of three different seeds, with the error band representing the range. Graph Heuristic Search consistently outperforms the baselines.

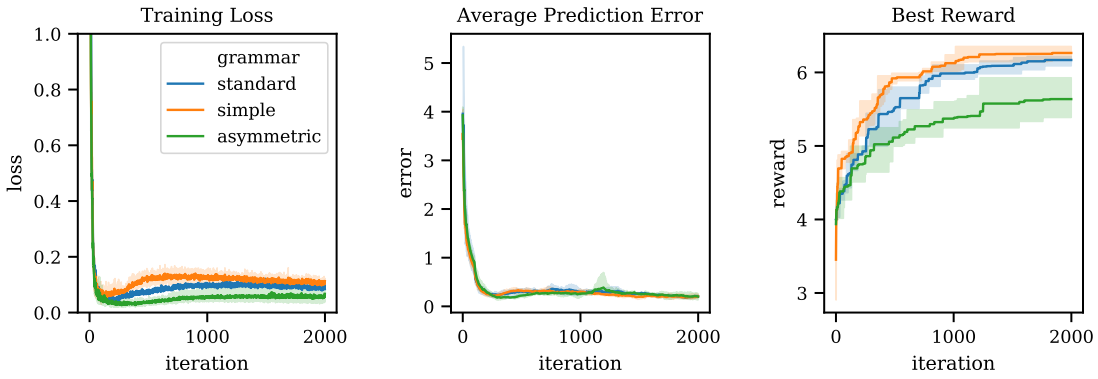


Figure 5-8: **Training loss, prediction error, and cumulative maximum reward versus iteration for Graph Heuristic Search on multiple grammars.** Robots are optimized for flat terrain. Prediction error is the absolute difference between predicted reward and evaluated reward, and is averaged over 100 iterations. Each solid line is the mean of at least three different seeds, with the error band representing the range. Graph Heuristic Search consistently converges in all three criteria.

increases complexity by allowing opposite limbs to develop independently. Figure 5-8 shows that Graph Heuristic Search consistently converges in training loss, prediction error, and maximum cumulative reward. More complex grammars require a greater number of iterations to achieve the same reward.

5.1.6 Summary

Intelligent and efficient generative robot design methods will drive the future of design processes for robotics. In this work we present a novel approach which leverages a learning-based method to co-optimize the control and discrete shape topology for terrain-driven robot. We introduce a Graph Heuristic Search algorithm to search the combinatorial search space, and couple it with MPC for control evaluation. Unlike many alternative approaches to generative robot design, this allows us to structure and limit the design space by applying a graph grammar, whilst allowing creative solutions to emerge. Importantly, the emergent designs are observably physically fabricable and there is significant potential for the designs to be translated to real-world scenarios and environments.

5.2 Co-Optimizing Robot Control and Continuous Shape Morphology: An End-to-End Differentiable Framework

5.2.1 Motivation

We have shown in the last section how we leverage a graph grammar design space and a learning-based algorithm to efficiently co-optimize the control and discrete shape topology for robots. In this section, we focus on how to efficiently co-optimize the robotic control and its continuous shape morphology with a predefined robot topology. We demonstrate our approach on the manipulator co-design problems as described in Section 3.2.

The design, control, and construction of manipulators is the cornerstone of robotics. Today this process is manual and time-consuming as concurrent design of many different components is required. For example, hardware components and control algorithms are typically constructed sequentially making the integration of different modules difficult which necessitates many design iterations. Ensuring that the designed manipulator meets the desired specifications is challenging since there is a complex interplay between the robot design, manufacturing constraints, and the control algorithm.

Due to the long iteration cycle, in practice, roboticists either (i) explore a rich design space, but make use of simple control algorithms [123, 124] or (ii) develop complex algorithms to control existing robots [125, 126, 127]. The end result is a sub-optimal system for the given task. Co-optimizing both the design and the control scheme can significantly improve the performance of today's robotic systems. One significant challenge is in defining a representation of the robot design that is amenable to optimization. We have demonstrated in Section 3.2 our deformation-based robot morphology representation which defines an expressive morphology space while being computationally inexpensive for inference, flexible (*i.e.*, user can easily

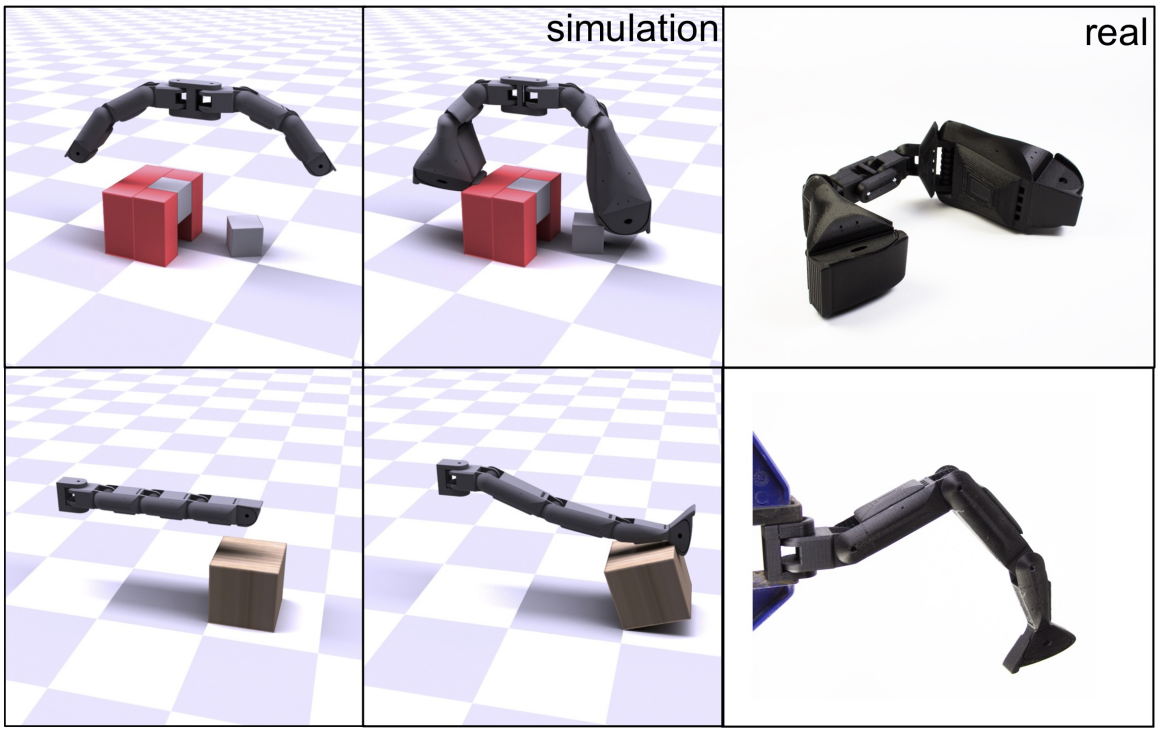


Figure 5-9: **Manipulator designs before and after optimization.** *Left column*: only optimizing the control algorithm using a nominal robot design fails to complete the task; *Middle*: co-optimization of morphology and control results in success; *Right:* pictures of 3D-printed manipulators. Our method outputs designs that are easy to print and assemble.

control the degrees of freedom describing the shape by changing the number of cage handles), and differentiable. We utilize the same shape representation here. Another challenge in co-optimization is the substantial increase in the number of parameters to be optimized. We tackle this challenge by presenting an end-to-end differentiable co-optimization framework for robot design so that we can leverage the analytical gradients to optimize numbers of parameters efficiently.

Specifically, we exploit the differentiability of the proposed deformation-based parameterization in Section 3.2, and further develop a differentiable articulated rigid body simulator for *contact-rich tasks*. Our simulator is based off the differentiable simulator introduced in Section 4.1. Additionally, to be able to optimize for the shape of the robots, we further derive the analytical gradients for a full spectrum of simulation parameters shown in Table 5.2, including the positions of contact points.

The combination of the proposed deformation-based parameterization and the dif-

ferentiable simulator allows us to build an end-to-end differentiable framework (Figure 5-10) for co-optimizing robot morphology and control for contact-rich manipulation tasks using analytical gradients. We test our framework on multiple manipulation problems, some of which are shown in Figure 5-9. The experiments show that due to the key feature of end-to-end differentiability, the proposed method outperforms several state-of-the-art gradient-free approaches and model-free reinforcement learning methods at jointly optimizing the *control scheme* and the *robot morphology*.

5.2.2 Method

We now describe our end-to-end differentiable framework for contact-aware robot co-design. The first part of our framework is the deformation-based design space detailed in Section 3.2 (shown in top-left block of Figure 5-10). To complete the whole framework, in Section 5.2.2.1, we describe our differentiable articulated rigid body dynamics simulation. As shown in the right block of Figure 5-10, the simulation takes the deformed meshes and a control sequence as input, executes the forward steps, and computes the objective loss $\mathcal{L}$. By combining the two key techniques above, in Section 5.2.2.2, we describe our end-to-end framework for robot design. Since each step is differentiable, the overall framework is differentiable, allowing us to use a gradient-based optimization method to search for the design parameters and the controls.

5.2.2.1 Differentiable Articulated Rigid Body Simulation

From the deformation-based parameterization (Section 3.2 and top-left block in Figure 5-10), we obtain the mesh vertices ψ_M. In order to simulate the constructed morphology, we further convert ψ_M into the simulation parameters ψ_p through an analytical function $\mathcal{P}$. As shown in Table 5.2, the simulation parameters ψ_p include both kinematics- and dynamics-related parameters. Specifically, the kinematic parameters are the relative transformations of the joints with respect to their parent joints, E_j, and the relative transformations of the bodies with respect to their parent

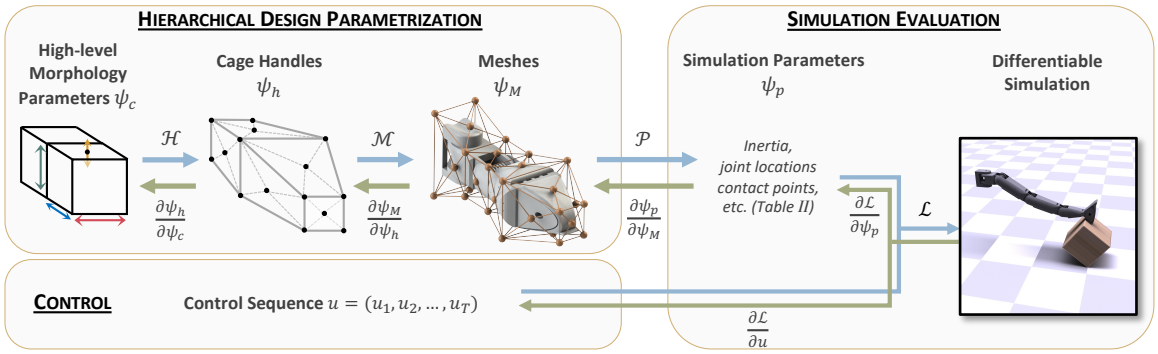


Figure 5-10: **End-to-end differentiable framework for morphology and control co-optimization.** Blue arrows labeled as $\mathcal{H}, \mathcal{M}, \mathcal{P}$, and $\mathcal{L}$ are hierarchical functions that evaluate the loss function given the high-level morphology parameters, ψ_c and controls, u. The corresponding green arrows are the derivatives.

joint, E_b; and the dynamic parameters are the generalized inertia, I, contact point positions with respect to the bodies, C_b, and surface area for each contact point, a. For the generalized inertia, we use cuboids for ease of differentiability, but it is also possible to use mesh-based inertia. (Note, however, that the 3D printed parts may not necessarily match the mesh-based inertia, depending on the in-fill.) In order to acquire the contact points C_b in the deformed mesh, we presample a uniformly distributed set of contact points on the surface of each mesh in the rest configuration. We then track the positions of these presampled contact points through the same cage-based deformation as the mesh vertices. Thus, the deformation-based parameterization provides us with differentiability not only for the mesh but also for the contact point positions. The approximate contact point area, a, is used to scale the magnitude of the frictional contact forces. To compute this parameter, we use the change in the total surface area of the cage before and after deformation.

In addition to the previous differentiable simulator mentioned in Section 4.1 which only computes the gradients of the simulation only with respect to control parameters $\partial \mathcal{L}/\partial u$, our simulator also provides the analytical gradients $\partial \mathcal{L}/\partial \psi_p$ for a full spectrum of simulation parameters described above. Such extension is non-trivial and is essential for allowing gradient-based morphology optimization.

Formally speaking, the simulation parameters ψ_p and the control sequence u, are the input to the differentiable articulated rigid body simulator. Our simulator uses

Table 5.2: **List of simulation parameters** ψ_p

Type	Notation	Parameter Description	Dimension
Kinematics	E_j	Joint transformation	$\mathrm{SE}(3) \times n_b$
	E_b	Body transformation	$\mathrm{SE}(3) \times n_b$
Dynamics	I	Generalized inertia	n_{DOF}
	C_b	Contact points on body	$3 \times n_c$
	a	Contact area	n_c

reduced coordinates as before (Section 4.1). The dynamics equations are implicitly integrated in time. We adopted two implicit time integration schemes, BDF1, and BDF2 (with SDIRK2 for the initial step). Since our simulation dynamics is now conditioned on the simulation parameters, the simulation dynamics in Eq. 4.3 and Eq. 4.4 need to be modified accordingly. For brevity, the simulation dynamics with BDF1 integration scheme is now:

$$\left.\begin{aligned} \boldsymbol{q}_t &= \boldsymbol{q}_{t-1} + h\dot{\boldsymbol{q}}_t \\ \dot{\boldsymbol{q}}_t &= \dot{\boldsymbol{q}}_{t-1} + h\ddot{\boldsymbol{q}}_t(\boldsymbol{q}_t, \dot{\boldsymbol{q}}_t, \boldsymbol{u}_t, \boldsymbol{\psi_p}) \end{aligned}\right\} \Rightarrow \underbrace{\boldsymbol{q}_t - \boldsymbol{q}_{t-1} - h\dot{\boldsymbol{q}}_{t-1} - h^2\ddot{\boldsymbol{q}}_t(\boldsymbol{q}_t, \dot{\boldsymbol{q}}_t, \boldsymbol{u}_t, \boldsymbol{\psi_p})}_{g(\boldsymbol{q}_{t-1}, \dot{\boldsymbol{q}}_{t-1}, \boldsymbol{u}_t, \boldsymbol{q}_t, \boldsymbol{\psi_p})} = 0 \tag{5.2}$$

with

$$\ddot{\boldsymbol{q}}_t(\boldsymbol{q}_t, \dot{\boldsymbol{q}}_t, \boldsymbol{u_t}, \boldsymbol{\psi_p}) = \mathsf{M}_r^{-1}(\boldsymbol{q}_t, \boldsymbol{\psi_p}) \Big[\mathsf{f}_r(\boldsymbol{q}_t, \dot{\boldsymbol{q}}_t, \boldsymbol{\psi_p}) + \mathsf{J}^\top(\boldsymbol{q}_t, \boldsymbol{\psi_p}) \mathsf{f}_m(\boldsymbol{q}_t, \dot{\boldsymbol{q}}_t, \boldsymbol{\psi_p}) + \mathsf{f}_{QVV}(\boldsymbol{q}_t, \dot{\boldsymbol{q}}_t, \boldsymbol{\psi_p}) + \boldsymbol{u}_t \Big], \tag{5.3}$$

We analytically derive all the derivatives required by these implicit time integration schemes, and we solve the resulting non-linear equations using Newton's Method with line search.

For the optimization, we consider open-loop control sequence instead of control policy as our control representation. To compute the gradients with respect to the control variables as well as the simulation parameters, we formulate the co-design

optimization problem as follows:

$$\underset{\boldsymbol{u},\boldsymbol{\psi_p}}{\text{minimize}} \;\; \mathcal{L} = \sum_{t=1}^{H} \mathcal{L}_t\big(\boldsymbol{u}_t, \boldsymbol{q}_t, \mathbf{v}_t(\boldsymbol{q}_t, \boldsymbol{\psi_p})\big) \qquad \text{(Step-wise Objective)} \tag{5.4a}$$

$$\text{s.t.} \;\; g(\boldsymbol{q}_{t-1}, \dot{\boldsymbol{q}}_{t-1}, \boldsymbol{u}_t, \boldsymbol{q}_t, \boldsymbol{\psi_p}) = 0 \qquad \text{(Equations of Motion)} \tag{5.4b}$$

We use the same adjoint method as described in Section 4.1.4 to compute the simulation derivatives, $\frac{\partial \mathcal{L}}{\partial \psi_p}$ and $\frac{\partial \mathcal{L}}{\partial u}$.

5.2.2.2 End-to-End Differentiable Co-Design Framework

By combining the proposed deformation-based parameterization and the differentiable simulator, we build an end-to-end differentiable framework for robot co-design as shown in Figure 5-10.

Mathematically speaking, our co-design framework starts with a three-layer morphology parameterization $\mathcal{F} = \mathcal{P} \circ \mathcal{M} \circ \mathcal{H} : \mathbb{R}^m \to \mathbb{R}^{|H|\times 3} \to \mathbb{R}^{|V|\times 3} \to \mathbb{R}^p$, where m is the number of high-level morphology parameters, $|H|$ is the total number of cage handles, $|V|$ is the total number of mesh vertices, and p is the number of low-level kinematic and dynamic parameters in the simulation. This hierachical parameterization converts high-level morphology parameters ψ_c into low-level simulation parameters ψ_p, going through three analytically differentiable steps including the morphology parameterizations $\mathcal{H}$ and $\mathcal{M}$ in Section 3.2.2, and the simulation parameter computation $\mathcal{P}$ in Section 5.2.2.1:

$$\psi_p = \mathcal{P}(\psi_M), \;\; \psi_M = \mathcal{M}(\psi_h), \;\; \psi_h = \mathcal{H}(\psi_c). \tag{5.5}$$

The differentiability of each step allows us to efficiently compute the derivatives from the simulation parameters ψ_p all the way to the high-level morphology parameters ψ_c through the chain rule:

$$\frac{\partial \psi_p}{\partial \psi_c} = \frac{\partial \psi_p}{\partial \psi_M} \frac{\partial \psi_M}{\partial \psi_h} \frac{\partial \psi_h}{\partial \psi_c}. \tag{5.6}$$

The framework then proceeds with the simulation described in Section 5.2.2.1 with the simulation parameters ψ_p and a control sequence u as input, and computes the task-specific objective loss $\mathcal{L}$. As our simulator is differentiable with respect to both the kinematic/dynamic parameters and control variables, we are able to compute the analytical derivatives $\frac{\partial \mathcal{L}}{\partial \psi_c} = \frac{\partial \mathcal{L}}{\partial \psi_p}\frac{\partial \psi_p}{\partial \psi_c}$ and $\frac{\partial \mathcal{L}}{\partial u}$ for the full framework efficiently. With the analytical derivatives calculated, we can use any gradient-based optimizer (*e.g.*, L-BFGS-B) to co-optimize the control and continuous morphology parameters.

5.2.3 Experiments

5.2.3.1 Implementation

We implemented our differentiable rigid body simulation in C++ and the deformation-based design parameterization in Python. The two components of the code are connected through Python bindings. The control sequence input to the simulation consists of the torques applied to the joints at each simulation time step. In all the tasks shown in below, we use L-BFGS-B [128] for co-optimization with our analytical derivatives, $\partial \mathcal{L}/\partial \psi_c$ and $\partial \mathcal{L}/\partial u$.

Table 5.3: **List of hyper-parameters for each example.**

Task	Δt_s	n_t	n_{ctrl}	$\lvert u\rvert$	$\lvert \psi_c\rvert$	$\lvert \psi_p\rvert$
Finger Reach	0.005	600	20	120	9	376
Flip Box	0.005	150	5	180	9	1478
Rotate Rubik's Cube	0.005	200	5	240	9	1478
Assemble	0.001	500	5	800	17	1226
Free-form Gripper	0.005	400	1	-	396	9228

5.2.3.2 Morphology and Control Co-Optimization

Tasks In order to test the performance of our differentiable contact-aware co-optimization framework, we designed four manipulation tasks as shown in Figure 5-12, consisting of three single-finger tasks and one two-finger task:

1) *Finger Reach*: In this task, the base of the finger is mounted on the wall, and the finger is required to reach four scattered target points in the space sequentially. The initial design of the finger is not long enough to reach two of them. Thus it requires the algorithm to optimize to finger to be longer in order to reach all four points. The cost $\mathcal{L}$ of this task is computed by:

$$\mathcal{L} = \sum_{t=1}^{T} c_u \|u_t\|^2 + c_p \|p_t - \hat{p}_t\| \tag{5.7}$$
$$\text{with } c_u = 0.1, c_p = 10$$

where $u_t \in [-1, 1]$ is the action at time t, p_t is the finger tip position at time t, and $\hat{p}_t$ is the target points at time t.

2) *Flip Box*: This task requires the finger to flip a heavy box by 90° and be as energy-efficient as possible. The bottom front edge of the heavy box is attached to the ground with a revolute joint. This task is more difficult than the previous one since the finger needs to interact with the box, which involves a rich amount of contacts and requires leverage of the contact force to flip the heavy box. The cost of this task is computed by:

$$\mathcal{L} = \sum_{t=1}^{T} c_u \|u_t\|^2 + c_{touch} \|p_t - p_{touch}\|^2 + c_{flip} \|\theta_t - \frac{\pi}{2}\|^2 \tag{5.8}$$
$$\text{with } c_u = 5, c_{touch} = \begin{cases} 1 & t < T/2 \\ 0 & t \geq T/2 \end{cases}, c_{flip} = 50$$

where $u_t \in [-1, 1]$ is the action at time t, p_t is the finger tip position at time t, p_{tourch} is a point on the back surface of the box, and θ_t is the rotation angle of the box at time t. The second term is designed to encourage the manipulator to touch the box and provide some simple heuristics of solving the task.

3) *Rotate Rubik's Cube*: A finger is required to rotate the top layer of a Rubik's cube by 90°. The bottom of the Rubik's cube is fixed on the ground. In this task, there is no clear heuristics in the objective function to guide the finger to

touch a specific place on the cube, so the finger needs to be optimized to find the correct strategy. The cost of this task is defined by:

$$\mathcal{L} = \sum_{t=1}^{T} \left(c_u \|u_t\|^2 + c_{touch} \|p_t - p_{cube}\|^2 \right) + c_{rotate} \|\theta_T - \frac{\pi}{2}\|^2 \tag{5.9}$$
$$\text{with } c_u = 5, c_{touch} = 0.1, c_{rotate} = 1000$$

where u_t and p_t are same as previous tasks, p_{cube} is the center of the Rubik's cube, and θ_T is the rotation angle of the top layer of the cube at the last time step.

4) *Assemble*: Two fingers need to collaborate to push and insert a small cube into its movable mount. The cube and the hole on the mount have similar sizes, making the task much more challenging and requiring high-accuracy manipulation. Moreover, the movable mount needs to stay as close as possible to the original position to mimic a restricted working platform environment. The two fingers are mounted on a manipulator base that is allowed to move in the horizontal plane. The cost of this task is computed as:

$$\begin{aligned} \mathcal{L} = \sum_{t=1}^{T} & c_{mount} \|p_t^M - p_0^M\|^2 + c_{touch} (\|p_t^{left} - p_t^M\|^2 + \|p_t^{right} - p_t^{box}\|^2) \\ & + c_p \|p_t^{box} - p_{hole}\|^2 + c_{rotation} \|\theta_t^M - \theta_t^{box}\|^2 \end{aligned} \tag{5.10}$$
$$\text{with } c_{mount} = 15, c_{touch} = 1, c_p = 5, c_{rotation} = 50$$

where p_t^M is the position of the movable mount at time t, p_t^{left} and p_t^{right} are the finger tip positions of left finger and right finger at time t, p_t^{box} is the position of the small box at time t, θ_t^M and θ_t^{box} are the rotation angle of the mount and the box. The first term is used to penalize moving the mount too far away from the original place, the second term is designed to encourage the fingers to touch on the objects (but not indicate any specific position on the object), and the third term and the last term together is to measure how well the box is inserted into the mount.

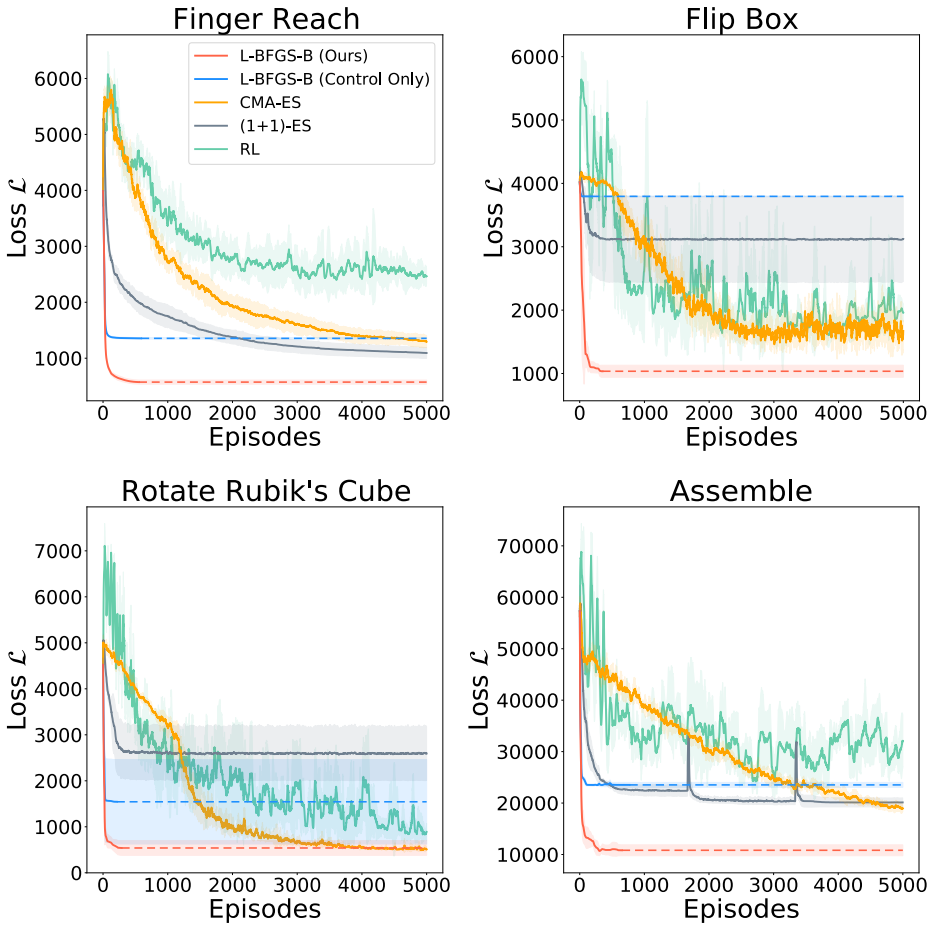


Figure 5-11: **Optimization curves comparison**. We run all the methods on all tasks 5 times with different random seeds. Mean and standard deviation in the loss objective are reported. The horizontal axis of each plot is the number of simulation episodes, and vertical axis is the objective loss value. L-BFGS-B optimization can terminate early once it satisfies the termination criterion. For better visualization, we extend the actual learning curves that use L-BFGS-B horizontally using dotted lines. We also smooth out the curves with a window size of 10.

The hyper-parameters of each task are listed in Table 5.3, where Δt_s is the simulation time steps size, n_t is the total number of simulation steps of the task, $|u|$ and $|\psi_c|$ are the total numbers of control and morphology variables in optimization respectively, and $|\psi_p|$ is the number of simulation parameters. We optimize for the control signals not every simulation step but every n_{ctrl} steps, giving us $|u| = (n_t/n_{ctrl}) \cdot |u_i|$ where $|u_i|$ is the number of control degrees of freedom of the robot.

Baselines We adopted the following three baseline algorithms for comparison.

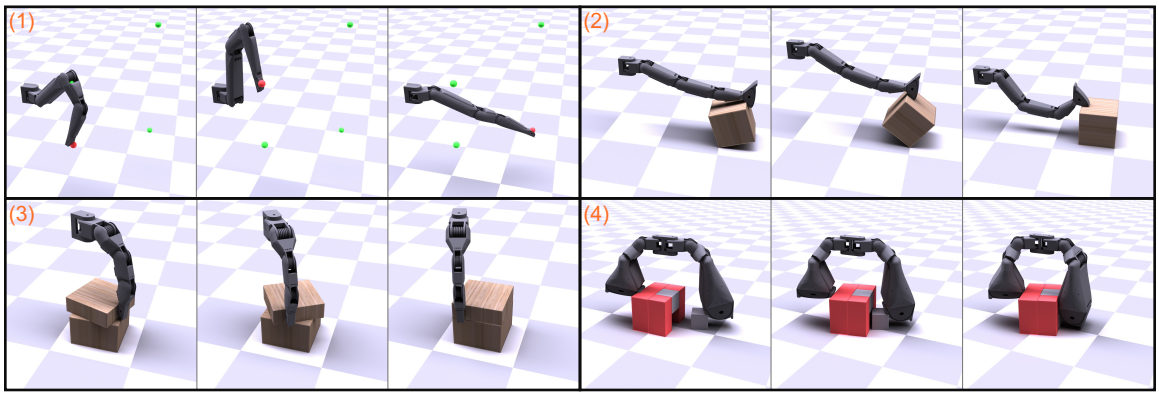


Figure 5-12: **Optimized designs and controls for four manipulator tasks**. (1) *Finger Reach*. (2) *Flip Box*. (3) *Rotate Rubik's Cube*. (4) *Assemble*. More visual results are provided in the supplementary video.

1) *ES*: Evolutionary strategy is widely used to search for optimal design and control parameters for robots [15, 16]. We tried various ES algorithms in the open-sourced Nevergrad library [129] and found that the $(1+1)$-*ES* [130, 131] algorithm and *CMA-ES* [132] work best on the proposed tasks.

2) *RL*: Luck et al. [20] is one of the state-of-the-art morphology and control co-optimization approaches using sample-efficient reinforcement learning (soft actor critic, SAC) algorithm and particle swarm optimization. We used their released implementation as a baseline.

3) *Control Only*: In this algorithm, we freeze the morphology parameters and only optimize the control sequence with L-BFGS-B.

Experiment Setup We use the same morphology parameterization for baselines and our method. The control parameter for the *RL* baseline is a neural network controller (a policy network) as proposed by Luck et al. [20], and is an open-loop control sequence for all other methods. We try both *ES* algorithms due to their different performances on different tasks. For fair comparison, we finetune the parameters of *ES* and *RL* baselines and run the experiments with the best-performing parameters. For the *Control Only* baseline and our method, we use the default parameters provided in the Scipy's L-BFGS-B optimizer. While our method can solve the *Assemble* task with a high success rate, we found that the loss objective can be further decreased by

Table 5.4: **Normalized Metric Comparison.** We design the task-related metrics to measure how successful each method performs on the tasks. For *Finger Reach* task, the metric is the time-averaged distance to the target tracking points. For *Flip Box* and *Rotate Rubik's Cube*, the metrics are the flipping/rotating angle error at the end of the task. For *Assemble*, we measure the distance between the center of the small box and the center of the hole on the movable mount. All the metrics are normalized.

Task	Finger Reach	Flip Box	Rotate Rubik's Cube	Assemble
CMA-ES	0.39 ± 0.02	0.00 ± 0.00	$\mathbf{0.02 \pm 0.01}$	0.28 ± 0.03
(1+1)-ES	0.35 ± 0.04	0.69 ± 0.39	0.87 ± 0.15	0.39 ± 0.09
RL	0.61 ± 0.05	0.41 ± 0.48	0.79 ± 0.31	0.91 ± 0.11
L-BFGS-B (Control Only)	0.41 ± 0.00	1.00 ± 0.00	0.42 ± 0.39	0.77 ± 0.03
L-BFGS-B (Ours)	$\mathbf{0.17 \pm 0.01}$	$\mathbf{0.00 \pm 0.00}$	0.07 ± 0.09	$\mathbf{0.12 \pm 0.11}$

using a continuation method [133, 35]. Specifically, on the *Assemble* task, we scale down the contact forces at the beginning of the optimization to provide a smoother objective function space, and scale it up as the optimization proceeds. We set three stages with contact force scale equal to 0.01, 0.1 and 1, and start the next optimization stage once the previous stage converges. To apply the continuation method on the baseline algorithms, we fix the number of simulations in each stage and proceed to the next stage after the previous stage uses up the budget. We run all the baseline algorithms with and without continuation method on *Assemble* task, and plot the best performing one. We run all the methods on each task for five times with different random seeds and plot the average training curves in Figure 5-11. We further measure the successfulness of the tasks for each method by task-related metrics, and report in the Table 5.4.

Results The results show that our differentiable co-optimization framework is able to find better morphology and control solutions with significant better sample efficiency (10-30 times fewer simulated episode data) compared to the gradient-free *ES* baselines and model-free *RL* baseline. On *Finger Reach* task, while most methods (except *Control Only*) find a finger configuration that can reach the four target points, our method can find a morphology and a control sequence that can track the target points most accurately. On the most challenging and contact-rich task, *Assemble*, our method is the only one that is able to solve the task successfully.

We also performed an ablation study on the importance of morphology design by comparing the performance of our method and a *Control Only* baseline. The significant performance gain of our method over *Control Only* baseline reveals that incorporating the optimization of morphology design leads to easier optimization and better solutions. We show some of the optimized morphology designs from our method in Figure 5-12. On the *Flip Box* task, the optimized morphology has a hook-like structure at the finger tip, so that it is able to hook on the back surface of the box to flip over the box more easily. For the *Assemble* task, the optimized morphology has fingers of different lengths, so that the long right finger is able to push the smaller cube while the short left finger can hold the mount. Moreover, the design has flat and larger fingers, which allows the manipulator to push the object much more stably than a thin finger. More visual results can be found in our supplementary video.

5.2.3.3 Flexibility of the Morphology Parameterization

By adding more cage handles, we can easily increase the degrees of freedom for the morphology design. We show such flexibility of our deformation-based morphology parameterization in this section using a free-form gripper task motivated by Ha et al. [24]. As shown in Figure 5-13, the algorithm needs to optimize the shape of a pair of gripper fingers such that the gripper can grasp a diamond-like object using a predefined control sequence. Each finger starts with a cube-like shape and is optimized with free-form deformation.

To support deformation in higher degrees of freedom (DoF), we add more handles on the cage around each gripper finger. To show the advantage of using a deformation cage based parameterization, we compare it to a differentiable mesh-based parameterization (similar to the Truncated Signed Distance Function (TSDF) parameterization used in Ha et al. [24]) which directly optimize over the vertex positions of the mesh. For both parameterization methods, we only optimize the handles/mesh vertices on the inner side for each gripper finger as shown in Figure 5-13 (bottom row). We test both parameterization methods in our differentiable framework, and conduct 30

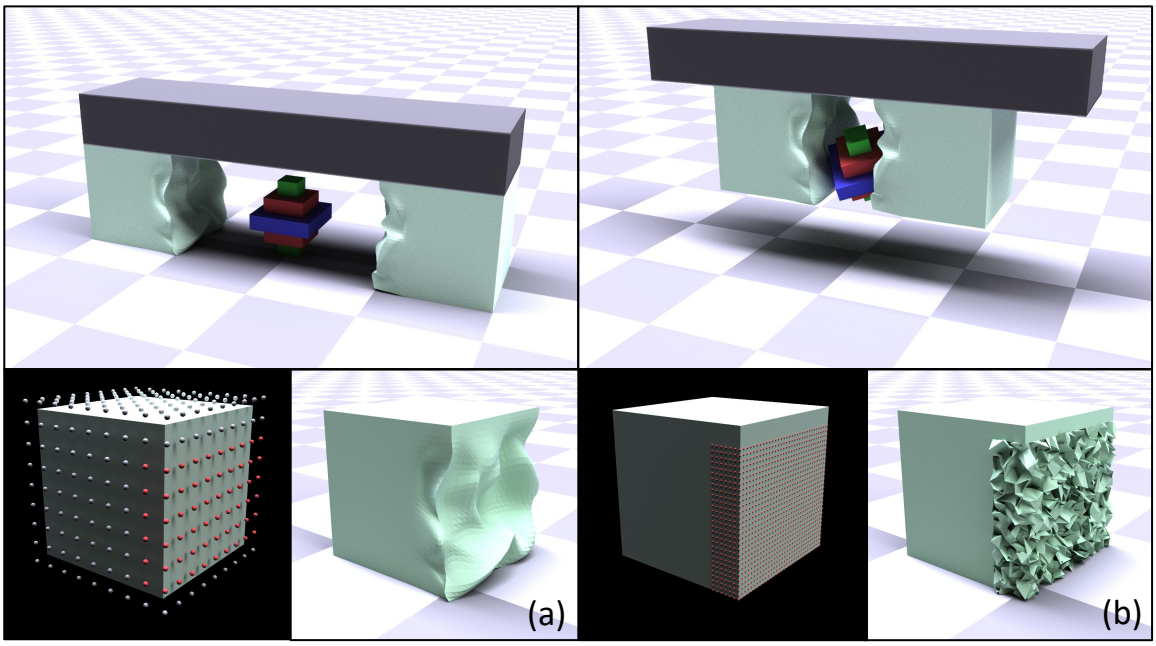


Figure 5-13: **Free-form Gripper**: The task is to pick up the object, as shown in the top row. We compare deformation-based parameterization (ours) and mesh-based parameterization. The optimization variables and optimized gripper morphology for the left gripper finger using both methods are shown in the bottom row. (a) *our parameterization method*: all the cage handles are shown on the left sub-figure and the ones used as optimization variables are highlighted in red. (b) *mesh-based parameterization*: we allow the optimization to directly optimize all the mesh vertices highlighted in red in the left sub-figure. In both cases, we do not modify the areas near the top of the gripper. The gripper morphology generated by our method is much smoother.

independent experiments for each method with different initial parameters.

Even though our deformation cage based parameterization has a much smaller optimization space and DoFs (396 optimization variables in the grasping task) than the mesh-based parameterization (8946 variables), our method is able to generate comparable success rates on the grasping task than the mesh-based parameterization (ours: 97%, mesh: 100%), and achieve better average loss (plot shown in Figure 5-14). Moreover, as shown in the Figure 5-13, our method is able to generate much smoother morphologies than the mesh parameterization which creates many reverted triangles on the mesh. Such advantage comes from the smoothness and feature-preserving properties of the cage-based deformation method.

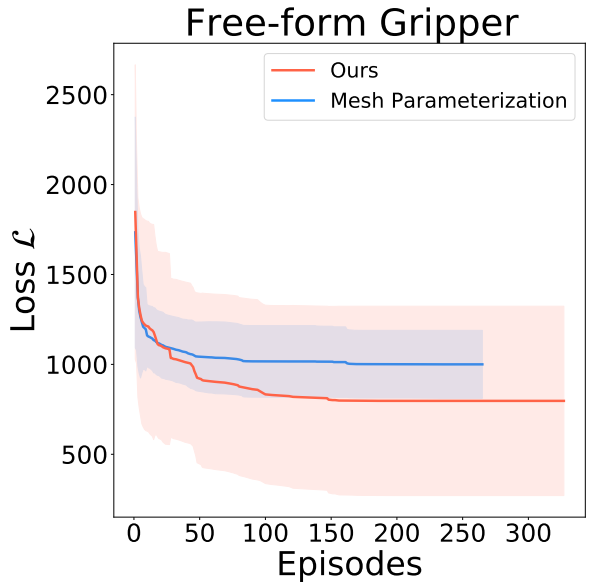
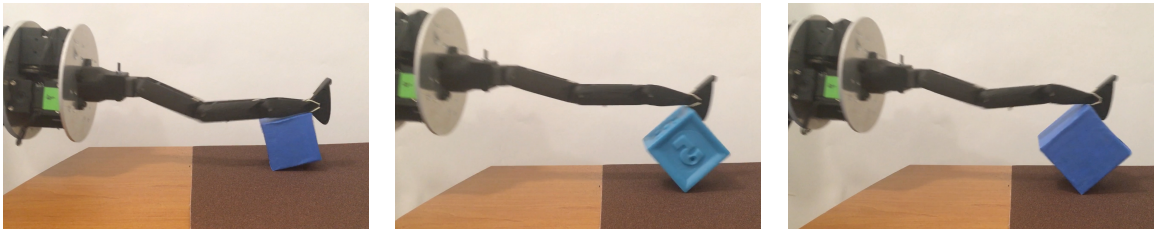


Figure 5-14: **Optimization curve comparison for *Free-form Gripper* task**. The horizontal axis is the number of simulation episodes during optimization, and the vertical axis is the loss value. The experiment results are averaged from 30 independent optimization runs with different initial guesses.

5.2.3.4 Manufacturing of Optimized Designs

We manufactured two optimized finger models from our method, one generated by the *Flip Box* task and the other generated by the *Assemble* task, as shown in Figure 5-9. The finger components were 3D printed on a Markforged printer using Onyx, a micro carbon fiber filled nylon, and assembled together after print. Minimal modification was required to prepare the program-generated models for printing, demonstrating a streamlined design process. This shows that our deformation-based morphology paramterization successfully maintains design manufacturability.

We further tested the functionality of the manufactured finger for the *Flip Box* task. Vectran cables were routed through the 3D printed finger and controlled by dynamixel DC servos at the base of the finger. The tendon-driven finger system was then mounted on a UR5 arm. We manually programmed a control sequence on the dynamixel motors and the UR5 arm to follow a series of waypoints from the trajectory optimized by the algorithm. The experiment shows that the manufactured finger can effectively flip the cube in real world. We also test the robustness of the optimized



(a) (b) (c)

Figure 5-15: **Test the optimized *Flip Box* manipulator design in real.** We test the robustness of the optimized design on the boxes of various sizes (a) 5 cm, (b) 5.5 cm, (c) 6 cm. Our manufactured design is able to successfully flip those boxes.

design on the cubes of various sizes and the experiment demonstrates that the finger can also perform the flipping tasks successfully. Please see the supplementary video of the associated paper for this real-world experiment.

5.2.4 Summary

In this work, we present an end-to-end differentiable framework for contact-aware robot designs. We focus on co-optimizing an open-loop control sequence and the continuous shape morphology parameters of the manipulator designs. At the core of our contribution is a novel deformation-based morphology parameterization for articulated robot designs, and a differentiable rigid body simulation carefully developed for contact-rich manipulation tasks. The experiments show that our innovative morphology parameterization approach provides us with an effective and expressive morphology design space. We also demonstrate that for a given manipulation task, by applying gradient-based optimization algorithm in our fully differentiable framework, our method is able to find a better morphology and control combination with significantly fewer number of simulation episodes than the state-of-the-art approaches. Furthermore, the optimized designs can be easily manufactured and are functional in real world.

Our deformation-based parameterization allows us to reuse the cages and the precomputed deformation weights for each individual component across different manipulator structures. By combining our discrete shape representation and the con-

tinuous shape representation, the hybrid shape representation (Seciton 3.3) enables the potential research on manipulator structure optimization in the future. It is also worth mentioning that while our examples only show the components with identical connection surfaces, it is not a limitation of our proposed deformation-based parameterization. One can easily applies our method for components with connection surfaces of different sizes by constructing proper cages for them.

Chapter 6

Multi-Objective Robot Optimization

The problems we have discussed so far are all defined as single-objective tasks, where the solution are optimized to maximize/minimize one single-objective performance metric. For example, in Section 4.1, we optimize the control policy for the manipulator to maximize its box pushing performance, and similarly in Section 5.1, we construct the algorithm that can find the optimal terrestrial robot structure to run as fast as possible on one specific terrain. However, most physical tasks in the real world are indeed evaluated by multiple metrics. In most cases, those different metrics/objectives are even conflicting to each other. For example, when we evaluate the driving skill of a driver, we will take into consideration both the speed and the safety of her driving.

Similar analogy exists in the robotic control problems. For example, when designing a control policy for a running quadruped robot, we need to consider two conflicting objectives: running speed and energy efficiency. In contrast to a *single-objective* environment, which measures performance using a single scalar value and where a single best solution exists, with a *multi-objective* problem, performance is measured using multiple objectives, and multiple optimal solutions exist. One optimal policy may prefer high speed at the cost of lower energy efficiency, whereas another optimal policy might prefer high energy efficiency at the cost of lower speed. In general, many optimal policies exist depending on the chosen trade-off between these two metrics.

Therefore, in such a multi-objective control problems, we are interested in the *Pareto Optimal* set of the control policies rather than a single optimal solution.

It is the same for the robot shape optimization problems. In general, we would like to find the robot shape/structure that is able to complete multiple tasks. For example, for a terrestrial robot, we may want it to have fast forward running speed, to have energy efficient gaits, and to be able to climb rough terrains. However, because form informs function and vice versa, it is natural that different robot designs will be better at different tasks and rarely will a single design be best at all tasks. Thus, an ideal co-design algorithm needs to extract robots with optimal trade-offs across different design objectives; *i.e.* the *Pareto set* of robot designs for the tasks at hand.

In this chapter, we present two techniques towards solving the multi-objective optimization problems for robots. In Section 6.1, we tackle the multi-objective control optimization problem where we efficiently find a set of Pareto Optimal control policies trading off different performance objectives and further show how to construct a continuous Pareto set representation by conducting Pareto analysis. In Section 6.2, we extend the Graph Heuristic Search algorithm in Section 5.1 to a multi-objective scenario, where the extended algorithm is able to find a set of terrestrial robot structure designs which are on the Pareto set of several different locomotion skills.

6.1 Prediction-Guided Multi-Objective Control Policy Learning

6.1.1 Motivation

Multi-objective problems have received significant attention because most real-world scenarios involve making trade-offs with respect to different performance metrics. This is especially true in robotic control, in which the notion of performance usually involves different conflicting objectives. For example, when designing a control policy for a running quadruped robot, we need to consider two conflicting objectives:

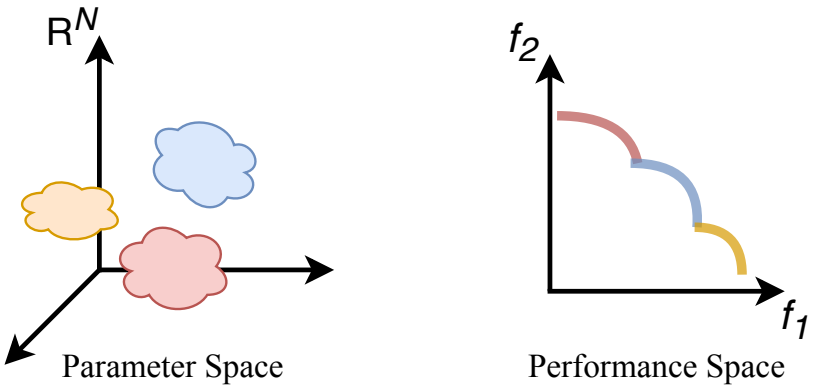


Figure 6-1: **Parameter space and performance space of the Pareto policies.** (Left) The Pareto set is composed from a disjoint set of policy families in the N dimensional parameter space. (Right) The policies from each family map to a continuous segment on the Pareto front in the performance space.

running speed and energy efficiency. In contrast to a *single-objective* environment, which measures performance using a single scalar value and where a single best solution exists, with a *multi-objective* problem, performance is measured using multiple objectives, and multiple optimal solutions exist. One optimal policy may prefer high speed at the cost of lower energy efficiency, whereas another optimal policy might prefer high energy efficiency at the cost of lower speed. In general, many optimal policies exist depending on the chosen trade-off between these two metrics. In the end, a human is responsible for selecting the preference among different metrics, and this determines the corresponding optimal policy.

One popular way of solving multi-objective control problems is to compute a meta policy [95]. A meta policy is a general policy that is not necessarily optimal but can be relatively quickly adapted to different trade-offs between performance objectives. Unfortunately, such adapted control policies are not necessarily optimal. For instance, adapting a general meta control policy for a quadruped robot to run as fast as possible will often result in a suboptimal policy for this metric.

In this work, we show that an effective representation for obtaining the best performance trade-offs for multi-objective robot control is a Pareto set of control policies. We empirically show that a Pareto set cannot be effectively represented using a single continuous policy family. Rather, a Pareto set is composed from a set of disjoint

policy families, each occupying a continuous manifold in the parameter space and being responsible for a segment on the Pareto front in the performance space (Figure 6-1).

To find such Pareto representations, we propose an efficient algorithm to compute the Pareto set of policies. Our algorithm works in two steps. In the first step, we find a dense and high-quality set of policies on the Pareto front using reinforcement learning strategies based on a novel prediction-guided evolutionary learning algorithm. In each generation, an analytical model is fitted for each policy to predict the expected improvement along each optimization direction. An optimization problem is then solved to select the policies and the associated optimization directions that are expected to best improve the quality of the Pareto. In the second step, we conduct a Pareto analysis on the computed Pareto-optimal policies to identify different policy families and to compute a continuous representation for each of these policy families.

In order to benchmark our proposed algorithm, we design a set of multi-objective robot control problems with a continuous action space. The performance of each policy can be evaluated using a physics-based simulation system [27]. Our experiments demonstrate that the proposed algorithm can efficiently find a significantly higher-quality set of Pareto-optimal policies than existing methods. Moreover, based on these policies it can reconstruct continuous policy families that span the whole Pareto front.

In overall, we propose our main technical contributions: a prediction-guided evolutionary learning algorithm for multi-objective control problems, and a Pareto analysis tool to construct a continuous Pareto representation. The preliminaries and background is introduced in Section 6.1.2, an overview of the algorithm is provided in Section 6.1.3, and the details of our main contributions are described in Sections 6.1.4-6.1.5. Extensive experiments are conducted and analyzed in Section 6.1.6.

6.1.2 Preliminaries

6.1.2.1 Multi-Objective Markov Decision Process

A multi-objective control problem can be formulated as a multi-objective Markov Decision Process (MOMDP), which is defined by the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \boldsymbol{R}, \boldsymbol{\gamma}, \mathcal{D} \rangle$ with state space $\mathcal{S}$, action space $\mathcal{A}$, state transition probability $\mathcal{P}(s' \mid s, a)$, vector of reward functions $\boldsymbol{R} = [\boldsymbol{r}_1, ..., \boldsymbol{r}_m]^\top$ with $\boldsymbol{r}_i : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, vector of discount factors $\boldsymbol{\gamma} = [\gamma_1, ..., \gamma_m]^\top \in [0, 1]^m$, initial state distribution $\mathcal{D}$, and the number of objectives m.

In MOMDPs, a policy $\pi_{\boldsymbol{\theta}} : \mathcal{S} \to \mathcal{A}$ is associated with a vector of expected returns $\boldsymbol{J}^\pi = [J_1^\pi, ..., J_m^\pi]^T$, where

$$J_i^\pi = \mathbb{E}\left[\sum_{t=0}^{T} \gamma_i^t \boldsymbol{r}_i(s_t, a_t) \mid s_0 \sim \mathcal{D}, a_t \sim \pi_{\boldsymbol{\theta}}(s_t)\right].$$

The state s_{t+1} is reached from state s_t by action a_t, and T is the horizon. We use π for $\pi_{\boldsymbol{\theta}}$ for brevity.

6.1.2.2 Multi-Objective Optimization

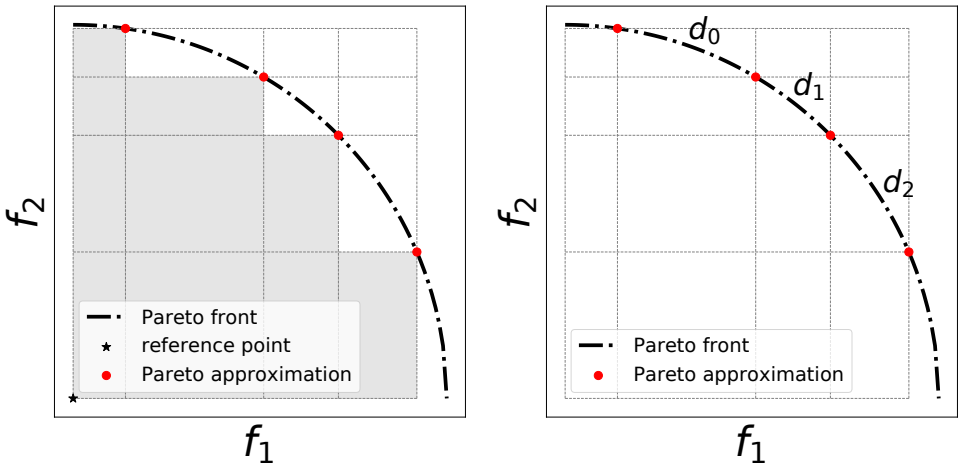
A multi-objective optimization problem is formulated as:

$$\max_\pi \mathbf{F}(\pi) = \max_\pi [f_1(\pi), f_2(\pi), ..., f_m(\pi)],$$

where m is the number of objectives, π is the policy, and in our problem $f_i(\pi) = J_i^\pi$.

In multi-objective optimization problems, no single optimal policy exists that maximizes all the objectives. Instead a set of non-dominated solutions called the Pareto set is desired:

Definition 6.1.1 *(Pareto optimality) We say policy π dominates policy π' if $\boldsymbol{F}(\pi) \geq \boldsymbol{F}(\pi')$ and $\boldsymbol{F}(\pi) \neq \boldsymbol{F}(\pi')$. A policy π is Pareto optimal if and only if it is not*



(a) Hypervolume metric (b) Sparsity metric

Figure 6-2: **Pareto Metrics.** (a) Hypervolume metric in 2-objective space is the area (shaded) dominated by the Pareto front approximation and dominating the reference point. (b) Sparsity metric in 2-objective space measures the average square distance between consecutive points in Pareto approximation. In this case, $\mathcal{S} = \frac{1}{3}(d_0^2 + d_1^2 + d_2^2)$.

dominated by any other policies. The set of all such policies is called the Pareto set, and the image of the Pareto set in the objective space is called the Pareto front.

Since the true (optimal) Pareto set is usually impossible to obtain in complex problems, the goal of multi-objective optimization is to find the set of solutions that best approximates the optimal Pareto set. To measure the quality of an approximated Pareto front, two factors are usually considered [134]: the convergence towards the true Pareto front and the uniformity of the solution distribution, which are best measured by hypervolume metric [135] (illustrated in Figure 6-2(a)):

Definition 6.1.2 *(Hypervolume metric) Let P be a Pareto front approximation in an m-dimensional objective space and $\boldsymbol{r} \in \mathbb{R}^m$ be the reference point. Then the hypervolume metric $\mathcal{H}(P)$ is*

$$\mathcal{H}(P) = \int_{\mathbb{R}^m} 1_{H(P)}(z)dz, \tag{6.1}$$

where $H(P) = \{\boldsymbol{z} \in Z \mid \exists 1 \leq i \leq |P| : \boldsymbol{r} \preceq \boldsymbol{z} \preceq P(i)\}$. $P(i)$ is the i-th solution in P,

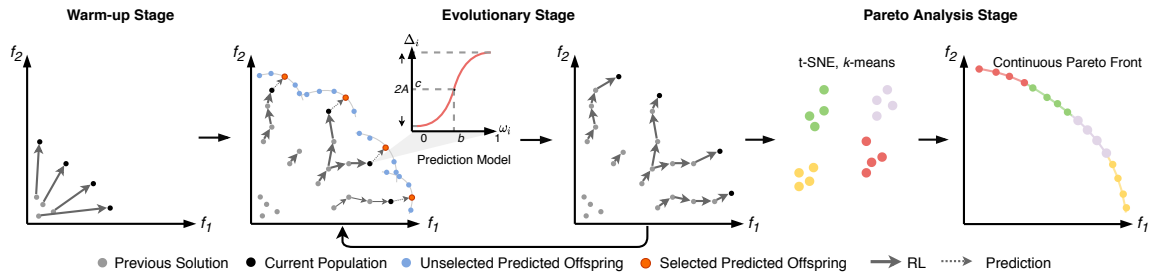


Figure 6-3: **Overview of the algorithm.** *Warm-up stage*: optimize n initial policies with different weights. *Evolutionary stage*: build an improvement prediction model for each policy and solve a prediction-guided optimization to select n best policy-weight pairs to be processed. The resulting polices are used to update the population and the prediction models. *Pareto analysis stage*: identify different policy families and construct a continuous Pareto representation.

$\preceq$ is the relation operator of objective dominance, and $1_{H(P)}$ is a Dirac delta function that equals 1 if $z \in H(P)$ and 0 otherwise.

A dense policy set is always preferred for better Pareto approximation. Therefore, a sparsity metric is also defined to measure that property (illustrated in Figure 6-2(b)):

Definition 6.1.3 *(Sparsity metric) Let P be a Pareto front approximation in an m-dimensional objective space. Then the Sparsity metric $\mathcal{S}(P)$ is*

$$\mathcal{S}(P) = \frac{1}{|P|-1} \sum_{j=1}^{m} \sum_{i=1}^{|P|-1} (\tilde{P}_j(i) - \tilde{P}_j(i+1))^2, \tag{6.2}$$

where $\tilde{P}_j$ is the sorted list for the j-th objective values in P, and $\tilde{P}_j(i)$ is the i-th value in this sorted list.

In a word, a desired Pareto set approximation is expected to have high hypervolume metric and low sparsity metric.

6.1.3 Algorithm Overview

As shown in Figure 6-3 and Algorithm 3, we propose an efficient algorithm to compute the Pareto set of policies. Our algorithm starts from a warm-up stage. In this stage,

n policies are randomly initialized, and each of them is optimized by multi-objective policy gradient (MOPG) (Algorithm 4 and Section 6.1.4.1) with one of n evenly distributed non-negative weights $\{\boldsymbol{\omega}_i\}$ $(\sum_j \omega_{i,j} = 1, 1 \leq i \leq n)$ for a specified number of iterations. The resulting policies form the first generation of the policy population. The warm-up stage is crucial for the whole algorithm to get the initial policies out of the low-performance region, where the learning process is usually highly noisy and unpredictable.

Next, the algorithm proceeds with the evolutionary stage. In each generation, an analytical model for each policy in the population is learned from past reinforcement learning data to predict the expected improvement along each optimization weight (Section 6.1.4.2). This prediction model is then used to guide a selection optimization algorithm to select n policy-weight pairs (we call RL tasks), which are expected to improve the quality of the Pareto set the most (Section 6.1.4.3). Finally, the selected tasks are optimized by multi-objective policy gradient algorithms for a fixed number of iterations in parallel to produce the new offspring policies, which are used to update the policy population. For the population update, we adopt the performance buffer strategy [136] to maintain the performance and diversity of the solutions. The evolutionary stage terminates when reaching the maximum number of generations. Through the whole evolutionary stage, an external Pareto archive is maintained to store all non-dominated intermediate policies and output as the approximated Pareto set when the evolutionary stage ends.

Once a discrete set of Pareto policies has been found, the algorithm conducts a Pareto analysis on the computed policies to identify different policy families, and then a continuous representation of the Pareto set is extracted by intra-family interpolation (Section 6.1.5).

Algorithm 3: Prediction-Guided MORL Algorithm

Input: #parallel tasks n, #warm-up iterations m_w, #task iterations m_t, #generations M.
Initialize population $\mathcal{P}$, external pareto archive EP, and RL history record $\mathcal{R}$.
▷ Warm-up Stage
Generate task set $\mathcal{T} = \{(\pi_i, \boldsymbol{\omega}_i)\}_{i=1}^{n}$ by random initial policies and evenly distributed weight vectors.
$\mathcal{P}' \leftarrow \text{MOPG}(\mathcal{T}, m_w, \mathcal{R})$ (Section 6.1.4.1)
Update $\mathcal{P}$ and EP with $\mathcal{P}'$.
▷ Evolutionary Stage
for *generation* $\leftarrow 1, 2, ..., M$ **do**
  Fit improvement prediction models $\{\boldsymbol{\Delta}^i\}$ for each policy in $\mathcal{P}$ from data in $\mathcal{R}$. (Section 6.1.4.2)
  $\mathcal{T} \leftarrow$TaskSelection$(n, \mathcal{P}, \{\boldsymbol{\Delta}^i\}, \text{EP})$ (Section 6.1.4.3)
  $\mathcal{P}' \leftarrow \text{MOPG}(\mathcal{T}, m_t, \mathcal{R})$ (Section 6.1.4.1)
  Update $\mathcal{P}$ and EP with $\mathcal{P}'$.
end for
▷ Pareto Analysis Stage
Compute families in EP and construct a continuous Pareto representation. (Section 6.1.5)
Output: The continuous Pareto representation.

6.1.4 Prediction-Guided MORL

6.1.4.1 Multi-Objective Policy Gradient

Given a policy $\pi_{\boldsymbol{\theta}}$ and a weight vector $\boldsymbol{\omega}(\sum_i \omega_i = 1)$, our multi-objective policy gradient worker aims to optimize the policy to maximize the weighted-sum reward $\mathcal{J}(\boldsymbol{\theta}, \boldsymbol{\omega})$:

$$\mathcal{J}(\boldsymbol{\theta}, \boldsymbol{\omega}) = \boldsymbol{\omega}^\top \mathbf{F}(\pi) = \sum_{i=1}^{m} \omega_i f_i(\pi) = \sum_{i=1}^{m} \omega_i J_i^\pi.$$

The most straight forward way is converting the environment from returning a vector of rewards into a scalar weighted-sum reward, regarding it as a single-objective control problem, and solving it with any single-objective policy gradient algorithm. Most policy gradient algorithms simultaneously learn a value function $V(s)$ and a policy network $\pi(a|s)$. The value function receives the current state s and estimates

the expected return (expected weighted-sum return here) under following the current policy and is used to lower the training variance.

However, with our evolutionary learning algorithm, a policy will be selected to be optimized with different weights during the whole learning process. It is inefficient to simply modify the environment to return a scalar weighted-sum reward and optimize the policy by a single-objective policy gradient algorithm. With this naive approach, the value network trained with previous weights would be invalid for the new weight and would need to be trained from scratch. Therefore, we improve the single-objective policy gradient algorithm by extending the value function to be vectorized, which shares a similar strategy as applied in multi-objective Q-learning [97].

Specifically, the vectorized value function $\boldsymbol{V}^{\pi}(s)$ maps a state s to the vector of expected returns under the current policy π. In this way, the value function is still valid when the optimization weight changes and can be directly used to train the policy for the new weight and quickly adapt its output to the new policy. The parameters of the value function are updated by Bellman Equation with a squared-error loss $\|\boldsymbol{V}^{\pi}(s_t) - \hat{\boldsymbol{V}}(s_t)\|^2$. $\hat{\boldsymbol{V}}(s_t)$ is the target value and computed by:

$$\boldsymbol{V}^{\pi}(s_t) = \hat{\boldsymbol{V}}(s_t) = \sum_{u} \pi(u|s_t)(\boldsymbol{R}(s_t) + \gamma \boldsymbol{V}^{\pi}(s_{t+1})),$$

where $\boldsymbol{R}(s_t)$ is the multi-objective reward vector of state s_t, and s_{t+1} is reached from state s_t by action u.

Algorithm 4: MOPG

Input: task set $\mathcal{T}$, #iterations m, RL history record $\mathcal{R}$.
Initialize offspring population P'.
for all task $(\pi_i, \boldsymbol{\omega}_i) \in \mathcal{T}$ **do**
  Run multi-objective policy gradient for task $(\pi_i, \boldsymbol{\omega}_i)$ for m iterations by Eq. 6.3.
  Collect the result policy π_i' in P'.
  Store $(\mathbf{F}(\pi_i), \mathbf{F}(\pi_i'), \boldsymbol{\omega}_i)$ in $\mathcal{R}$.
end for
Output: Offspring population $\mathcal{P}'$.

To update the policy, the policy gradient is extended to be:

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta}, \boldsymbol{\omega}) &= \sum_{i=1}^{m} \omega_i \nabla_{\boldsymbol{\theta}} J_i(\boldsymbol{\theta}) \\
&= \sum_{i=1}^{m} \omega_i \mathbb{E}\left[\sum_{t=0}^{T} A_i^{\pi}(s_t, a_t) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_t|s_t) \right] \\
&= \mathbb{E}\left[\sum_{t=0}^{T} \boldsymbol{\omega}^{\top} \boldsymbol{A}^{\pi}(s_t, a_t) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_t|s_t) \right] \\
&= \mathbb{E}\left[\sum_{t=0}^{T} A_{\boldsymbol{\omega}}^{\pi}(s_t, a_t) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_t|s_t) \right], \qquad (6.3)
\end{aligned}$$

where $\boldsymbol{A}^{\pi}(s_t, a_t)$ is the vectorized advantage function. In our extension, the new advantage function $A_{\boldsymbol{\omega}}^{\pi}(s_t, a_t)$ is simply represented as a weighted-sum scalarization of the advantage functions for individual objectives.

Such value network and policy gradient extension can be easily applied to most existing policy gradient methods. In our implementation, we choose to adapt the Proximal Policy Optimization (PPO) [60] into our multi-objective weighted-sum version, where the clipped surrogate objective is applied to update the policy parameters, and the Generalized Advantage Estimation [137] is used to compute the advantage function and the target values.

6.1.4.2 Policy Improvement Prediction Model

In this section, we present our prediction model for policy improvement. Given a policy π and a weight $\boldsymbol{\omega}$, the prediction model aims to predict the improvement of the objectives after applying the policy gradient on the policy π with the weight $\boldsymbol{\omega}$ for m_t iterations. However it is challenging due to the small amount of reinforcement learning history data we can collect during the learning process. Therefore, a concise analytical model with few parameters needs to be considered.

We propose a monotonic hyperbolic model based on an intuitive observation that the more weight put on one objective, the better that objective can be optimized. Formally speaking, if we run multi-objective policy gradient for a policy π with objectives $\mathbf{F}(\pi) = [f_1(\pi), f_2(\pi), ..., f_m(\pi)]$ with weights $\boldsymbol{\omega}_1$ and $\boldsymbol{\omega}_2$ separately (where $\omega_{1,1} > \omega_{2,1}$), the resulting policies π_1 and π_2 should satisfy the monotonic property that $\Delta f_1(\pi_1) = f_1(\pi_1) - f_1(\pi) \geq f_1(\pi_2) - f_1(\pi) = \Delta f_1(\pi_2)$. Furthermore, the improvement function should be bounded on two sides. Based on these observations, we construct the following four-parameter hyperbolic model $\Delta_j^i(\omega_j)$ for each policy π_i and each objective f_j:

$$\Delta_j^i(\omega_j) = A \cdot \frac{e^{a(\omega_j - b)} - 1}{e^{a(\omega_j - b)} + 1} + c. \tag{6.4}$$

The function is illustrated in Figure 6-3 (middle left), where $\boldsymbol{\xi} = \{A, a, b, c\}$ are the four parameters that need to be determined for each model. In order to fit the parameters, we record the objective improvements of reinforcement learning every m_t iterations in a record data structure $\mathcal{R}$. Each entry in $\mathcal{R}$ is a triplet $(\mathbf{F}(\pi), \mathbf{F}(\pi'), \boldsymbol{\omega})$, where $\mathbf{F}(\pi)$ and $\mathbf{F}(\pi')$ are the objectives for the policy before and after being optimized by reinforcement learning for m_t iterations, and $\boldsymbol{\omega}$ is the optimizing weight. As shown in Figure 6-3, $\mathcal{R}$ is a directed graph (precisely a directed rooted forest) storing the full RL optimization history for each policy.

In each generation, for each policy π_i, the data $\{(\boldsymbol{\omega}, \Delta\mathbf{F})\} = \{(\boldsymbol{\omega}, \mathbf{F}(\pi') - \mathbf{F}(\pi))\}$ in the neighborhood of the policy π_i (*i.e.*, $\|\mathbf{F}(\pi) - \mathbf{F}(\pi_i)\| < \delta\|\mathbf{F}(\pi_i)\|$) is collected, and the following nonlinear least-square regression is applied to fit the parameters of

the hyperbolic model:

$$\min_{\boldsymbol{\xi}} \sum_{k=1}^{n} \rho((\Delta_j^i(\boldsymbol{\omega}_j^i; \boldsymbol{\xi}) - \Delta \mathbf{F}_j^i)^2),$$

where n is the size of the dataset and $\rho(z) = 2(\sqrt{1+z} - 1)$ is the soft-l_1 loss. For the threshold δ, we set it as 0.1 in all experiments. In the rare cases that there is not enough data around the policy π_i (< 4 data points), we iteratively relax the threshold δ until enough data points are collected.

6.1.4.3 Prediction-Guided Optimization for Task Selection

Taking into account the hypervolume (Eq. 6.1) and sparsity (Eq. 6.2) metrics, we propose a prediction-guided algorithm for task selection from first principles.

In each generation, our algorithm aims to select the most important tasks (pairs of policy and weight) that can best improve the Pareto metrics. Specifically, the algorithm needs to select n tasks $\mathcal{T}_i$ to be processed by multi-objective policy gradient for m_t iterations. Here each task $\mathcal{T}_i$ is composed by a pair of policy π_i from current population $\mathcal{P}$ and an optimization weight $\boldsymbol{\omega}_i$. The selected tasks seek to maximize a weighted mixture metric $\mathcal{H}(\boldsymbol{F}(\texttt{EP}^*)) + \alpha \mathcal{S}(\boldsymbol{F}(\texttt{EP}^*))$ ($\alpha < 0$ for minimizing the sparsity metric), where $\texttt{EP}^*$ is the new Pareto set after inserting the offspring policies from those tasks. Guided by the prediction models trained for each policy, we can predict the expected objectives of the new offspring policy for each task as $\mathbf{F}(\pi_i) + \boldsymbol{\Delta}^i(\boldsymbol{\omega}_i)$, and can formulate this optimization problem as:

$$\begin{aligned} \max_{\mathcal{T}=\{(\pi_i,\omega_i)\}_{i=1}^n} \quad & \mathcal{Q}(\texttt{EP}, \mathcal{T}) = \mathcal{H}(P) + \alpha \mathcal{S}(P) \\ \text{with } P &= \mathbf{F}(\texttt{EP}^*) \\ &= \texttt{Pareto}(\mathbf{F}(\texttt{EP}) \cup \{\mathbf{F}(\pi_i) + \boldsymbol{\Delta}^i(\boldsymbol{\omega}_i)\}), \end{aligned} \tag{6.5}$$

where $\texttt{EP}$ is the current Pareto archive, and $\texttt{Pareto}$ is the function computing the Pareto front from a set of objectives.

The optimization problem in Eq. 6.5 is a mixed-integer programming problem,

Algorithm 5: Prediction-Guided Task Selection

Input: #tasks n, population $\mathcal{P}$, improvement prediction models $\{\boldsymbol{\Delta}^i\}$, Pareto archive EP.
Initialize task set $\mathcal{T}$ and virtual Pareto archive $\texttt{EP}^* = \texttt{EP}$.
for $i \leftarrow 1, 2, ..., n$ **do**
  Initialize task $\mathcal{T}_i \leftarrow$ None
  for all $\pi_i \in \mathcal{P}$ and $\boldsymbol{\omega} \in$ candidate weights **do**
    if $(\pi_i, \boldsymbol{\omega})$ has not been selected and
        $\mathcal{Q}(\texttt{EP}^*, (\pi_i, \boldsymbol{\omega})) > \mathcal{Q}(\texttt{EP}^*, \mathcal{T}_i)$ **then**
      $\mathcal{T}_i \leftarrow (\pi_i, \boldsymbol{\omega})$
    end if
  end for
  Append task $\mathcal{T}_i$ into $\mathcal{T}$.
  Update $\texttt{EP}^*$ by inserting the predicted offspring of $\mathcal{T}_i$.
end for
Output: Selected task set $\mathcal{T}$.

which is difficult to solve directly. Therefore, we approximate it by discretizing the continuous weight to K candidate sample weights (Figure 6-3 middle left) and instead solve a knapsack problem: given $K \times |\mathcal{P}|$ candidate points in the objective space, we want to select n of them to maximize the mixture metric after inserting them into the current Pareto archive EP. Although in the two objective case, it can be solved by dynamic programming in polynomial time complexity, exactly solving the knapsack problem in general is an NP-hard problem. Therefore, in order to improve the generalizability of the algorithm, we adopt a greedy algorithm (Algorithm 5). Our greedy algorithm maintains a virtual policy set $\texttt{EP}^*$ for the predicted Pareto archive. It then iteratively selects the task that best improves the Pareto metric of $\texttt{EP}^*$ and then updates $\texttt{EP}^*$ by inserting the predicted offspring policy of the selected task.

6.1.5 Pareto Analysis and Continuous Pareto Representation

Once a set of Pareto optimal policies is computed from the evolutionary stage, we conduct a Pareto analysis to analyze the structure of policy parameters on the Pareto front.

Since the deep neural network policies are not linearly co-related, we use t-SNE

[138], which is a standard nonlinear dimensionality reduction method, to embed the high dimensional policy parameter space into a lower dimensional space for better visualization. In our case, we found mapping to two dimensional space to work well.

For the purpose of dimensionality reduction, there are also other available methods (e.g., LLE, PCA, Isomap). We choose t-SNE due to its better visualization effect. We provide a comparison with other dimensionality reduction methods in Appendix E.5.

Once the embedding is generated, we use k-means to cluster the reduced policies into several families as illustrated in Figure 6-3 (right). As expected, the whole Pareto-optimal set is composed from several disjoint policy families, and each family is responsible for a continuous segment on the Pareto front. A continuous Pareto representation is then constructed by linearly interpolating the policies inside the same family. For any target objectives on the continuous Pareto front approximation, we first identify which policy family can cover those target objectives and then linearly interpolate the parameters of the nearby policies. Although the deep neural networks are not linearly co-related, we surprisingly find that by computing a dense Pareto approximation set and conducting the Pareto analysis, such intra-family interpolation works successfully, which is demonstrated by the results in Section 6.1.6.3.

6.1.6 Experiments

6.1.6.1 Benchmark Problems

In order to benchmark our proposed algorithm, we design seven multi-objective RL environments with continuous action space based on Mujoco [27]. We keep the same state spaces $\mathcal{S}$ and action spaces $\mathcal{A}$ as used in Mujoco for most problems. We make several modifications to the physical parameters of some robots (e.g. mass, friction, actuator limit) for working better on the multiple objectives, which can be found in our provided code. The description of each environment is illustrated as follows, where R_i means the reward for the i-th objective, and the reward function are designed so that the values are in similar scale.

1) *HalfCheetah-v2*: (two objectives) Observation and action space dimensionality: $\mathcal{S} \in \mathbb{R}^{17}, \mathcal{A} \in \mathbb{R}^{6}$, and the environment runs for 500 steps.

The first objective is forward speed:

$$R_1 = \min(v_x, 4) + C$$

The second objective is energy efficiency:

$$R_2 = 4 - \sum_i a_i^2 + C$$

where $C = 1$ is the alive bonus, v_x is the speed in x direction, a_i is the action of each actuator.

2) *Hopper-v2*: (two objectives) Observation and action space dimensionality: $\mathcal{S} \in \mathbb{R}^{11}, \mathcal{A} \in \mathbb{R}^{3}$, and the environment runs for 500 steps.

The first objective is forward speed:

$$R_1 = 1.5v_x + C$$

The second objective is jumping height:

$$R_2 = 12(h - h_{init}) + C$$

where $C = 1 - 0.0002 \sum_i a_i^2$ is composed of alive bonus and energy efficiency, v_x is the speed in x direction, h is the current height, h_{init} is the initial height, a_i is the action of each actuator.

3) *Swimmer-v2*: (two objectives) Observation and action space dimensionality: $\mathcal{S} \in \mathbb{R}^{8}, \mathcal{A} \in \mathbb{R}^{2}$, and the environment runs for 500 steps.

The first objective is forward speed:

$$R_1 = v_x$$

The second objective is energy efficiency:

$$R_2 = 0.3 - 0.15 \sum_i a_i^2$$

where v_x is the speed in x direction, a_i is the action of each actuator.

4) *Ant-v2*: (two objectives) Observation and action space dimensionality: $\mathcal{S} \in \mathbb{R}^{27}, \mathcal{A} \in \mathbb{R}^8$, and the environment runs for 500 steps.

The first objective is x-axis speed:

$$R_1 = v_x + C$$

The second objective is y-axis speed:

$$R_2 = v_y + C$$

where $C = 1 - 0.5 \sum_i a_i^2$ is composed of alive bonus and energy efficiency, v_x is x-axis speed, v_y is y-axis speed, a_i is the action of each actuator.

5) *Walker2d-v2*: (two objectives) Observation and action space dimensionality: $\mathcal{S} \in \mathbb{R}^{17}, \mathcal{A} \in \mathbb{R}^6$, and the environment runs for 500 steps.

The first objective is forward speed:

$$R_1 = v_x + C$$

The second objective is energy efficiency:

$$R_2 = 4 - \sum_i a_i^2 + C$$

where $C = 1$ is the alive bonus, v_x is the speed in x direction, a_i is the action of each actuator.

6) *Humanoid-v2*: (two objectives) Observation and action space dimensionality:

$\mathcal{S} \in \mathbb{R}^{376}, \mathcal{A} \in \mathbb{R}^{17}$, and the environment runs for 1000 steps.

The first objective is forward speed:

$$R_1 = 1.25v_x + C$$

The second objective is energy efficiency:

$$R_2 = 3 - 4\sum_i a_i^2 + C$$

where $C = 3$ is the alive bonus, v_x is the speed in x direction, a_i is the action of each actuator.

7) *Hopper-v3*: (three objectives) Observation and action space dimensionality: $\mathcal{S} \in \mathbb{R}^{11}, \mathcal{A} \in \mathbb{R}^{3}$, and the environment runs for 500 steps.

The first objective is forward speed:

$$R_1 = 1.5v_x + C$$

The second objective is jumping height:

$$R_2 = 12(h - h_{init}) + C$$

The third objective is energy efficiency:

$$R_3 = 4 - \sum_i a_i^2 + C$$

where $C = 1$ is the alive bonus, v_x is the speed in x direction, h is the current height, h_{init} is the initial height, a_i is the action of each actuator.

6.1.6.2 Experiment Setup

We implement our prediction-guided evolutionary learning algorithm as described in Section 6.1.4 and Section 6.1.5, and implement five baseline algorithms for comparison. [1]

1) *RA*: The Radial Algorithm assigns a set of weights and runs reinforcement learning to optimize the policies for each weight separately [92].

2) *PFA*: In the evolutionary stage, we gradually fine tune the weight of the RL to cover the whole Pareto front, which is an adaptation of original PFA algorithm [92] to DRL setting.

3) *MOEA/D*: Multi-Objective Evolutionary Algorithm based on Decomposition [87] decomposes the problem into subproblems by different weights and solves them in a collaborative way.

4) *RANDOM*: A random selection strategy is designed to uniformly sample RL task in each generation.

5) *META*: A meta-learning based MORL method [95] trains a meta policy and then adapts the meta policy to the policies for different preferences in a few iterations.

To fairly compare the baseline algorithms to ours, we implement the first four baselines in a common framework with our proposed algorithm and apply the same population strategy and external Pareto archive to them. For the META, our implementation is based on the codebase [139] which implements Model-Agnostic Meta-Learning [140] and generates the Pareto approximation by adapting the meta-policy to N uniformly sampled weights (we set N as a large number compared to the number of solutions in other methods). Furthermore, we set all the shared hyperparameters to be the same and run all algorithms with same amount of simulation steps.

[1] The code can be found at https://github.com/mit-gfx/PGMORL

Table 6.1: **Evaluation of our algorithm and baseline algorithms on the proposed benchmark problems.** We run all algorithms on each problem for 6 runs and report the average Hypervolume (Hv) and Sparsity (Sp) metrics. Bold number is the best in each row.

Example	Metric	Ours	RA	PFA	MOEA/D	RANDOM	META
HalfCheetah-v2	Hv ($\times 10^6$)	**5.77**	5.66	5.75	5.61	5.69	5.18
	Sp ($\times 10^3$)	**0.44**	15.87	3.81	16.96	1.09	2.13
Hopper-v2	Hv ($\times 10^7$)	2.02	1.96	1.90	**2.03**	1.88	1.25
	Sp ($\times 10^4$)	**0.50**	5.99	3.96	2.73	1.20	4.84
Swimmer-v2	Hv ($\times 10^4$)	**2.57**	2.33	2.35	2.42	2.38	1.23
	Sp ($\times 10^1$)	**0.99**	4.43	2.49	5.64	1.94	2.44
Ant-v2	Hv ($\times 10^6$)	**6.35**	5.98	6.23	6.28	5.54	2.40
	Sp ($\times 10^4$)	**0.37**	5.50	1.56	1.97	1.13	1.56
Walker2d-v2	Hv ($\times 10^6$)	**4.82**	4.15	4.16	4.44	4.11	2.10
	Sp ($\times 10^4$)	**0.04**	0.74	0.37	1.28	0.07	2.10
Humanoid-v2	Hv ($\times 10^7$)	4.64	3.53	3.70	**4.65**	3.21	-
	Sp ($\times 10^4$)	**0.19**	4.50	0.38	3.82	0.42	-
Hopper-v3	Hv ($\times 10^{10}$)	**3.74**	3.50	-	3.64	3.36	2.15
	Sp ($\times 10^3$)	**0.03**	0.61	-	0.58	0.27	12.48

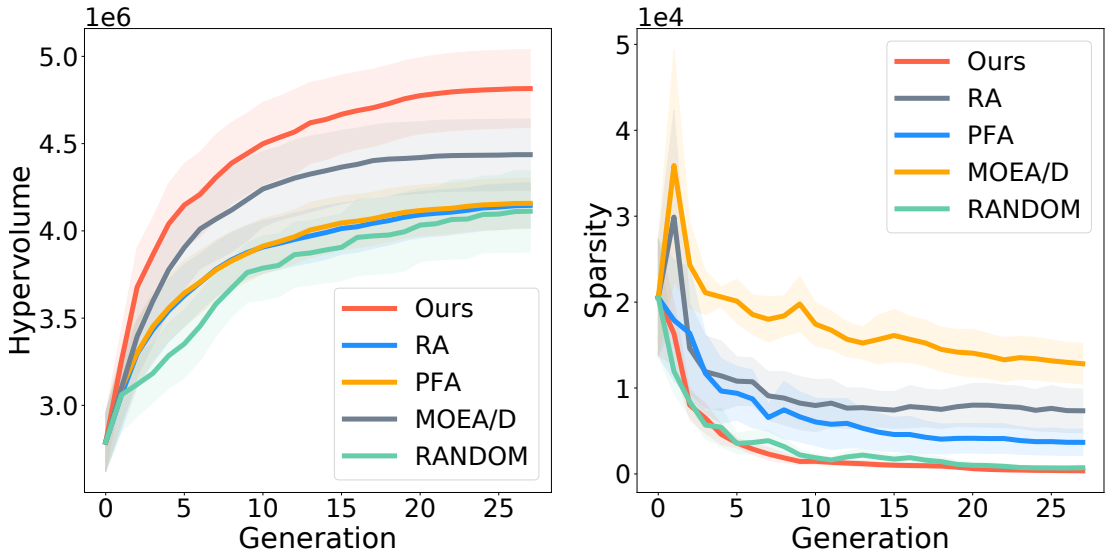
6.1.6.3 Results

We test the performance of our algorithm and all the baselines on the proposed benchmark problems. The training details and parameters are reported in Appendix D.2. We provide more visual results in the supplementary video. [2]

Pareto Quality Comparison We first use the hypervolume metric (Eq. 6.1) and the sparsity metric (Eq. 6.2) to compare the quality of the computed Pareto set approximations. We run each algorithm on each problem for six times and report the average metrics in Table 6.1. The training curves on Walker2d-v2 problem are shown in Figure 6-4. We provide the learning curve and Pareto front comparison results on other problems in Figure 6-8, Figure 6-9 and Figure 6-10.

The results in Table 6.1 demonstrate that our proposed algorithm outperforms all

[2] https://people.csail.mit.edu/jiex/papers/PGMORL/video.mp4



(a) Hypervolume (b) Sparsity

Figure 6-4: **The learning curves of our algorithm and baseline algorithms on Walker2d-v2.** The x-axis is the generation, the y-axis is the metric and the shadow area is the standard deviation. The Hypervolume at generation 0 is measured after the warm-up stage. The learning curve of META is not plotted as its metrics can only be measured during the final adaptation stage. (a) Hypervolume metric (higher is better). (b) Sparsity metric (lower is better).

the baselines on most benchmark problems in both metrics. The training curves show that our prediction-guided algorithm is able to select the important reinforcement learning tasks to improve the Pareto quality much more efficiently than the baseline methods.

RA archives high-performance solutions in some regions on the Pareto but the solutions are spread sparsely in the performance space because RA assigns all computing resources into optimizing for those pre-selected weights.

PFA generates denser Pareto approximations than RA because it finetunes the optimization weights to cover the whole weight range. However, because it blindly changes a weight to its neighboring weight, a good policy is unable to transfer its knowledge to wider range of weights. Therefore, it can only recover some pieces of the Pareto front. In our algorithm, a policy can be optimized along the whole Pareto front as long as it can improve the Pareto quality. Moreover, PFA is hard to extend

to the three-objective case as the sequence of weights in three dimension is undefined.

MOEA/D is the most competitive baseline on hypervolume metric as it periodically shares the better solutions across subproblems. However, it also suffers from high sparsity.

RANDOM computes the densest Pareto approximation in all baselines as it distributes the RL tasks evenly onto every weight and policy. However, the random task selection strategy leads to the low-performance of the computed Pareto front, which is reflected by the low hypervolume metric.

META computes a compromise policy family that can perform well for every preference but not achieve the optimal control. In contrast, the multi-family representation in our method allows a much better Pareto set approximation. We discuss it more in Section 6.1.6.3. The META results on Humanoid-v2 are not reported, since in our experiments, META is not able to generate a Pareto front in the first quadrant.

In summary, none of the baseline algorithms distribute the computing resource to the RL tasks that best improve the Pareto quality. In contrast, by using the policy improvement prediction model, our algorithm is able to identify which regions on the Pareto are already near optimal and which regions can still be improved. Therefore the best tasks can be selected and a high-quality Pareto can be generated efficiently and effectively.

Pareto Analysis Results For each Pareto set solution computed by our evolutionary learning algorithm, we conduct the Pareto set analysis described in Section 6.1.5 to identify different families in the solution set. The family identification for Walker2d-v2 problem is illustrated in Figure 6-5. For Walker2d-v2, the whole Pareto set is split into four families in the parameter space. The Pareto front corresponding to each policy family comprise a continuous segment in the performance space.

We further construct a continuous Pareto representation for each family as described in Section 6.1.5. The constructed continuous Pareto front for Walker-v2 prob-

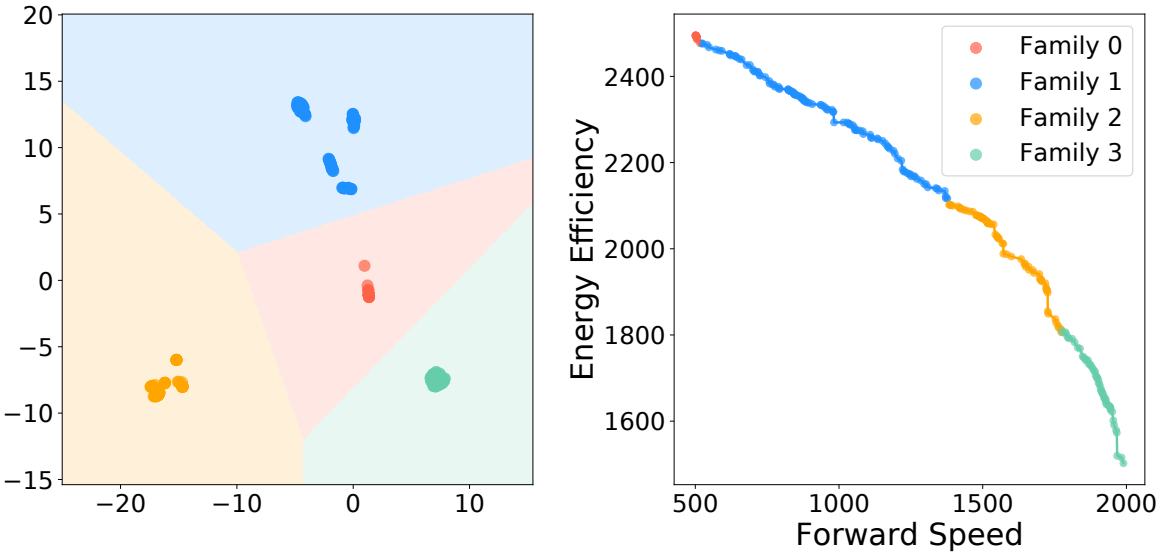


Figure 6-5: **Pareto analysis for Walker2d-v2 problem.** (Left) The policy families identified by t-SNE and k-means. (Right) Visualization of the families in objective space. The curve going through each family is the continuous Pareto approximation.

lem is shown in Figure 6-5 (right), and the continuous Pareto fronts for all benchmark problems is shown in Figure 6-11 and Figure 6-12.

To test the accuracy of the constructed continuous Pareto representation, we sample points on the continuous Pareto front for each family, and evaluate the relative error between the desired objectives and the objectives of the interpolated policy. For three-objective problem (Hopper-v3), we build a triangle mesh from the Pareto policies and then sample the testing points on the surfaces. The errors are reported in the first row of Table 6.2. The results show that the objectives of the interpolated policy is close enough to the desired objectives on the constructed Pareto front, which means by intra-family interpolation we can potentially get infinite number of policies on the Pareto front. We further test the necessity of the multi-family representation for the Pareto front. We sample points on the boundary of different families in performance space, and interpolate the policies from the different families and evaluate the relative error between the desired objectives and the objectives of the interpolated policy. The errors are reported in the second row of Table 6.2. The results show that it is impossible to get a policy with the desired objectives by interpolating the policies from different families, which further validate that the different families are

disjoint in the parameter space. The illustration of the tests is shown in Figure 6-6.

We also demonstrate in the supplementary video that the policies in different families show different behaviors (e.g., gait patterns), and such different behaviors help achieve the optimal control under the different objective preferences.

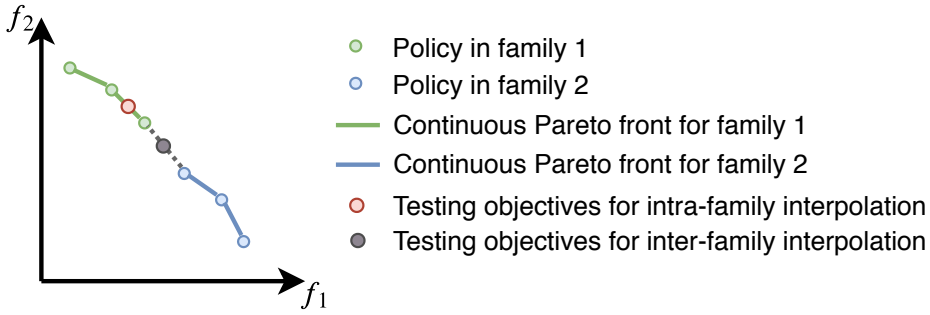
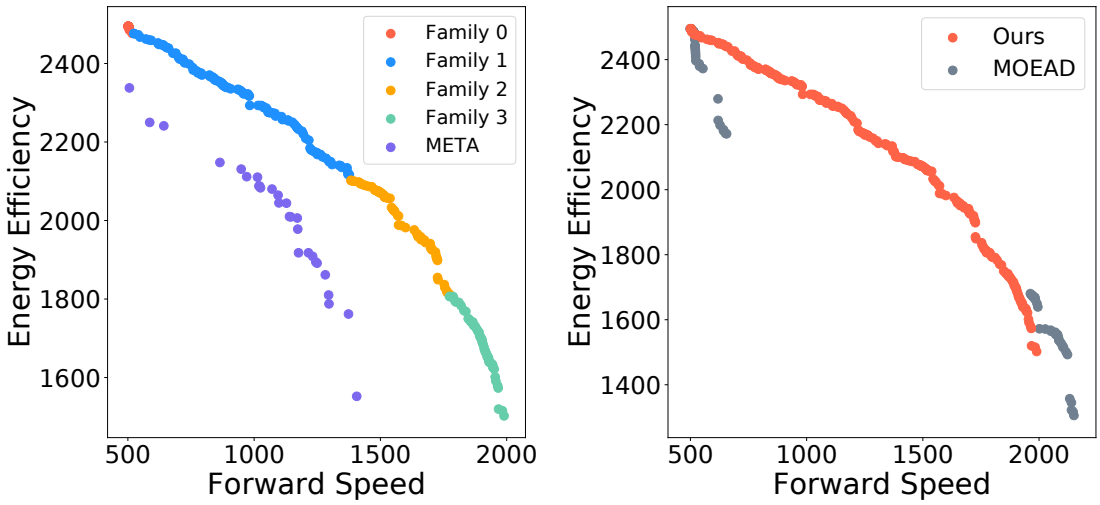


Figure 6-6: **Illustration for continuous Pareto front accuracy evaluation.** We sample testing objectives on the continuous Pareto front to test intra-family interpolation and on the boundary between families to test inter-family interpolation.

Table 6.2: **Intra-family and Inter-family interpolation errors.** We evaluate the relative error for intra-family and inter-family interpolation respectively. For two objective cases, we sample 1000 testing objectives for intra-family interpolation and 100 testing objectives for inter-family interpolation. For the three objective case the number of samples are 20000 and 5000 for intra-family and inter-family respectively. The average errors are reported below.

Example	HalfCheetah-v2	Hopper-v2	Swimmer-v2	Ant-v2	Walker2d-v2	Hopper-v3
Intra-Family	0.39%	0.85%	0.47%	3.94%	0.52%	0.87%
Inter-Family	6.71%	88.62%	2.81%	67.84%	7.95%	17.34%

Meta Policy or Multi-Family? By comparing META and our algorithm on Walker2d-v2 (Figure 6-7(a)), we empirically show that a typical Pareto set is composed from a set of disjoint policy families. Therefore, it is natural to compare this representation to meta policy. Meta policy method provides generalizability and represents the Pareto solutions by a single policy family. However, it sacrifices the optimality of the control. On the contrary, the multi-family representation can help achieve optimal control, but a hard switch between policy families is required while changing the preference on the boundary of each family.



(a) **META** v.s. **Ours** (b) **MOEA/D** v.s. **Ours**

Figure 6-7: **Pareto front comparison on Walker2d-v2 problem.** (a) **META** v.s. **Ours**. The multi-family representation in our method helps achieve better control for different preferences. (b) **MOEA/D** v.s. **Ours**. Our algorithm is unable to recover the Pareto on the bottom right corner due to the *long-term local minima problem*.

Failure Case As shown in Figure 6-7(b), in Walker2d-v2, although our algorithm can generate a much denser and higher-quality Pareto set than MOEA/D, we are unable to recover the bottom right corner of the Pareto front.

This problem is caused by the fact that in order to reach the bottom right corner of the Pareto front, the policy can be trapped in local minima for an extended time before moving towards a better solution. We call this the *long-term local minima problem*. Furthermore, as our algorithm predicts potential improvements based on the RL history data, it can fail to predict potential improvements for such policies. This is, indeed, a trade-off in our algorithm design: spending more time on the local minima area in order to reach better performance, or spending time on optimizing the other regions of the Pareto front.

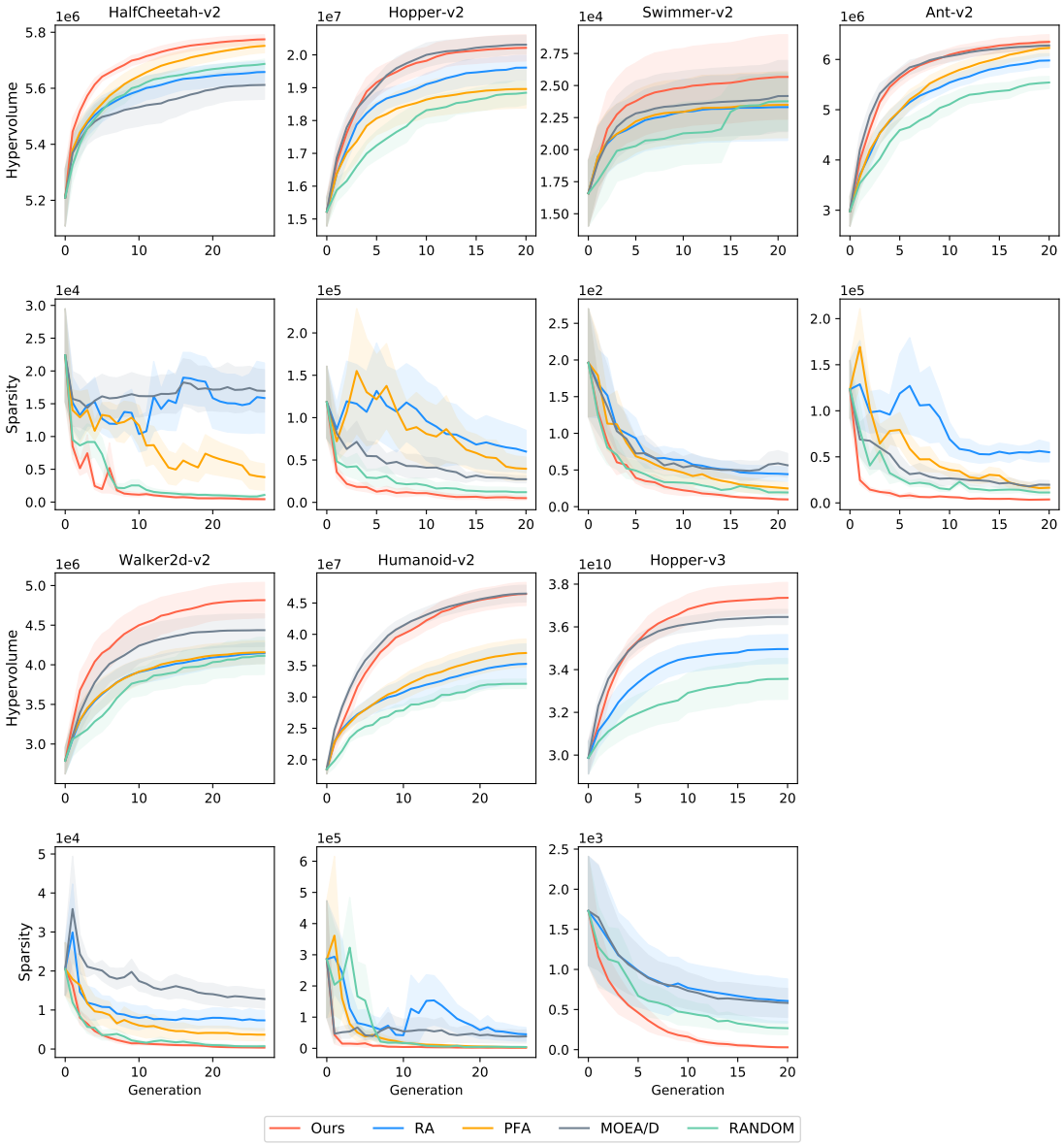


Figure 6-8: **The learning curves of hypervolume and sparsity metrics of different algorithms on all benchmark problems.** The x-axis is the generation, the y-axis is the metric and the shadow area is the standard deviation. We do not plot the learning curve of META because it can be measured only during the final adaptation stage. For Hopper-v3, we do not run PFA as the sequence of the weights in three dimensional space is undefined.

6.1.7 Summary

In this work, we show that an effective representation for obtaining the best performance trade-offs for multi-objective robot control is a Pareto set, which is composed from different policy families. We present an efficient algorithm to compute such

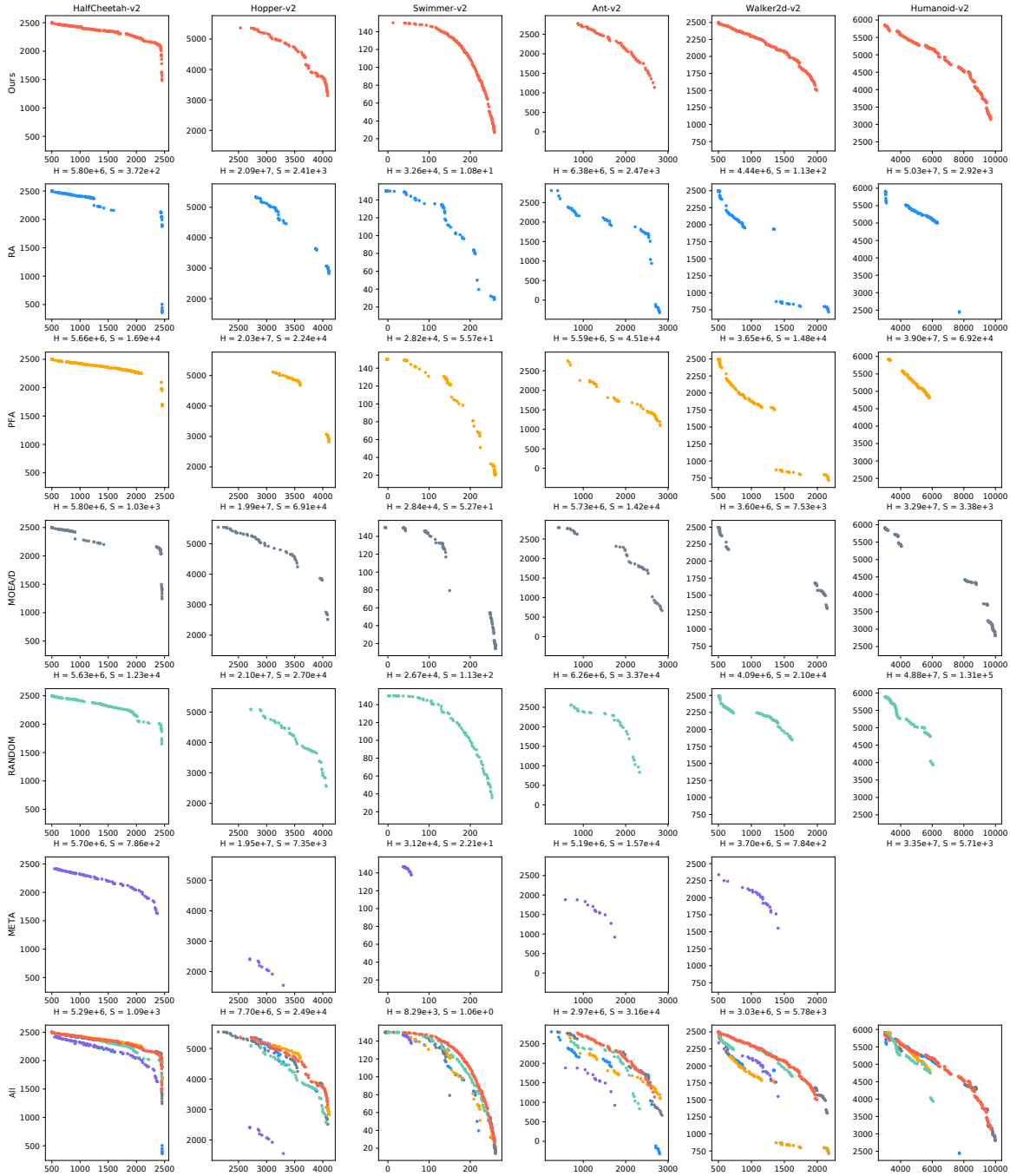


Figure 6-9: **The Pareto front approximation comparison for all 2-objective benchmark problems.** For each problem, we show the result for each algorithm with the same random seed. The Pareto of META on Humanoid-v2 problem is not plotted, since in our experiments, META is not able to generate a Pareto front in the first quadrant.

Pareto representations. A prediction-guided evolutionary learning algorithm is first employed to find a high-quality set of policies on the Pareto set. Then we conduct a Pareto analysis on the computed Pareto-optimal policies to construct a continuous

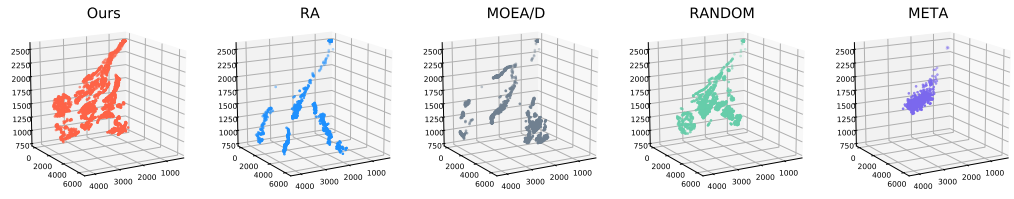


Figure 6-10: **The Pareto front approximation comparison for Hopper-v3.**

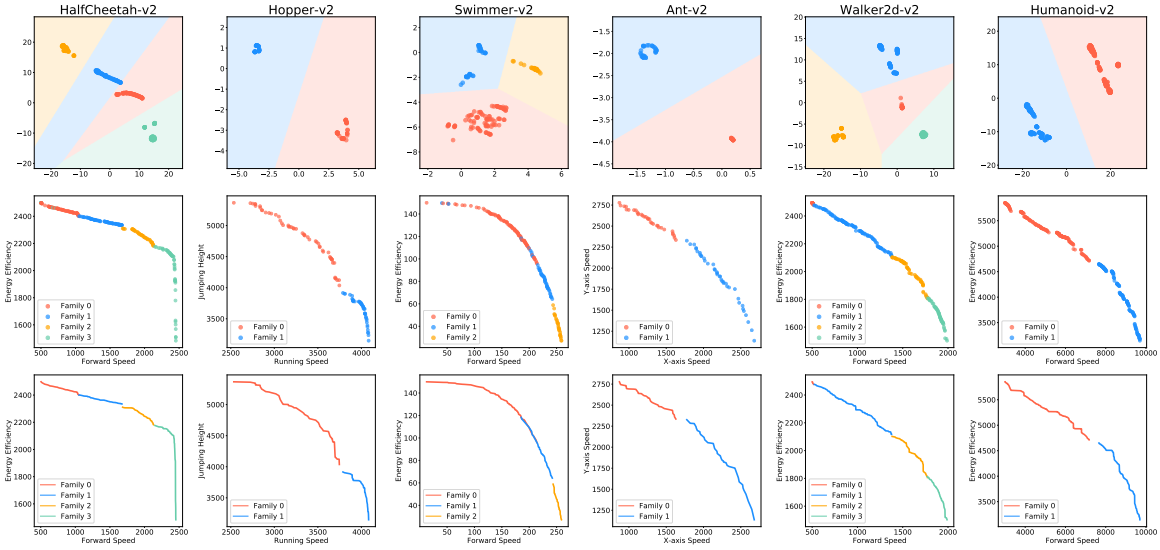


Figure 6-11: **Pareto analysis results for 2-objective benchmark problems.** The first row is the family identification in the parameter space by t-SNE and k-means. The second row is the corresponding objectives of those families in the performance space. The third row is the constructed continuous Pareto front approximation.

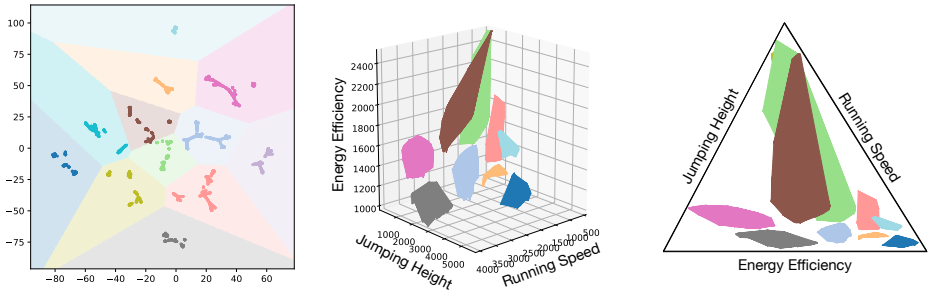


Figure 6-12: **Pareto analysis results for Hopper-v3.** (left) The family identification in the parameter space by t-SNE and k-means. (middle) The constructed continuous Pareto front approximation in the performance space. (right) Embedding the continuous Pareto front approximation in barycentric coordinates for better visualization.

Pareto representation. Furthermore, we design a set of multi-objective RL environments with continuous action space, and conduct extensive experiments to validate the effectiveness of our algorithm.

There are several directions which can be explored in the future. First, we believe that the learning efficiency could be further improved by sharing the sampled trajectories through the whole learning process. Second, it is desired to develop a more robust model to solve the *long-term local minima problem*. Finally, it is worthwhile to apply this method to solve multi-objective control problems for real-world robots.

6.2 MOGHS: Multi-Objective Robot Control and Shape Topology Co-Design

6.2.1 Motivation

Most physical tasks in the world, performed by humans or other animals, require being adept at multiple skills. For example, a lizard hunting prey may need to be proficient at climbing trees and running; a duck in migration needs to be able to both fly and swim; a human hurdler must be fast at running along bends and straightaways as well as jumping. If such animals were only capable of single motions, we would not expect them to be very successful.

We similarly should expect diverse adroitness from our robots. Yet, recent successes in algorithms which can co-design robots over morphology and control have been typically catered to singlular task specifications. For example, an algorithm may be able to co-design robots for forward running speed, energy efficient gaits, or climbing rough terrains, but not all three skills at once. In order to computationally develop robots capable of composite tasks rather than single repetitious motions, we require algorithms that simultaneously optimize over collections of requisite skills.

This work presents a method for multi-objective rigid robot co-design over both control and morphology (including discrete topology). Unlike much previous work on multi-objective co-design which only examined continuous parameters, we consider discrete topology as well as continuous control parameters. Because form informs function and vice versa, it is natural that different robot designs will be better at

different tasks. But, as in nature, rarely will a single design be best at all tasks. Thus, our goal is to extract robots with optimal trade-offs across different design objectives; *i.e.* the Pareto set of robot designs for the tasks at hand.

Our method builds upon RoboGrammar [6], a method that proposes a bio-inspired grammar for robot topological design and employs a learning-based morphological search over that design space. Robots are co-designed over topology and control; specifically, it employs a model predictive control (MPC) scheme. Complex grammars like that of RoboGrammar can be very expressive but typically yield large search spaces that are intractable to optimize over *via* naïve methods; the problem only becomes more difficult when multiple objectives are considered. The algorithm we present here is the only proposed method for solving this difficult and important multi-objective, topology/control co-design problem; thus making it novel in both problem scope and solution.

In this work, we contribute: *1)* Multi-objective co-design algorithms for finding Pareto-optimal robot topologies and controllers over challenging objective trade-offs, *2)* Demonstrations on combinations of terrestrial robot locomotion tasks, some with design restrictions, and *3)* Comparisons of our proposed methods benchmarked against baselines, demonstrating the power and importance of our techniques.

6.2.2 Overview

Our method builds upon Graph Heuristic Search algorithm introduced in Section 5.1. Graph Heuristic Search (GHS) chooses a grammar as a search space enables one to search over discrete operations that define robot topology; this provides for much more expressive designs than afforded by purely continuous parameters. We refer the reader to Section 5.1 for details regarding the GHS algorithm.

The remainder of this part is structured as follows. First, we introduce the scalarization approach to multi-objective optimization, and present two approaches to robot co-design which leverage this approach: a simple method based on solving a set

of standalone subproblems, and a more sophisticated and efficient Multi-Objective Graph Heuristic Search (MOGHS) that uses GHS to iteratively expand the Pareto front along different sampled directions in objective space, while learning a heuristic function shared across these directions. This shared heuristic is key to making our search efficient. Then, we present experiments demonstrating the efficacy of our methods compared to baselines, and conclude with possible extensions to this work.

6.2.3 A Naive Linear Scalarization Approach

Let $\boldsymbol{\mathcal{D}}$ define the space of valid designs (morphology and control sequence). We define a multi-objective function $\mathbf{F} : \boldsymbol{\mathcal{D}} \rightarrow \mathbb{R}^m$, such that for $d \in \boldsymbol{\mathcal{D}}$, $\mathbf{F}(d) = \big(f_1(d), f_2(d), ..., f_m(d)\big)$. Our goal is to generate designs which are Pareto optimal in the objective space; in other words, designs which are *non-dominated* by any other discovered design for any objective trade-off. In our maximization problem setting, a design d is said to be Pareto optimal if $d' \in \boldsymbol{\mathcal{D}}$ s.t. $: \forall i \;\; f_i(d') \geq f_i(d) \wedge \exists i$ s.t. $f_i(d') > f_i(d)$. In layman's terms, a design is considered Pareto optimal if there does not exist another design that is strictly better than this design at all objectives. We call the set of objective values of Pareto optimal designs the *Pareto front*.

In practice, computing the exact Pareto set/front is intractable for most hard problems; thus, we seek an algorithm which can find a Pareto approximation set that is as "good" as possible. We discuss quantitative metrics for evaluating the quality of a Pareto front in Sec. 6.2.7.5. However, we describe a qualitative way of determining the goodness of Pareto fronts here. Consider weight vectors $\boldsymbol{\omega} \in \mathbb{R}^m$ *s.t.* $\forall i \;\; \omega_i \geq 0$ and $\|\boldsymbol{\omega}\|_p = 1$ for some norm p. This can be thought of as the space of rays that sweep out radially in the first orthant. We wish to find a Pareto set such that, for every valid $\boldsymbol{\omega}$, there exists a point d in the Pareto set for which $\boldsymbol{\omega} \cdot \mathbf{F}(d)$ is large. In other words, for each (scalarization direction) ray $\boldsymbol{\omega}$ we want to find points whose objectives are far away from the origin along that ray.

Given this, an obvious strategy for multi-objective optimization arises, termed *scalarization* methods. For a large collection of weight vectors $\{\boldsymbol{\omega}^i\}_{i=1}^n$, a Pareto

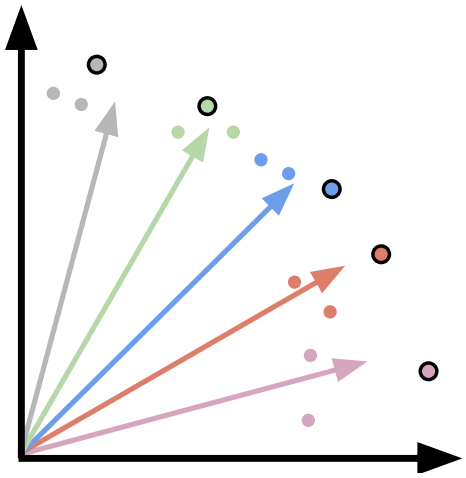


Figure 6-13: **A cartoon depicting the scalarization method.** Weight pairs form rays that project radially outward from the origin. Each circle represents a point that might be found during a single objective optimization using the weights defined by the ray of its color. Circles with black borders are the optimal solutions to the corresponding weights, which form a convex Pareto approximation front.

approximation set can be extracted by solving the set of optimization subproblems, where subproblem $\mathcal{P}_i$ is $\arg\max_{d \in \mathcal{D}} \boldsymbol{\omega}^i \cdot \mathbf{F}(d)$ (Fig. 6-13). This approach leads to two challenges. The first challenge is to find (as close to) the global maximum of each optimization subproblem. The second challenge is to have an efficient optimization scheme such that a dense set of weight vectors can be optimized.

As a first attempt at solving this problem, we propose the following (naïve) "discrete weights" strategy. Given a budget n, sample a uniformly spaced set $\{\boldsymbol{\omega}^i\}_{i=1}^n$ *a priori*. Then, for each weight vector $\boldsymbol{\omega}^i$, solve the n corresponding $\mathcal{P}_i$ independently, using the approach presented in [6] as a black box with reward $\boldsymbol{\omega}^i \cdot \mathbf{F}$. Such an approach can unfortunately have poor sample efficiency, especially in high-dimensional objective spaces, and treats problems with shared structure as decoupled. Thus, here we propose an alternative algorithm that is more effective at extracting good Pareto fronts.

6.2.4 Multi-objective Graph Heuristic Search

The discrete weights strategy has two shortcomings. The first is its decision to fix weights *a priori*. This makes it harder to find Pareto optimal points that do not lie

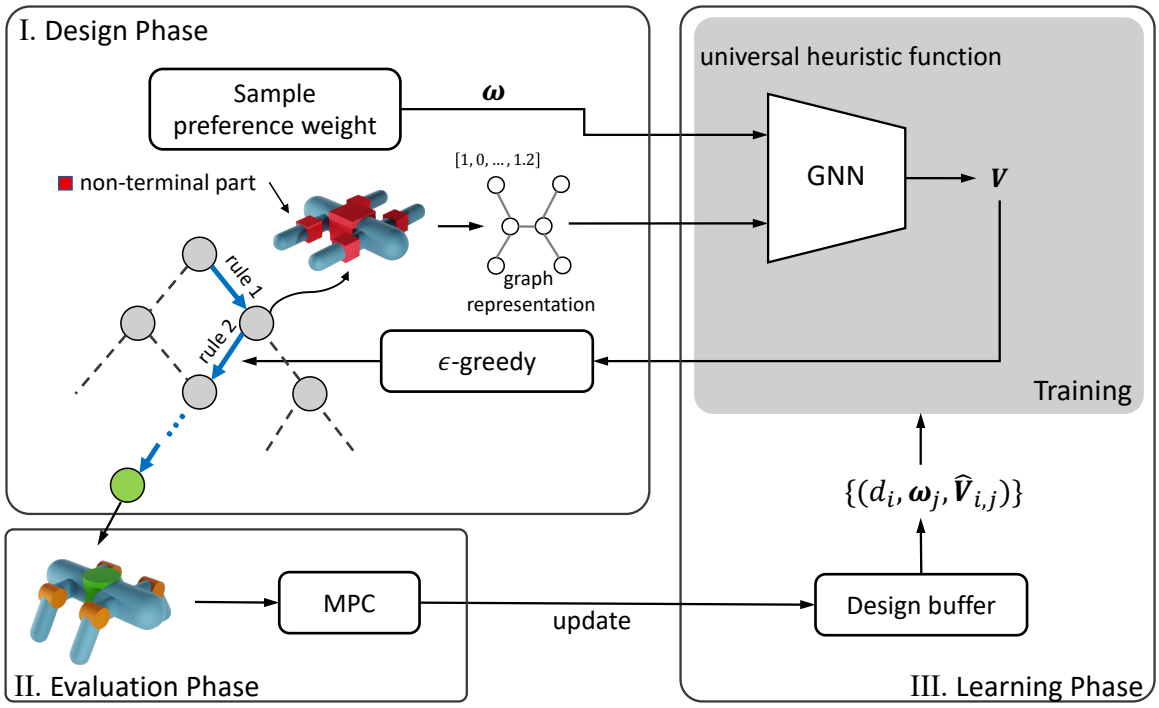


Figure 6-14: **Overview of the Multi-Objective Graph Heuristic Search (MOGHS).** In each episode, the algorithm conducts three phases (similar to GHS). *Design Phase*: A robot design is selected using a learned universal graph heuristic function along with a randomly picked preference weight $\boldsymbol{\omega}$. *Evaluation Phase*: The selected robot design is evaluated by MPC for each objective. *Learning Phase*: All the designs seen so far are leveraged to improve the heuristic.

along the sampled scalarization directions. The second shortcoming is the decision to treat each subproblem independently, despite the clear shared structure between the subproblems.

We thus improve the approach presented in RoboGrammar to efficiently search in multi-objective spaces. Our multi-objective optimization strategy makes the following two core changes. First, we sample weight vectors uniformly randomly at each episode of the algorithm both to search over the entire space of weight vectors in a *single* invocation of search algorithm, as well as to have a *dense* set of scalarization directions. Our second core innovation is to modify our learning model to be *multi-objective* and *universal* (named after its similarity in structure to universal control evaluators [98]), which we describe shortly. This improves search efficiency because the representations learned by the heuristic function can be shared by all weight combinations.

The process of MOGHS is visualized in Fig. 6-14, and the full algorithm can be found in Algorithm 6. Here, the original GHS is presented in black and teal as in Zhao et al. [6], with our new modifications highlighted in red. We now describe the major changes to the three phases of RoboGrammar's GHS to formulate our MOGHS.

6.2.4.1 Design Phase

In GHS, a designs is sampled in each episode according to a "double ϵ-greedy" approach in order to balance exploration and exploitation. First, K designs are sampled by ϵ-greedily choosing the next rule to apply at each step of the generation process, where the greedy choice is chosen by the one with the best heuristic function value. The final design can then be chosen greedily or rejected for a random one at an ϵ rate. In MOGHS, we instead first sample an $\boldsymbol{\omega}$ vector uniformly at random, a search direction for this episode of the design search. Using this, the greedy selection scheme is generalized to multiple objectives by the linear scalarization $\boldsymbol{\omega} \cdot \mathbf{V}(d, \boldsymbol{\omega})$.

Readers may note that the new greedy selection criterion requires two changes to the structure of the heuristic function. First, the heuristic must accept a weight vector $\boldsymbol{\omega}$ in addition to a design d as input, thus making it a *universal* predictor among all possible $\boldsymbol{\omega}$. Second, it outputs a vector of the predicted value of each objective function when evaluated by MPC, thus making it *multi-objective*. Thus, we design such a heuristic function $\mathbf{V} : \boldsymbol{\mathcal{D}} \times \mathbf{R}^m \to \mathbf{R}^m$.

6.2.4.2 Evaluation Phase

A candidate design is evaluated m times, in m separate invocations of MPC. The i^{th} invocation is run optimizing reward (objective) f_i. This returns m different optimized control sequences for the sampled design[3].

[3]If f_i does not depend on the robot's motion (*e.g.* the design complexity objective in section **??**), this process can be skipped for that objective.

Algorithm 6: Multi-Objective Graph Heuristic Search

Inputs: Number of iterations N, number of candidate designs K, Adam optimization steps opt_iter and batch size M, number of sampled weights N_w.
Output: A set of Pareto-Optimal designs P.
Initialize the universal graph neural network $\mathbf{V}_\theta(d, \boldsymbol{\omega})$.
Initialize the Pareto-Optimal design set $P \leftarrow \{\}$.
for episode $j \leftarrow 1$ **to** N **do**
  ▷ **Design Phase:** Generate a candidate design
  Sample a preference weight $\boldsymbol{\omega}$.
  $C \leftarrow \{\}$ ▷ Initialize possible design candidates
  ▷ Sample K designs by ϵ-greedy approach
  for $k \leftarrow 1$ **to** K **do**
    $d \leftarrow$ initial design graph
    while d has non-terminals **do**
      With probability ϵ select a random rule a, otherwise select $a = \arg\max_a \boldsymbol{\omega} \cdot \mathbf{V}_\theta(d', \boldsymbol{\omega})$, where d' is the robot design after applying rule a on design d.
      $d \leftarrow d'$
    end while
    Add possible candidate d to C.
  end for
  ▷ Choose one to be the candidate
  With probability ϵ select a random sampled design d from C, otherwise select $d = \arg\max_{d \in C} \boldsymbol{\omega} \cdot \mathbf{V}_\theta(d, \boldsymbol{\omega})$.
  ▷ **Evaluation Phase:** Compute the rewards for the design
  Run MPC in parallel for each task to evaluate the rewards vector $\vec{r}$ of design d.
  ▷ Update the design Pareto set
  Update P by d and $\vec{r}$.
  ▷ **Learning Phase:** train heuristic value function $\mathbf{V}_\theta$
  for $i \leftarrow 1$ **to** opt_iter **do**
    Sample a minibatch S_b of seen designs (partial or complete) of size M.
    Sample N_w preference weights $W = \{\boldsymbol{\omega}^j\}$.
    Compute target values for each $s \in S_b$ and $\boldsymbol{\omega} \in W$,

$$\hat{\mathbf{V}}(s, \boldsymbol{\omega}) = \arg\max_{d \in \text{descendant}(s)} \boldsymbol{\omega} \cdot \vec{r}(d)$$

    Update $\mathbf{V}_\theta(s, \boldsymbol{\omega})$ one step by Adam with the loss:

$$\sum_{s \in S_b, \boldsymbol{\omega} \in W} \|\mathbf{V}_\theta(s, \boldsymbol{\omega}) - \hat{\mathbf{V}}(s, \boldsymbol{\omega})\|^2$$

  end for
end for

6.2.4.3 Learning Phase

Given sampled batch of designs $S_b = \{d_i\}$ and weights $W = \{\omega^j\}$, we first compute the target heuristic value $\hat{\mathbf{V}}_{ij}$ for each design-weight combination. Then we train our universal heuristic function by using Adam [119] to minimize the following loss function:

$$\sum_{i,j} \|\mathbf{V}(d_i, \omega^j) - \hat{\mathbf{V}}_{ij}\|_2^2 \tag{6.6}$$

The target heuristic value $\hat{\mathbf{V}}$ is computed by maintaining a Pareto front of rewards for each partial design to keep track of the optimal terminal designs can be induced from it.

6.2.5 Universal Multi-Objective Heuristic Function

Our heuristic function is a graph neural network which maps robot morphologies to output vectors of predicted rewards. The morphology is represented as a graph, where each node corresponds to a link of the robot (and whose corresponding feature vectors describe their geometric and inertial properties), and each edge corresponds to a joint, thus encoding the topology. The architecture used is based on the differentiable pooling architecture presented by [120].

We wish for this network to operate on both a robot morphology as well as a weight vector as input; however, this architecture only handles graphs. In order to learn latent representations in the network which include the effects of the objective weights, we include $\boldsymbol{\omega}$ as additional features to each node in the graph. Finally, we modify the final linear operator of the network to output m channels instead of one, thus providing a multi-objective output space of the heuristic.

6.2.6 Other Improvements

We include the following further modifications which improve sample efficiency:

6.2.6.1 DAG-Based Target Updates

In [6], when a design is evaluated, its value is propagated up the derivation path to update the target value of the partial designs that generated it. Realizing the design rules actually form a directed acyclic graph (DAG) — each partial or complete design can have many rule sequences that generate it — we now perform the upward propagation procedure even on non-evaluated designs, which provides an opportunity to merge previously visited partial designs up newly discovered branches of the DAG. This simple refinement vastly improves sample efficiency by improving the accuracy of the computed target values.

6.2.6.2 Invalid Design Marking

Although the grammar is designed to avoid invalid designs, they still can occur. Once we know a design is invalid, however, we need not visit it again. We mark all invalid designs as such, and remove them from the pool of candidates to generate. If all designs that could be generated by a partial design are invalid, we further mark that partial design as invalid in an upward propagation scheme similar to the DAG-based update.

6.2.7 Experiments

6.2.7.1 Implementation

Despite MOGHS requiring many samples over design and control, the algorithm provides many opportunities for parallelization over CPU cores, thus keeping it practical. First, MOGHS samples many designs in parallel, which can be parallelized over many CPU cores. Second, the main bottleneck of MOGHS is the evaluation phase; fortunately, our MPC algorithm is sampling-based; this sampling procedure can also be parallelized over CPU cores to improve efficiency. Finally, the learning procedure can be accelerated by batching MN_W samples in parallel. Running 2000 episodes of

MOGHS takes approximately 20 hours on a 64-core Google Cloud machine, and the breakdown for each phase is around 3 hours (using 16 cores) for the design phase, 11 hours (using 64 cores) for the MPC evaluation phase, and 6 hours (using 1 core) for the learning phase.

6.2.7.2 Baseline Algorithms

We compare the Pareto fronts discovered by three algorithms: *1)* a random baseline, in which designs are sampled by randomly selecting rules until a terminal design is generated , *2)* The discrete weights method proposed in **??**, which is a discrete version of our MOGHS algorithm, and *3)* our MOGHS algorithm. We use the same total MPC evaluation budget (*i.e.* number of evaluated designs) for all three algorithms.

6.2.7.3 Task Specifications

We demonstrate our algorithm on six combinations of seven tasks, and compare each solution set qualitatively and quantitatively; please see the video for demonstrations of robots along the discovered Pareto fronts.

1) *Flat Terrain Locomotion*: In this task, the robot is rewarded for the forward running speed, and we assign additional reward to encourage stability in the forward direction.

2) *Low Power Flat Terrain Locomotion*: The same as the Flat Terrain task, except the maximum impulse of the motors is set to 20% of that normally available. This task highlights locomotion in scenarios when power must be conserved.

3) *Wall Terrain Locomotion*: Also the same as the Flat Terrain task, however, slalom-like walls are added to the terrain. Successful robots must run forward with the ability to move somewhat laterally to navigate terrain.

4) *Jumping*: In this task, the robot must jump as high as possible. The reward is set proportional to the height of the lowest part of the robot. As before, an additional reward is added to discourage the robot from falling over.

5) *Spin-In-Place*: This task tests the agility of the robot around the vertical axis. The reward is set proportional to the angular velocity of the robot around the vertical axis.

6) *Design Complexity*: The first of two tasks that is purely design-dependent (does not involve control), the reward is set inversely proportional to the number of links in the robot.

7) *Robot Height*: The second pure design task, the reward is set proportional to the height of the robot, with a penalty for changes in pitch, promoting tall, upright robots.

6.2.7.4 Experimental Setup

We run each experiment for 2000 episodes. For each task combination, we run each algorithm three times. In comparing metrics in Table 6.3, we compute the metric by averaging over all runs for that algorithm. For metrics that require a reference set, we take the union of all sampled designs of all runs of all algorithms, and compute its Pareto front. Hyperparameters used for the MOGHS algorithm are the same as GHS [6], with the preference weight minibatch size N_W set to 10. For the discrete weights algorithm, we sample 11 uniform weights in the two-objective cases (we did not consider this baseline in the three-objective case).

6.2.7.5 Results

We numerically evaluate the optimized Pareto fronts on three metrics, commonly used in the multi-objective optimization literature [141]: Hypervolume, Generational Distance, and Inverse Generational Distance. We present some Pareto fronts in Fig. 6-15, along with some selected designs. We encourage the reader to watch the video for more renderings of optimized Pareto fronts, and animations of the designs that populate them.

a) Hypervolume [142]: The hypervolume metric (HV) measures the hypervolume of the polytope defined by the space enclosed by the hyperplanes created by the axes

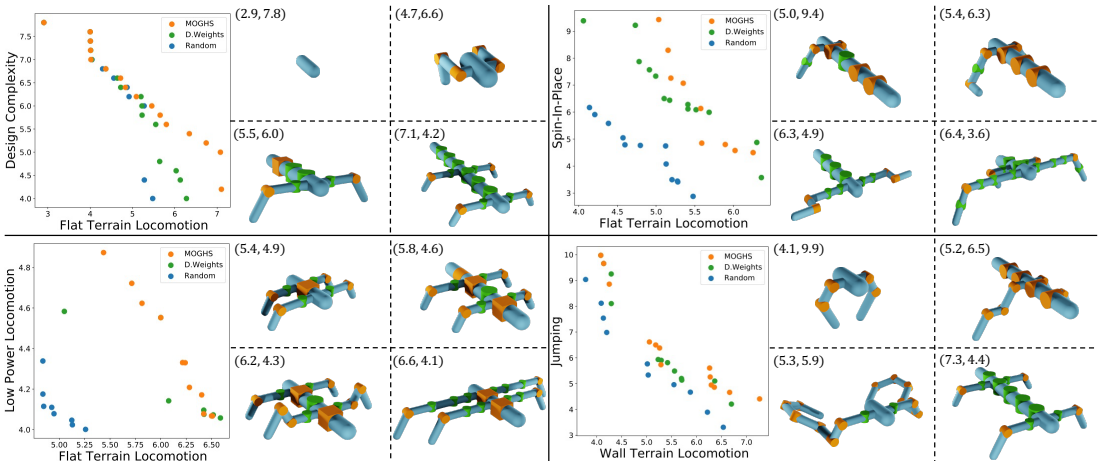


Figure 6-15: **Pareto front comparison of four of our two-objective experiments, and example designs from the Pareto front.** MOGHS produces more diverse and better performing results than discrete weights or the random baseline.

along the first orthant and the hull of the sampled points. As is visually evident, a larger HV is better.

b) Generational Distance [143]: The generational distance (GD) of a Pareto front P is defined as:

$$GD(P) = \frac{1}{|P|}(\sum_{\mathbf{p}\in P} \min_{\mathbf{r}\in R} d(\mathbf{p},\mathbf{r})^p)^{\frac{1}{p}} \tag{6.7}$$

where p is a norm (we choose 1), d is the Euclidean distance function, each $\mathbf{p}^i$ is a point in P, and the set R is a reference set generated by combining results from all optimization experiments. A smaller GD is better.

c) Inverse Generational Distance [144]: The inverse generational distance (IGD) is defined as:

$$IGD(P) = \frac{1}{|R|} \sum_{\mathbf{r}\in R} \min_{\mathbf{p}\in P} d(\mathbf{r},\mathbf{p}) \tag{6.8}$$

Again, a smaller IGD is better. Roughly speaking, GD measures the distance of all points on the captured Pareto front to the *best* known Pareto front, while IGD measures the distance of all points on the *best* known Pareto front to the captured Pareto front.

Numerical results are presented in Table 6.3. As can be seen in MOGHS dominates discrete weights in all task combinations across all metrics, often by significant margins. Discrete weights, in turn tends to beat the random baseline, but not as consistently. The consistency and quality of results returned by MOGHS emphasizes the importance of this method. Qualitatively, MOGHS's Pareto fronts, as seen in Fig. 6-15 tend to yield better performing objective trade-offs than other methods, while maintaining dense coverage.

The morphologies and motions of the designs found on the Pareto fronts of each problem are physically principled, but still interesting and exciting. We consider four of the two-objective trade-offs here. We leave the Design-Height problem, which serves as a benchmarking example, and the complex three-objective Flat-Jump-Spin task, which is better visualized animated, for the video. For example, in the Flat-Design task, a wide spectrum of robots from a single link (simplest design but no motion) to long, complex, fast, walkers are recovered. Along the way, various slower walkers with fewer links are found along the front, with increasingly dynamic motions. The Flat-LowPower task measures robots in various stages of power consumption. In the low power configuration, the robot is unable to balance if the legs are too wide, due to the increase gravitational torque on the torso. This leads to poor forward locomotion. However, longer legs lead to faster strides for the normal power state robots. The trade-off provides a spectrum of robots of varying width and compactness, trading off the importance of being effective at forward locomotion in the two power states. The Flat-Spin trade-off produces a very exciting and surprising result, as faster spinners trade off forward locomotion skill for an ability to spin in a top-like fashion. The fastest spinners possess motions that resemble breakdancing. The Wall-Jump robots meanwhile transition from maneuverable walkers to increasingly frog-like morphology with increased capacity to hop.

Table 6.3: **A comparison of the three numerical metrics among all three algorithms presented.** For each problem, metrics are presented in the following order: HV, GD, IGD. Bolded numbers mean that column's algorithm performed best for that algorithm/problem combination. MOGHS outperforms other methods in all metrics across all problems

Problem		Moghs	D. Weights	Random
Design-Height	Hv	**64.70**	52.43	51.60
	GD	**0.04**	0.45	0.29
	IGD	**0.28**	1.02	0.96
Flat-Design	Hv	**46.30**	42.25	39.11
	GD	**0.10**	0.29	0.31
	IGD	**0.19**	0.41	0.56
Flat-Spin	Hv	**49.94**	46.90	29.88
	GD	**0.34**	0.40	1.35
	IGD	**0.37**	0.50	1.79
Flat-LowPower	Hv	**29.94**	28.03	22.25
	GD	**0.04**	0.26	0.92
	IGD	**0.08**	0.28	1.07
Wall-Jump	Hv	**57.95**	54.09	44.11
	GD	**0.37**	0.73	1.23
	IGD	**0.36**	0.64	1.15
Flat-Jump-Spin	Hv	**307.68**	-	166.71
	GD	**0.22**	-	1.09
	IGD	**0.38**	-	1.69

6.2.8 Conclusion

We have demonstrated methods for co-designing robots over morphology and control over multiple objectives. Our multi-objective graph heuristic search is first of its kind, and extracts far superior Pareto fronts with higher efficiency than more naïve methods. The tasks demonstrated, including running, jumping, spinning, and obstacle navigation have direct practical value in real-world terrestrial agile robots.

There remain important avenues for future research. First, all examples demonstrated in this paper were tested in simulation. A study fabricating these designs and demonstrating their physical accuracy would be valuable. Second, all examples presented in this paper were for two or three objective trade-offs. It would be interesting to see if this algorithm would scale to higher objective spaces. Finally, although we have demonstrated our algorithm for robotics, most of our method should be general

to any grammar-defined domain, excepting the evaluation (control) procedure and choice of heuristic architecture. Extensions of our algorithm to other domains would demonstrate further value of our approach.

Chapter 7

Conclusion and Outlook

Hardware design (body) and software control (brain) play equally important roles on a robot's task performance. However, co-designing the body and brain of a robot in the real world is still a challenging problem due to the complicated coupling of robot hardware design, control algorithm, and manufacturing constraints. Today, the process of improving robots for the given tasks still requires tons of slow and labor-intensive iterations. A fully automated computational robot design pipeline can revolutionize this procedure via largely speeding up the iteration speed and potentially discovering novel and better robot designs beyond human prior.

We envision that in the future, an ultimate computational robot design pipeline is supposed to optimize all the robot hardware-related parameters and control policy simultaneously for physically realizable robots and be able to find a set of Pareto-optimal solutions for multiple objectives. The difficulties of constructing a feature-complete computational robot design pipeline come from several key problems, including but not limited to robot design and design space representation, performance evaluation, and robot optimization. In this thesis, we argue:

1) **Having good robot design and design space representation is crucial.** A good robot representation should be able to express tons of designs with a compact set of parameters, to produce valid and manufacturable designs only, and to be easy to be optimized. To achieve so, in Chapter 3, we present a graph

grammar representation for discrete robot topology design and a deformation-based representation for continuous robot morphology design. Our graph grammar representation converts the robot topology design problem into a sequential decision making problem, while our deformation-based representation enables numerical optimization with box constraints only for the robot morphology and control co-optimization.

2) **Having an efficient and effective simulator is important.** We argue that a good simulator should be fast and differentiable. This is because the simulation serves as the most inner loop of a computational robot design framework and is usually the bottle neck of the whole algorithm's running time. On the other hand, the differentiability of the simulator can offer the optimizer the ability to actively exploit the underlining dynamics phyiscs model of the robot and the task. Therefore, in Chapter 4 and Chapter 5, we develop differentiable articulate rigid body simulator to offer the analytical gradients of the task execution, and also leverage the GPU-parallelization to speed up the simulator.

3) **The way to optimize a robot design is critical.** A lesson from this thesis is that while numerical gradient-based optimizer can improve the optimization efficiency of continuous robot variables, learning-based approaches might be necessary when more complex or discrete variables are involved. Specifically, in Chapter 5, we present our numerical optimization based approach to co-optimize the robot continuous morphology and control, and present a learning-based algorithm to co-optimize the robot discrete topology along with the control strategy.

4) **Multi-objective optimization is necessary when considering more realistic problems.** Motivated by this, in Chapter 6, we introduce our efforts on multi-objective control policy optimization and multi-objective robot co-design problems.

We hope that this thesis can contribute a significant step towards this ultimate goal of a feature-complete computational robot design pipeline and provide useful

guidance to future research. To conclude this thesis, below, we discuss the limitations of this thesis, and point out a few future research directions.

Joint optimization of all shape parameters and control policy Each of the technique we introduced in this thesis solves a subsystem of the whole pipeline while sacrificing the other aspects. For terrestrial robots, we optimize the discrete shape topology and compute the Pareto-Optimal solutions for multiple objectives, while keeping the continuous morphology parameters unchangeable and parameterizing the control as an open-loop trajectory. For manipulator robots, we propose a manufacturing-aware representation for the continuous shape morphology and optimize the continuous shape parameters, while assuming the discrete topology to be fixed, and the control to be open-loop sequence. Although we present a viable and hybrid representation for both continuous and discrete shape parameters, we have just demonstrate its effectiveness through a user interface instead of integrating it into an automatic co-design optimization. For the tactile control algorithm, we optimize for a neural network control policy while not considering any modifications on the robot shape variables. A very first future step towards the envisioned ideal system is to jointly optimize all the shape parameters including both the continuous ones and the discrete ones, as well as a sophisticated control policy such as a neural network. We have some preliminary exploration of jointly optimizing both the continuous shape parameters and discrete shape parameters for Unmanned Aerial Vehicle [145] and Unmanned Underwater Vehicle [146] by leveraging the hybrid shape representation we introduced in Section 3.3. However, the robots in those preliminary attempts are just relatively simple single rigid bodies with no contact to the environment, and the control strategy are just classical LQR controls around a stationary trim state. Exploring more sophisticated control representations and more complicated contact-rich robot tasks is an important and immediate future step.

Many-objective robot design We have discussed some optimization algorithms for multi-objective control problems in Chapter 6. However, so far in this thesis,

we have only solved the problems with two or three objectives. Many problems in real life can indeed evaluated by more than three objectives, which we call it *many-objective* problems. The size of the solution set increases incredibly as the number of objectives increases. For single-objective problem, there is a single optimal solution exists. When the number of objectives are two, the Pareto front is a 1-D frontier curve. In the work discussed in Section 6.1, the number of policies on Pareto front we computed for two-objective problems are typically three to four hundreds. When the number of objectives increases to three, the Pareto front becomes a 2d surface. As an illustration, for the three-objective task *Hopper-v3* in Section 6.1, the number of Pareto policies is thousands. The size of the Pareto set scales exponentially if we further increase the number of the objectives thus naively computing every single solution in the Pareto set as we have done so far becomes infeasible in terms of both memory cost and computation cost. Therefore, it requires a more effective Pareto solution set representation and a more efficient computational algorithm for many-objective robot problems.

Sensor and actuator optimization In our discussed works, we assume we have full access to the robot dynamics state, such as positions and velocities. However, that is rarely the case in a robotic system in real since the real observation input to a robotic system highly depends on its sensor types and placements. When we design the hardware of a robot, not only should we design the robot's shape, but the sensor design is also a critical component of robot hardware design. For different tasks, we may prefer different types of sensors such as tactile sensors, vision sensors or accelerometers, and different choices of sensors may have difference task-optimal placements for the sensors. For examples, for a manipulation task, we may want a vision sensor which is mounted at the head of a robot while we may also want some tactile sensors installed at the manipulator fingertips to sense its contact to the environment. Furthermore, the sensor design also has sophisticated effect on the optimal design of the robot shape and control. Thus how to jointly optimize the shape, control as well as sensor can be an interesting direction to explore.

While the sensors are responsible for sensing the surroundings and providing the observation input to the robotics system, actuator plays the role to process the control output of the robotic system and convert the control signal to its real effect into the environment. So far throughout this thesis, we use simple rigid joint actuators to control the robots. However, adopting and optimizing the structure of other actuator types such as soft compliant joint and tendon-driven actuators can have its potential to further improve the task performance on real robot.

Sim-to-real transfer Eventually, we would like to put the optimized robot hardware into the real via a manufacturing process and deploy the optimized control algorithm on the real system. To achieve so, we have two sim-to-real requirements to be satisfied. First, the *hardware* sim-to-real, which requires us to constrain the optimization search space to be always manufacturing valid. We have demonstrated this part in our work. We apply graph grammar to constrain the search space of discrete shape topology within valid designs, and propose deformation-based representation with *connectivity* constraints and *fabrication* constraints for the continuous morphology parameters. Second, the *control* sim-to-real, which may come from the dynamics mismatch between real physics and simulation physics, some intricate dynamics effects that cannot be easily modeled in simulation, as well as the robot modeling mismatch between the robots in real and in simulation. Although there has been a lot of effort in the robotics field to narrow down this control sim-to-real gap, and we also have some preliminary attempts on sim-to-real for stationary tactile-based control [4] and for hybrid UAV control [52], the current techniques are still highly problem specific, far from satisfying the needs of a general-purpose computational robot design pipeline.

Bibliography

[1] Seunghwan Lee, Moonseok Park, Kyoungmin Lee, and Jehee Lee. Scalable muscle-actuated human simulation and control. *ACM Trans. Graph.*, 38(4), July 2019.

[2] Jie Xu, Tao Chen, Lara Zlokapa, Michael Foshey, Wojciech Matusik, Shinjiro Sueda, and Pulkit Agrawal. An End-to-End Differentiable Framework for Contact-Aware Robot Design. In *Proceedings of Robotics: Science and Systems*, Virtual, July 2021. doi: 10.15607/RSS.2021.XVII.008.

[3] Lara Zlokapa, Yiyue Luo, Jie Xu, Michael Foshey, Kui Wu, Pulkit Agrawal, and Wojciech Matusik. An Integrated Design Pipeline for Tactile Sensing Robotic Manipulators. In *IEEE International Conference on Robotics and Automation*, Philadelphia, PA, May 2022.

[4] Jie Xu, Sangwoon Kim, Tao Chen, Alberto Rodriguez Garcia, Pulkit Agrawal, Wojciech Matusik, and Shinjiro Sueda. Efficient Tactile Simulation with Differentiability for Robotic Manipulation. In Submission to CoRL 2022.

[5] Jie Xu, Viktor Makoviychuk, Yashraj Narang, Fabio Ramos, Wojciech Matusik, Animesh Garg, and Miles Macklin. Accelerated policy learning with parallel differentiable simulation. In *International Conference on Learning Representations*, 2021.

[6] Allan Zhao, Jie Xu, Mina Konaković-Luković, Josephine Hughes, Andrew Spielberg, Daniela Rus, and Wojciech Matusik. Robogrammar: graph grammar for terrain-optimized robot design. *ACM Transactions on Graphics (TOG)*, 39(6): 1–16, 2020.

[7] Jie Xu, Yunsheng Tian, Pingchuan Ma, Daniela Rus, Shinjiro Sueda, and Wojciech Matusik. Prediction-guided multi-objective reinforcement learning for continuous robot control. In *Proceedings of the 37th International Conference on Machine Learning*, 2020.

[8] Jie Xu, Andrew Speilberg, Allan Zhao, Daniela Rus, and Wojciech Matusik. Multi-objective graph heuristic search for terrestrial robot design. IEEE, 2021.

[9] Agrim Gupta, Silvio Savarese, Surya Ganguli, and Li Fei-Fei. Embodied intelligence via learning and evolution. *arXiv preprint arXiv:2102.02202*, 2021.

[10] Karl Sims. Evolving virtual creatures. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 15–22. ACM, 1994.

[11] Tingwu Wang, Yuhao Zhou, Sanja Fidler, and Jimmy Ba. Neural graph evolution: Towards efficient automatic robot design. *arXiv preprint arXiv:1906.05370*, 2019.

[12] Sehoon Ha, Stelian Coros, Alexander Alspach, Joohyung Kim, and Katsu Yamane. Joint optimization of robot design and motion parameters using the implicit function theorem. In *Robotics: Science and systems*, volume 8, 2017.

[13] Christopher Hazard, Nancy Pollard, and Stelian Coros. Automated design of manipulators for in-hand tasks. In *2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)*, pages 1–8. IEEE, 2018.

[14] Raphael Deimel, Patrick Irmisch, Vincent Wall, and Oliver Brock. Automated co-design of soft hand morphology and control strategy for grasping. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1213–1218, 2017. doi: 10.1109/IROS.2017.8202294.

[15] KM Digumarti, C Gehring, S Coros, J Hwangbo, and R Siegwart. Concurrent optimization of mechanical design and locomotion control of a legged robot. In *17th International Conference on Climbing and Walking Robots (CLAWAR)*, pages 315–+. WORLD SCIENTIFIC PUBL CO PTE LTD, 2014.

[16] Andre Meixner, Christopher Hazard, and Nancy Pollard. Automated design of simple and robust manipulators for dexterous in-hand manipulation tasks using evolutionary strategies. In *2019 IEEE-RAS 19th International Conference on Humanoid Robots (Humanoids)*, pages 281–288. IEEE, 2019.

[17] Xinlei Pan, Animesh Garg, Animashree Anandkumar, and Yuke Zhu. Emergent hand morphology and control from optimizing robust grasps of diverse objects. *arXiv preprint arXiv:2012.12209*, 2020.

[18] Kevin Wampler and Zoran Popović. Optimal gait and form for animal locomotion. *ACM Transactions on Graphics (TOG)*, 28(3):1–8, 2009.

[19] Tianjian Chen, Zhanpeng He, and Matei Ciocarlie. Hardware as policy: Mechanical and computational co-optimization using deep reinforcement learning. *arXiv preprint arXiv:2008.04460*, 2020.

[20] Kevin Sebastian Luck, Heni Ben Amor, and Roberto Calandra. Data-efficient co-adaptation of morphology and behaviour with deep reinforcement learning. In *Conference on Robot Learning*, pages 854–869. PMLR, 2020.

[21] Charles Schaff, David Yunis, Ayan Chakrabarti, and Matthew R Walter. Jointly learning to construct and control agents using deep reinforcement learning.

In *2019 International Conference on Robotics and Automation (ICRA)*, pages 9798–9805. IEEE, 2019.

[22] Adriana Schulz, Jie Xu, Bo Zhu, Changxi Zheng, Eitan Grinspun, and Wojciech Matusik. Interactive design space exploration and optimization for cad models. *ACM Transactions on Graphics (TOG)*, 36(4):1–14, 2017.

[23] Christian Hafner, Christian Schumacher, Espen Knoop, Thomas Auzinger, Bernd Bickel, and Moritz Bächer. X-cad: Optimizing cad models with extended finite elements. *ACM Trans. Graph.*, 38(6), November 2019. ISSN 0730-0301. doi: 10.1145/3355089.3356576. URL `https://doi.org/10.1145/3355089.3356576`.

[24] Huy Ha, Shubham Agrawal, and Shuran Song. Fit2form: 3d generative model for robot gripper form design. *arXiv preprint arXiv:2011.06498*, 2020.

[25] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[26] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. 2016.

[27] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

[28] Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, and Koray kavukcuoglu. Interaction networks for learning about objects, relations and physics. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pages 4509–4517, 2016.

[29] Michael B Chang, Tomer Ullman, Antonio Torralba, and Joshua B Tenenbaum. A compositional object-based approach to learning physical dynamics. *arXiv preprint arXiv:1612.00341*, 2016.

[30] Yunzhu Li, Jiajun Wu, Russ Tedrake, Joshua B Tenenbaum, and Antonio Torralba. Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids. *arXiv preprint arXiv:1810.01566*, 2018.

[31] Damian Mrowca, Chengxu Zhuang, Elias Wang, Nick Haber, Li Fei-Fei, Joshua B Tenenbaum, and Daniel LK Yamins. Flexible neural representation for physics prediction. *arXiv preprint arXiv:1806.08047*, 2018.

[32] Brandon Amos and J. Zico Kolter. Optnet: Differentiable optimization as a layer in neural networks, 2019.

[33] Filipe de Avila Belbute-Peres, Kevin Smith, Kelsey Allen, Josh Tenenbaum, and J Zico Kolter. End-to-end differentiable physics for learning and control. *Advances in neural information processing systems*, 31:7178–7189, 2018.

[34] Jonas Degrave, Michiel Hermans, Joni Dambre, et al. A differentiable physics engine for deep learning in robotics. *Frontiers in neurorobotics*, 13:6, 2019.

[35] Moritz Geilinger, David Hahn, Jonas Zehnder, Moritz Bächer, Bernhard Thomaszewski, and Stelian Coros. Add: analytically differentiable dynamics for multi-body systems with frictional contact. *ACM Transactions on Graphics (TOG)*, 39(6):1–15, 2020.

[36] Eric Heiden, David Millard, Hejia Zhang, and Gaurav S Sukhatme. Interactive differentiable simulation. *arXiv preprint arXiv:1905.10706*, 2019.

[37] Ying Wang, Nicholas J. Weidner, Margaret A. Baxter, Yura Hwang, Danny M. Kaufman, and Shinjiro Sueda. REDMAX: Efficient & flexible approach for articulated dynamics. *ACM Trans. Graph.*, 38(4), July 2019. ISSN 0730-0301. doi: 10.1145/3306346.3322952. URL https://doi.org/10.1145/3306346.3322952.

[38] Tao Du, Kui Wu, Pingchuan Ma, Sebastien Wah, Andrew Spielberg, Daniela Rus, and Wojciech Matusik. Diffpd: Differentiable projective dynamics with contact. *arXiv preprint arXiv:2101.05917*, 2021.

[39] David Hahn, Pol Banzet, James M Bern, and Stelian Coros. Real2sim: Viscoelastic parameter estimation from dynamic motion. *ACM Transactions on Graphics (TOG)*, 38(6):1–13, 2019.

[40] Yuanming Hu, Jiancheng Liu, Andrew Spielberg, Joshua B Tenenbaum, William T Freeman, Jiajun Wu, Daniela Rus, and Wojciech Matusik. Chainqueen: A real-time differentiable physical simulator for soft robotics. In *2019 International conference on robotics and automation (ICRA)*, pages 6265–6271. IEEE, 2019.

[41] Yuanming Hu, Luke Anderson, Tzu-Mao Li, Qi Sun, Nathan Carr, Jonathan Ragan-Kelley, and Frédo Durand. Difftaichi: Differentiable programming for physical simulation. In *International Conference on Learning Representations*, 2020.

[42] Zhiao Huang, Yuanming Hu, Tao Du, Siyuan Zhou, Hao Su, Joshua B. Tenenbaum, and Chuang Gan. Plasticinelab: A soft-body manipulation benchmark with differentiable physics. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=xCcdBRQEDW.

[43] Krishna Murthy Jatavallabhula, Miles Macklin, Florian Golemo, Vikram Voleti, Linda Petrini, Martin Weiss, Breandan Considine, Jerome Parent-Levesque, Kevin Xie, Kenny Erleben, et al. gradsim: Differentiable simulation for system identification and visuomotor control. *arXiv preprint arXiv:2104.02646*, 2021.

[44] Junbang Liang, Ming Lin, and Vladlen Koltun. Differentiable cloth simulation for inverse problems. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/28f0b864598a1291557bed248a998d4e-Paper.pdf.

[45] Yi-Ling Qiao, Junbang Liang, Vladlen Koltun, and Ming Lin. Scalable differentiable physics for learning and control. In *International Conference on Machine Learning*, pages 7847–7856. PMLR, 2020.

[46] Stelian Coros, Andrej Karpathy, Ben Jones, Lionel Reveret, and Michiel van de Panne. Locomotion skills for simulated quadrupeds. *ACM Trans. Graph.*, 30 (4), jul 2011. ISSN 0730-0301. doi: 10.1145/2010324.1964954. URL https://doi.org/10.1145/2010324.1964954.

[47] Igor Mordatch, Emanuel Todorov, and Zoran Popović. Discovery of complex behaviors through contact-invariant optimization. *ACM Transactions on Graphics (TOG)*, 31(4):1–8, 2012.

[48] Michael Posa, Cecilia Cantu, and Russ Tedrake. A direct method for trajectory optimization of rigid bodies through contact. *The International Journal of Robotics Research*, 33(1):69–81, 2014.

[49] John T Betts. Survey of numerical methods for trajectory optimization. *Journal of guidance, control, and dynamics*, 21(2):193–207, 1998.

[50] Jemin Hwangbo, Joonho Lee, Alexey Dosovitskiy, Dario Bellicoso, Vassilios Tsounis, Vladlen Koltun, and Marco Hutter. Learning agile and dynamic motor skills for legged robots. *Science Robotics*, 4(26), 2019.

[51] 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, 2019.

[52] Jie Xu, Tao Du, Michael Foshey, Beichen Li, Bo Zhu, Adriana Schulz, and Wojciech Matusik. Learning to fly: computational controller design for hybrid uavs with reinforcement learning. *ACM Transactions on Graphics (TOG)*, 38 (4):1–12, 2019.

[53] Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning quadrupedal locomotion over challenging terrain. *Science robotics*, 5(47), 2020.

[54] 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. *The International Journal of Robotics Research*, 39(1):3–20, 2020.

[55] Tao Chen, Jie Xu, and Pulkit Agrawal. A system for general in-hand object re-orientation. In *5th Annual Conference on Robot Learning*, 2021.

[56] Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel van de Panne. Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Trans. Graph.*, 37(4):143:1–143:14, July 2018. ISSN 0730-0301.

[57] Xue Bin Peng, Ze Ma, Pieter Abbeel, Sergey Levine, and Angjoo Kanazawa. Amp: Adversarial motion priors for stylized physics-based character control. *ACM Trans. Graph.*, 40(4), July 2021.

[58] Libin Liu and Jessica Hodgins. Learning basketball dribbling skills using trajectory optimization and deep reinforcement learning. *ACM Trans. Graph.*, 37 (4), July 2018.

[59] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

[60] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[61] 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 *ICLR (Poster)*, 2016.

[62] 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 Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1928–1937, New York, New York, USA, 20–22 Jun 2016. PMLR.

[63] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pages 1587–1596. PMLR, 2018.

[64] 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*, pages 1861–1870. PMLR, 2018.

[65] Thanard Kurutach, Ignasi Clavera, Yan Duan, Aviv Tamar, and Pieter Abbeel. Model-ensemble trust-region policy optimization. In *International Conference on Learning Representations*, 2018.

[66] Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: Model-based policy optimization. *Advances in Neural Information Processing Systems*, 32:12519–12530, 2019.

[67] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. In *International Conference on Learning Representations*, 2019.

[68] Ignasi Clavera, Yao Fu, and Pieter Abbeel. Model-augmented actor-critic: Backpropagating through paths. In *International Conference on Learning Representations*, 2020.

[69] Michael Mozer. A focused backpropagation algorithm for temporal pattern recognition. *Complex Systems*, 3, 01 1995.

[70] Yi-Ling Qiao, Junbang Liang, Vladlen Koltun, and Ming C. Lin. Efficient differentiable simulation of articulated bodies. In *ICML*, 2021.

[71] Miguel Angel Zamora Mora, Momchil Peychev, Sehoon Ha, Martin Vechev, and Stelian Coros. Pods: Policy optimization via differentiable simulation. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 7805–7817. PMLR, 18–24 Jul 2021.

[72] Gregory S Hornby, Hod Lipson, and Jordan B Pollack. Generative representations for the automated design of modular physical robots. *IEEE transactions on Robotics and Automation*, 19(4):703–719, 2003.

[73] Nick Cheney, Robert MacCurdy, Jeff Clune, and Hod Lipson. Unshackling evolution: evolving soft robots with multiple materials and a powerful generative encoding. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, pages 167–174, 2013.

[74] Nicholas Cheney, Jeff Clune, and Hod Lipson. Evolved electrophysiological soft robots. In *Artificial Life Conference Proceedings 14*, pages 222–229. MIT Press, 2014.

[75] Francesco Corucci, Nick Cheney, Hod Lipson, Cecilia Laschi, and Josh Bongard. Evolving swimming soft-bodied creatures. In *ALIFE XV, The Fifteenth International Conference on the Synthesis and Simulation of Living Systems, Late Breaking Proceedings*, volume 6, 2016.

[76] Sehoon Ha, Stelian Coros, Alexander Alspach, James M Bern, Joohyung Kim, and Katsu Yamane. Computational design of robotic devices from high-level motion specifications. *IEEE Transactions on Robotics*, 34(5):1240–1251, 2018.

[77] Andrew Spielberg, Brandon Araki, Cynthia Sung, Russ Tedrake, and Daniela Rus. Functional co-optimization of articulated robots. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5035–5042. IEEE, 2017.

[78] Moritz Geilinger, Roi Poranne, Ruta Desai, Bernhard Thomaszewski, and Stelian Coros. Skaterbots: Optimization-based design and motion synthesis for robotic creatures with legs and wheels. *ACM Transactions on Graphics (TOG)*, 37(4):1–12, 2018.

[79] MF Ashby. Multi-objective optimization in material design and selection. *Acta materialia*, 48(1):359–369, 2000.

[80] Xingtao Liao, Qing Li, Xujing Yang, Weigang Zhang, and Wei Li. Multiobjective optimization for crash safety design of vehicles using stepwise regression model. *Structural and multidisciplinary optimization*, 35(6):561–569, 2008.

[81] Salim Fettaka, Jules Thibault, and Yash Gupta. Design of shell-and-tube heat exchangers using multiobjective optimization. *International Journal of Heat and Mass Transfer*, 60:343–354, 2013.

[82] Christos A Nicolaou and Nathan Brown. Multi-objective optimization methods in drug design. *Drug Discovery Today: Technologies*, 10(3):e427–e435, 2013.

[83] Kalyanmoy Deb. *Multi-objective optimization using evolutionary algorithms*, volume 16. John Wiley & Sons, 2001.

[84] Kalyanmoy Deb, Samir Agrawal, Amrit Pratap, and Tanaka Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii. In *International conference on parallel problem solving from nature*, pages 849–858. Springer, 2000.

[85] Kenneth O Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.

[86] Christian Igel, Nikolaus Hansen, and Stefan Roth. Covariance matrix adaptation for multi-objective optimization. *Evolutionary computation*, 15(1):1–28, 2007.

[87] Qingfu Zhang and Hui Li. Moea/d: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on evolutionary computation*, 11 (6):712–731, 2007.

[88] Gabriele Eichfelder. An adaptive scalarization method in multiobjective optimization. *SIAM Journal on Optimization*, 19(4):1694–1718, 2009.

[89] Adriana Schulz, Harrison Wang, Eitan Grinspun, Justin Solomon, and Wojciech Matusik. Interactive exploration of design trade-offs. *ACM Transactions on Graphics (TOG)*, 37(4):1–14, 2018.

[90] Zoltán Gábor, Zsolt Kalmár, and Csaba Szepesvári. Multi-criteria reinforcement learning. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 197–205. Morgan Kaufmann Publishers Inc., 1998.

[91] Shie Mannor and Nahum Shimkin. The steering approach for multi-criteria reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 1563–1570, 2002.

[92] S. Parisi, M. Pirotta, N. Smacchia, L. Bascetta, and M. Restelli. Policy gradient approaches for multi-objective sequential decision making. In *2014 International Joint Conference on Neural Networks (IJCNN)*, pages 2323–2330, July 2014. doi: 10.1109/IJCNN.2014.6889738.

[93] Sriraam Natarajan and Prasad Tadepalli. Dynamic preferences in multi-criteria reinforcement learning. In *Proceedings of the 22nd international conference on Machine learning*, pages 601–608, 2005.

[94] Kaiwen Li, Tao Zhang, and Rui Wang. Deep reinforcement learning for multi-objective optimization. *arXiv preprint arXiv:1906.02386*, 2019.

[95] Xi Chen, Ali Ghadirzadeh, Mårten Björkman, and Patric Jensfelt. Meta-learning for multi-objective reinforcement learning. *arXiv preprint arXiv:1811.03376*, 2018.

[96] Andrea Castelletti, Francesca Pianosi, and Marcello Restelli. Multi-objective fitted q-iteration: Pareto frontier approximation in one single run. In *2011 International Conference on Networking, Sensing and Control*, pages 260–265. IEEE, 2011.

[97] Runzhe Yang, Xingyuan Sun, and Karthik Narasimhan. A generalized algorithm for multi-objective reinforcement learning and policy adaptation. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 14610–14621. Curran Associates, Inc., 2019.

[98] Axel Abels, Diederik Roijers, Tom Lenaerts, Ann Nowé, and Denis Steckelmacher. Dynamic weights in multi-objective deep reinforcement learning. In *International Conference on Machine Learning*, pages 11–20, 2019.

[99] Alec Jacobson, Ilya Baran, Jovan Popovic, and Olga Sorkine. Bounded biharmonic weights for real-time deformation. *ACM Trans. Graph.*, 30(4):78, 2011.

[100] Tao Ju, Scott Schaefer, and Joe Warren. Mean value coordinates for closed triangular meshes. In *ACM SIGGRAPH 2005 Papers*, SIGGRAPH '05, page 561–566, New York, NY, USA, 2005. Association for Computing Machinery. ISBN 9781450378253. doi: 10.1145/1186822.1073229. URL `https://doi.org/10.1145/1186822.1073229`.

[101] Yiyue Luo, Yunzhu Li, Pratyusha Sharma, Wan Shou, Kui Wu, Michael Foshey, Beichen Li, Tomás Palacios, Antonio Torralba, and Wojciech Matusik. Learning human–environment interactions using conformal tactile textiles. *Nature Electronics*, 4(3):193–201, 2021.

[102] Daolin Ma, Elliott Donlon, Siyuan Dong, and Alberto Rodriguez. Dense tactile force estimation using gelslim and inverse fem. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 5418–5424. IEEE, 2019.

[103] Ernst Hairer, Christian Lubich, and Gerhard Wanner. *Geometric numerical integration: structure-preserving algorithms for ordinary differential equations*, volume 31. Springer Science & Business Media, 2006.

[104] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[105] Alex Church, John Lloyd, Raia Hadsell, and Nathan F Lepora. Optical tactile sim-to-real policy transfer via real-to-sim tactile image translation. *arXiv preprint arXiv:2106.08796*, 2021.

[106] C. Daniel Freeman, Erik Frey, Anton Raichuk, Sertan Girgin, Igor Mordatch, and Olivier Bachem. Brax - a differentiable physics engine for large scale rigid body simulation. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021.

[107] Vijay R Konda and John N Tsitsiklis. Actor-critic algorithms. In *Advances in neural information processing systems*, pages 1008–1014, 2000.

[108] Jacky Liang, Viktor Makoviychuk, Ankur Handa, Nuttapong Chentanez, Miles Macklin, and Dieter Fox. Gpu-accelerated robotic simulation for distributed reinforcement learning. In *Conference on Robot Learning*, pages 270–282. PMLR, 2018.

[109] Arthur Allshire, Mayank Mittal, Varun Lodaya, Viktor Makoviychuk, Denys Makoviichuk, Felix Widmaier, Manuel Wüthrich, Stefan Bauer, Ankur Handa, and Animesh Garg. Transferring Dexterous Manipulation from GPU Simulation to a Remote Real-World TriFinger. *arXiv preprint arXiv:2108.09779*, 2021.

[110] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, 32, 2019.

[111] Andreas Griewank and Andrea Walther. Introduction to automatic differentiation. *PAMM*, 2(1):45–49, 2003.

[112] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[113] Paavo Parmas, Carl Edward Rasmussen, Jan Peters, and Kenji Doya. Pipps: Flexible model-based policy search robust to the curse of chaos. In *International Conference on Machine Learning*, pages 4065–4074. PMLR, 2018.

[114] Richard S Sutton, Andrew G Barto, et al. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

[115] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[116] Denys Makoviichuk and Viktor Makoviychuk. RL Games, 2021. URL `https://github.com/Denys88/rl_games/`.

[117] Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, et al. Isaac gym: High performance gpu based physics simulation for robot learning. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021.

[118] Peter Norvig and Stuart Russell. *Artificial Intelligence: A modern approach*. Prentice Hall, 3rd edition, 2002.

[119] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL `http://arxiv.org/abs/1412.6980`.

[120] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *Advances in neural information processing systems*, pages 4800–4810, 2018.

[121] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in neural information processing systems*, pages 1024–1034, 2017.

[122] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[123] Raphael Deimel and Oliver Brock. A novel type of compliant and underactuated robotic hand for dexterous grasping. *The International Journal of Robotics Research*, 35(1-3):161–185, 2016.

[124] Daniela Rus and Michael T Tolley. Design, fabrication and control of soft robots. *Nature*, 521(7553):467–475, 2015.

[125] Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, et al. Solving rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113*, 2019.

[126] OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020.

[127] Anusha Nagabandi, Kurt Konolige, Sergey Levine, and Vikash Kumar. Deep dynamics models for learning dexterous manipulation. In *Conference on Robot Learning*, pages 1101–1112. PMLR, 2020.

[128] Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

[129] J. Rapin and O. Teytaud. Nevergrad - A gradient-free optimization platform. `https://GitHub.com/FacebookResearch/Nevergrad`, 2018.

[130] Anne Auger. Benchmarking the (1+ 1) evolution strategy with one-fifth success rule on the bbob-2009 function testbed. In *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*, pages 2447–2452, 2009.

[131] Ingo Rechenberg. Evolutionsstrategien. In *Simulationsmethoden in der Medizin und Biologie*, pages 83–114. Springer, 1978.

[132] Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2):159–195, 2001.

[133] Eugene L Allgower and Kurt Georg. *Introduction to numerical continuation methods*. SIAM, 2003.

[134] N. Riquelme, C. Von Lücken, and B. Baran. Performance metrics in multi-objective optimization. In *2015 Latin American Computing Conference (CLEI)*, pages 1–11, Oct 2015. doi: 10.1109/CLEI.2015.7360024.

[135] E. Zitzler and L. Thiele. Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. *IEEE Transactions on Evolutionary Computation*, 3(4):257–271, Nov 1999. ISSN 1941-0026. doi: 10.1109/4235.797969.

[136] Adriana Schulz, Harrison Wang, Eitan Grinspun, Justin Solomon, and Wojciech Matusik. Interactive exploration of design trade-offs. *ACM Trans. Graph.*, 37 (4), July 2018. ISSN 0730-0301. doi: 10.1145/3197517.3201385. URL `https://doi.org/10.1145/3197517.3201385`.

[137] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

[138] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[139] Tristan Deleu. Model-Agnostic Meta-Learning for Reinforcement Learning in PyTorch, 2018. Available at: https://github.com/tristandeleu/pytorch-maml-rl.

[140] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. *International Conference on Machine Learning (ICML)*, 2017. URL `http://arxiv.org/abs/1703.03400`.

[141] Tao Chen, Miqing Li, and Xin Yao. How to evaluate solutions in pareto-based search-based software engineering? a critical review and methodological guidance. *arXiv preprint arXiv:2002.09040*, 2020.

[142] Eckart Zitzler and Lothar Thiele. Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. *IEEE transactions on Evolutionary Computation*, 3(4):257–271, 1999.

[143] David A Van Veldhuizen. Multiobjective evolutionary algorithms: classifications, analyses, and new innovations. Technical report, AIR FORCE INST OF TECH WRIGHT-PATTERSONAFB OH SCHOOL OF ENGINEERING, 1999.

[144] Carlos A. Coello Coello and Margarita Reyes Sierra. A study of the parallelization of a coevolutionary multi-objective evolutionary algorithm. In Raúl Monroy, Gustavo Arroyo-Figueroa, Luis Enrique Sucar, and Humberto Sossa, editors, *MICAI 2004: Advances in Artificial Intelligence*, pages 688–697, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg. ISBN 978-3-540-24694-7.

[145] Allan Zhao, Tao Du, Jie Xu, Josie Hughes, Juan Salazar, Pingchuan Ma, Wei Wang, Daniela Rus, and Wojciech Matusik. Automatic co-design of aerial robots using a graph grammar. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2022*. IEEE, 2022.

[146] Allan Zhao, Jie Xu, Juan Salazar, Wei Wang, Pingchuan Ma, Daniela Rus, and Wojciech Matusik. Graph grammar-based automatic design for heterogeneous fleets of underwater robots. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 3143–3149, 2022. doi: 10.1109/ICRA46639.2022.9811808.