Learning Locomotion Skills for Cassie: Iterative Design and Sim-to-Real

Zhaoming Xie
University of British Columbia
zxie47@cs.ubc.ca

Patrick Clary
Oregon State University
claryp@oregonstate.edu

Jeremy Dao
Oregon State University
daoje@oregonstate.edu

Pedro Morais
Oregon State University
autranep@oregonstate.edu

Jonanthan Hurst
Oregon State University
jonathan.hurst@oregonstate.edu

Michiel van de Panne
University of British Columbia
van@cs.ubc.ca

Abstract: Deep reinforcement learning (DRL) is a promising approach for developing legged locomotion skills. However, current work commonly describes DRL as being a one-shot process, where the state, action and reward are assumed to be well defined and are directly used by an RL algorithm to obtain policies. In this paper, we describe and document an iterative design approach, which reflects the multiple design iterations of the reward that are often (if not always) needed in practice. Throughout the process, transfer learning is achieved via Deterministic Action Stochastic State (DASS) tuples, representing the deterministic policy actions associated with states visited by the stochastic policy. We demonstrate the transfer of policies learned in simulation to the physical robot without dynamics randomization. We also identify several key components that are critical for sim-to-real transfer in our setting.

Keywords: reinforcement learning, biped locomotion, sim-to-real, control design

1 Introduction

Recent successes in deep reinforcement learning (DRL) have inspired multiple efforts towards learning locomotion policies for legged robots. Impressive results have been demonstrated on quadrupeds [1, 2]. However, it remains to be seen whether similar performance can be demonstrated on human-scale bipeds. In this paper, we demonstrate that locomotion policies can be successfully learned in simulation, and then deployed on the physical robot, as demonstrated on the large-scale Cassie biped. We make the following specific contributions:

- We describe a multi-step design process that reflects the practical, iterative nature of controller design (often left undocumented). This is supported via mixed policy gradients learning. Transfer learning between design iterations is enabled via state-action samples taken from the policies at the previous iteration.
- We apply this design process to train various locomotion policies on a simulated model of the bipedal robot Cassie. These policies are successfully transferred to the physical robot without further tuning or adaptation.
- We investigate the key components that contribute to the sim-to-real success in our setting. Notably, in contrast to other recent sim-to-real results for legged locomotion, no dynamics randomization or learned actuator dynamics models are necessary in our case.

To the best of our knowledge, this is the first time that learned neural network policies for variable-speed locomotion have been successfully deployed on a large $3D$-bipedal robot such as Cassie. The physical robot reaches speeds of $1.24m/s$ with the learned policy, which is 20% faster than other gaits reported in the literature, e.g., [3]. We believe that the iterative design process that we propose may better reflect the realities of controller design than the more conventional view of controller design as a purely algorithmic problem.

2 Related Work

Imitation Learning Imitation learning seeks to approximate an expert policy. In its simplest form, one can collect a sufficiently large number of state-action pairs from an expert and apply supervised learning. This is also referred to as behavior cloning, as used in early seminal autonomous driving work, e.g., Pomerleau [4]. However, this method often fails due to compounding errors and covariate shift [5]. DAGGER [6] is proposed to solve this by iteratively querying the expert policy to augment the dataset. DART [7] injects adaptive noise into the expert policy to reduce expert queries. Recent work uses efficient policy cloning to learn generalized tracking policies via learned latent spaces [8]. Another line of work is to formulate imitation learning as reinforcement learning by inferring the reward signal from expert demonstration using methods such as GAIL [9]. Expert trajectories can also be stored in the experience buffer to accelerate the reinforcement learning process [10, 11]. Imitation learning is also used to combined multiple policies. It is used to train policies to play multiple Atari games [12, 13] and simulated humanoids to traverse different terrains [14].

Bipedal Locomotion Bipedal locomotion skills are important for robots to traverse terrains that are typical in human environments. Many methods use the ZMP to plan stable walking motions, e.g., [15, 16]. Simple models such as SLIP can be used to simplify the robot dynamics [17, 18] for faster planning. To utilize the full dynamics of the robots, trajectory optimization [19] is used to generate trajectories, and tracking controllers based on quadratic programming [20] or feedback linearization [3] can be designed along them. Reinforcement learning has also been applied to bipedal locomotion, with results on physical robots demonstrated on either 2D bipeds [21, 22] or small bipeds with large feet [23, 24]. More recently, DRL has been applied to 3D bipedal locomotion, e.g., [25, 26, 27]. However, they have not yet been shown to be suitable for physical robots.

Sim-to-real Due to the large number of interactions needed for RL algorithms, training directly on robots is usually impractical. Therefore policies are usually trained in simulation and transferred to the physical robots. Discrepancy between simulation and the physical robot, known as the reality-gap, usually causes naive transfer to fail. Robust policies that cross the reality gap can be obtained by randomizing the dynamic properties of the robot across episodes during training. This approach succeeds in various manipulation [28, 29] and quadruped [1, 2] sim-to-real tasks. Another approach is to train policies that are conditioned on the unknown dynamic parameters and to then use physical robot data to do system identification online [24]. Actuator dynamics can be difficult to model and is another potential contributor to the reality gap. Experimental data from actuators can be used to learn a model of the actuator dynamics, which can then be leveraged during training [24, 1].

3 Preliminaries

In this section we briefly outline the reinforcement learning and imitation learning framework.

In reinforcement learning, we wish to learn an optimal policy for a Markov Decision Process (MDP). The MDP is defined by a tuple $\{\mathcal{S}, \mathcal{A}, P, r, \gamma\}$, where $S \in \mathbb{R}^n$, $A \in \mathbb{R}^m$ are the state and action space of the problem, the transition function $P : S \times S \times A \rightarrow [0, \infty)$ is a probability density function, with $P(s_{t+1} \mid s_t, a_t)$ being the probability density of visiting s_{t+1} given that at state s_t, the system takes the action a_t. The reward function $r : S \times A \rightarrow \mathbb{R}$ gives a scalar reward for each transition.$\gamma \in [0, 1]$ is the discount factor. The goal of reinforcement learning is to find a parameterized policy π_θ, where $\pi_\theta : S \times A \rightarrow [0, \infty)$ is the probability density of a_t given s_t, that solves the following optimization:

$$\max_\theta J_{RL}(\theta) = E\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)\right]$$

Policy gradient algorithms [30] are a popular approach to solving this problem, where $\nabla_\theta J_{RL}$ is estimated using on-policy samples, i.e., using data collected from the current stochastic policy.

In imitation learning, we have an MDP as defined above, and an expert policy π_e that can be queried. The goal of imitation learning is to find a parameterized policy π_θ that minimizes the difference between π_θ and π_e. More formally, we aim to solve the following optimization problem:

$$\begin{aligned} \min_\theta J_{imit}(\theta) =& E_{s\sim p_e(s)}[(a-a_e)^2] \\ \text{subject to } & a \sim \pi_\theta(. \mid s) \\ & a_e \sim \pi_e(. \mid s) \end{aligned} \tag{1}$$

where $p_e(s)$ is the probability density of s with policy π_e: $p_e(s) = \sum_{i=0}^{\infty} \gamma^i p(s_t = s \mid \pi_e)$. The expectation in the objective is often estimated by collecting a dataset of expert demonstrations. In behavior cloning, the expert policy is assumed to be deterministic. It suffers from a common failure mode wherein the student policy will accumulate errors over time and drift to states that are dissimilar to the states seen by the expert during data collection. Popular remedies to this issue include DAGGER [6] and DART [7], which query the expert policy iteratively to augment the dataset.

4 Iterative Design Process

Given a fully-defined RL problem and policy parameterization, there exists an ever-growing number of algorithms for learning the solutions. However, a purely algorithmic perspective does not fully reflect the iterative design process that needs to occur in practice and often goes undocumented. In the following, we describe our iterative design process that is well adapted for learning control policies that can reliably run on a physical robot.

The design process relies on three building blocks: reinforcement learning, constrained reinforcement learning and supervised learning. These are shown in Figure 1. An initial policy is obtained using RL with motion tracking rewards. Constrained reinforcement learning, which is realized using a form of mixed policy gradients, is used to iteratively refine the policy as needed. Finally, policy distillation, via supervised learning, can be used to combine and compress multiple policies. We now describe each component in detail, before describing results in Section 5.

4.1 Motion Tracking

It is difficult to obtain policies that can be directly run on a physical robot with the use of simple forward progress rewards, as commonly seen in benchmark locomotion tasks from OpenAI Gym. This naive approach can lead to motions that are unnatural or that have other undesirable properties, including peak torques or impact forces that are incompatible with hardware, e.g., [27]. Instead, we begin with the approach described in [31] and define a reward in terms of tracking a reference motion in order to establish a suitable and predictable initial control policy. Note that the reference motion can be a crude "sketch" of the desired motion, i.e., it can be unphysical and even kinematically inconsistent, having body and foot velocities that are mutually incompatible. We further note that the tracking problem is still an RL problem, and not an imitation learning problem, because the actions that are needed to achieve robust tracking are not observable and therefore need to be learned.

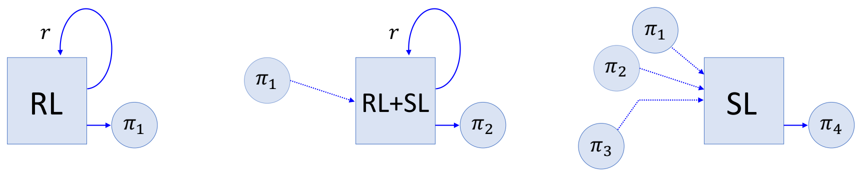


Figure 1: Three building blocks for our design process.

4.2 Policy Transfer Using Sample Tuples

We employ sets of sample tuples to achieve transfer between successive refinements of control policies. This is more flexible than simply transfering the weights in two ways: (i) they can be mixed with RL policy gradient updates, yielding mixed policy-gradient updates in support of learning policy refinements that are constrained in scope; (ii) they directly support distillation to compress and combine policies.

We use a simple and effective expert (s, a) tuple collection technique we refer to as Deterministic Actions Stochastic States (DASS). Intuitively, this is motivated by the fact that actions recorded from a deterministic expert policy will only cover the limit cycle of a motion such as walking. As a result, the student will not observe the feedback that needs to be applied for perturbed states which need to be guided back to the limit cycle. The stochastic actions of a stochastic expert policy does produce (and therefore visit) perturbed states, and for these states we then record the deterministic (noise-free) version of the actions. The learned feedback is illustrated schematically in Figure 2, where the blue curves represent the limit cycle produced by a deterministic policy, and the green arrows represent the deterministic feedback actions associated with the additional states resulting from the execution of the stochastic policy.

Formally, if we assume $\pi_e(. \mid s)$ and $\pi_\theta(. \mid s)$ are Gaussian distributions with the same covariance, minimizing the imitation objective function (1) is equivalent to minimizing $J(\theta) = E_{s\sim p_e(s)}[(\mu_e(s) - \mu_\theta(s))^2]$, where μ_e and μ_θ are the means of π_e and π_θ. It is generally impractical to calculate this expectation exactly; in practice, we will collect an expert dataset $\mathcal{D} = \{(s_i, \mu_e(s_i))\}_{i=1}^N$ of size N, where s_i is the state visited by the expert during policy execution, and minimize training error over $\mathcal{D}$, i.e, we thus solve the following supervised learning problem:

$$\min_\theta J_{SL}(\theta) = E_{s\sim\mathcal{D}}[(\mu_\theta(s) - \mu_e(s))^2] \tag{2}$$

The DASS tuples are closely related to [7], where adaptive noise is added to the expert policy to prevent covariate shift, and Merel et al. [8], where it is used to combine multiple locomotion policies. Algorithm 1 summarizes the data collection procedure.

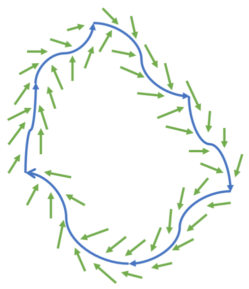


Figure 2: A walking policy produces a limit cycle, represented by the closed blue curve, and the green arrows indicate the required feedback to return to the limit cycle.

Algorithm 1 DASS

```
1: Initialize $\mathcal{D} = \{\}$
2: Reset from some initial state distribution $s_0 \sim p_0(.)$
3: for $i = 0, 1 \dots, N$ do
4:     $\mathcal{D} = \mathcal{D} \cup \{(s_i, \mu_e(s_i))\}$
5:     $a_i \sim \pi_e(. \mid s_i)$, $s_{i+1} \sim P(. \mid s_i, a_i)$
6:     if $s_{i+1} \in \mathcal{T}$ for some termination set $\mathcal{T}$ then
7:         $s_{i+1} \sim p_0(.)$
```

4.3 Policy Refinement

During policy design, it is common to introduce multiple reward terms to achieve desired behaviors. For example, Hwangbo et al. [1] uses up to 11 reward terms to achieve a desired behavior. Instead of doing iterative addition of reward terms and reward reweighting, we choose to iteratively refine policies with the help of DASS. Specifically, at each design iteration, we have access to the policy from previous iteration as our expert policy π_e and a new behavior reward specification, r, and aim to solve the following problem:

$$\begin{aligned}\max_{\theta} J_{RL}(\theta) =& E\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)\right] \\ \text{subject to } & s_{t+1} \sim P(. \mid s_t, a_t) \\ & a_t \sim \pi_\theta(. \mid s_t) \\ & J_{SL}(\theta) \leq \epsilon\end{aligned}$$

To make this problem easier, we make J_{SL} a soft constraint and rewrite the objective to be $J_{total} = J_{RL} - wJ_{SL}$. At each policy update, we will estimate $\nabla_{\theta_t} J_{RL}$ using the usual policy gradient algorithm, and update θ according to the mixed policy gradient defined by $\theta_{t+1} = \theta_t + \alpha(\nabla_{\theta_t} J_{RL} - w\nabla_{\theta_t} J_{SL})$. The resulting policy π_θ will become our expert policy for the next design iteration.

4.4 Policy Distillation

During the design process, we may progressively develop a variety of new policies for the robot, i.e., forwards, backwards, and sideways walking. Instead of training everything from scratch, DASS allows us to train policies specifically for individual new tasks. DASS samples from each policies can then be used to distill them together. This allows for new skills to be obtained without forgetting old skills, and affords experimentation with different neural network architectures.

5 Results

We evaluate our design process using the Cassie robot, shown in Figure 3. It stands approximately 1 meter tall and has a mass of 31 kg, with most of the weight concentrated in the pelvis. There are two leaf springs in each leg to make them more compliant. This introduces extra underactuation into the system and makes the control design more difficult for traditional control techniques.

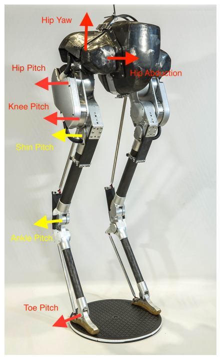


Figure 3: Cassie.

We train multiple policies following our design process, as illustrated by the design paths shown in Appendix A. The results are best seen in the supplementary video, which shows the policies running on the physical robot, after training in simulation.

5.1 Reference Motion Tracking

To learn well behaved policies, we begin the design with policies that attempt to track reference motions while also following desired body velocities. As noted earlier, the reference motions need not be physically feasible. We use three reference motions in our experiments: a stepping-in-place motion (RM1); a walking forwards motion (RM2), and another stepping-in-place motion (RM3). RM1 and RM2 are recorded from an existing functional heuristic-based controller. RM3 is designed using simple keyframes for the foot positions, and inverse kinematics. In all cases, reference motions for various forward speeds and side-stepping are generated by simply translating the pelvis of the robot at the desired speed while leaving the leg motions unchanged. This leads to unphysical artifacts in the reference motion, such as sliding feet, but these do not impede the learning of the tracking control policies.

The input to the policy is $S = \{X, \hat{X}\}$, where X is the state of the robot that evolves according to the robot's dynamics, and $\hat{X}$ is the reference motion that evolves deterministically according to the reference motion. The output of the policy are the residual PD target angles for the active joints, which are then added to the reference joint angles and fed to a low level PD controller. The reward is defined as follows: $r = w_j r_j + w_{rp} r_{rp} + w_{ro} r_{ro}$, where r_j measures how similar the joint angles are to the reference joint angles. r_{rp} and r_{ro} measure similarity of the pelvis position and orientation. The joint reward is computed as $r_j = \exp(-||x_j - \hat{x}_j||^2)$. The remaining terms are similar in structure. The weights are $0.5, 0.3, 0.2$, respectively. Further details are described in Appendix B.1.

The RM1 tracking policy successfully transfers to the physical robot. However, it demonstrates shaky behavior that is ill-suited to the hardware. This motivates subsequent policy refinement.

5.2 Policy Refinement

The policies trained with tracking reference motions alone exhibit undesirable motions, with the pelvis shaking. We make use of mixed policy gradients, with DASS samples, to remain close to these original policies while refining them with a new reward. For the following experiments, we define rewards $r = 0.5r_{design} + 0.5r_{rp}$, where r_{rp} rewards tracking the original desired body positions and r_{design} rewards behavior specified by the designers. The gradient of policy parameters θ then takes the form of $\nabla_\theta = \nabla_\theta J_{RL} - w\nabla_\theta J_{SL}$, where w is a hyperparameter ($w = 10$ in our experiments). We use $r_{design} = \exp\left(-\left\|\omega_{pelvis}\right\|^2\right)$, where ω_{pelvis} is the angular velocity of the pelvis. The resulting policies yield stable walking motion, with significant less pelvis shaking.

We find that the policy trained from tracking RM3 impacts its feet into the ground in a way which could damage the legs. We design a reward $r_{design} = \exp\left(-v_{lfoot}^2 - v_{rfoot}^2\right)$, where v_{lfoot} and v_{rfoot} are vertical velocities for the left foot and right foot respectively. The optimized motion exhibits much softer foot impact and is safe to run on the robot.

We also experiment with other stylistic rewards. We observe that the policies trained from RM2 still exhibit some unnecessarily rough movements, and we thus optimize for reduced joint accelerations. We also experiment with a reward to lift the feet of the robot higher. While the previous policies lift the feet up to 10 cm during each step, here we penalize the policy for lifting the foot less than 20 cm. We test these policies on the physical robot. The motions on the robot are comparable to the motions in simulation. The policy that rewards low joint accelerations makes significantly softer and quieter ground contact. The policy optimized for higher stepping also achieves its goals, lifting its foot up to 20 cm while maintaining good walking motions.

5.3 Policy Distillation

Compression In deep reinforcement learning, network size can play an important role in determining the end result [32]. It is shown in [12] that for learning to play Atari game, a large network is necessary during RL training to achieve good performance, but that the final policy can be compressed using supervised learning without degrading the performance. For our problem, we also observe that using a larger network size for reinforcement learning improves learning efficiency as well as producing more robust policies, as shown in Figure 4.

While we need a large network to efficiently do RL, we find that we can compress the expert policy into a much smaller size network while maintaining the robustness of the final policy. With as little as 600 samples, we can recover a stepping in place policy with a $(16, 16)$ hidden layer size. Table 1 compares policies trained using supervised learning across varying choices of hidden layer sizes, numbers of training samples, and the presence or absence of noise during data collection. With only 600 samples, a large network can easily overfit the training data. We find that while larger networks can have this issue, with validation error orders of magnitude larger than the training error, the resulting policy still performs comparably to the original policy in terms of robustness. We also successfully test the $(16, 16)$ policy on the physical robot, demonstrating that the policy is indeed robust to unmodeled perturbation.

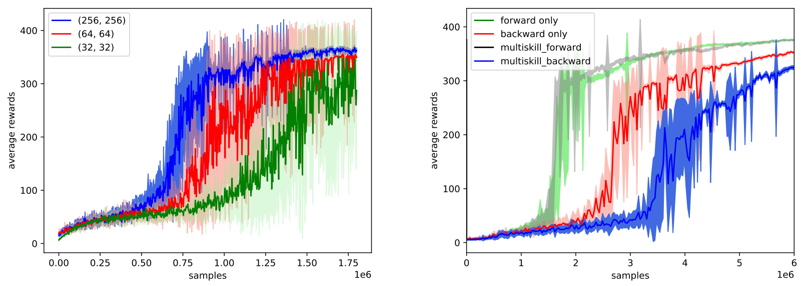


Figure 4: Learning Curves. Left: Network size impacts the final RL results, as computed across the average of 3 runs. Larger network sizes learn faster and yield more stable policies. Right: Comparison of learning skills separately and learning all the skills together.

# samples; NN size	training loss	validation loss	no noise	0.1 policy noise	20% mass increase	50N pushes
expert	0	0	394.53	389.82	378.32	367.28
300, $(16, 16)$	$1.20 \pm 0.08 \times 10^{-3}$	$2.99 \pm 0.25 \times 10^{-3}$	392.88 ± 0.34	366.53 ± 26.79	350.36 ± 15.08	346.27 ± 5.68
600, $(16, 16)$	$1.33 \pm 0.20 \times 10^{-3}$	$2.08 \pm 0.12 \times 10^{-3}$	394.24 ± 0.48	388.66 ± 0.37	375.03 ± 3.49	363.56 ± 3.11
600, $(512, 512)$	$6.00 \pm 1.18 \times 10^{-6}$	$5.45 \pm 0.42 \times 10^{-4}$	394.36 ± 0.34	389.49 ± 0.34	371.65 ± 4.04	351.52 ± 19.00
600, $(8, 8)$	$5.16 \pm 0.42 \times 10^{-3}$	$6.17 \pm 0.65 \times 10^{-3}$	92.52 ± 20.94	72.44 ± 10.63	60.69 ± 9.64	81.47 ± 12.94
600, $(16, 16)$, no DASS	$5.04 \pm 1.73 \times 10^{-4}$	$8.11 \pm 0.94 \times 10^{-3}$	66.19 ± 8.96	50.66 ± 4.79	65.82 ± 13.47	66.59 ± 6.00

Table 1: Policies trained with various settings. We report the training/validation loss and rewards over 400 steps after injecting action noise, changing the body mass, and using (50N, 0.2s) pushes.

Combining Policies After training a policy for a specific skill, we may want the policy to learn additional skills. In the context of the Cassie robot, we desire a control policy to not only step in place, but to also walk forwards and backwards on command. Distillation via supervised learning provides a convenient way to integrate multiple policies into a single policy, where they do not conflict, and without forgetting. We distill expert policies that are trained separately for walking forwards (v=0.8 m/s), stepping in place, and walking backwards (v=-0.8 m/s) into one policy that masters all three tasks. As noted earlier, the velocity is commanded via the desired body velocity of the original reference motion, which is otherwise left unchanged. In this way, the policy is conditioned on the commanded velocity. With remarkably few samples (600) collected from each of these three policies, these policies are combined into one policy with hidden layer sizes of $(64, 64)$. This is then capable of a full range of walking forwards/backwards walking speeds while turning towards a desired heading angle, and it successfully transfers to the physical robot.

An alternative to distillation is to simultaneously learn the desired skills, e.g., to directly train a policy that is conditioned on the desired speed, with the desired speeds for any given training episode being sampled from the range of desired speeds. Such simultaneously learning could potentially be advantaged by transfer effects and potentially disadvantaged by interference effects. We conduct an experiment to compare the sample efficiency between learning forwards and backwards walking skills separately, and learning these skills at the same time. In particular, we train 3 policies: (a) walking forwards at speeds sampled from $[0, 0.8]m/s$; (b) walking backwards at speeds sampled from $[-0.8, 0]m/s$; and (c) ”multiskill”: walking forwards and backwards at speeds sampled from $[-0.8, 0.8]m/s$. The learning curves are shown in Figure 4(right). For fair comparison, the multiskill training cases only count samples coming from the relevant individual skills, i.e., multiskill-forwards only counts samples from multiskill episodes that request forwards velocities. The results show that (i) backwards walking is more difficult to learn than forwards walking; (ii) multiskill makes no difference for learning to walk forwards; (iii) multiskill is disadvantageous for learning to walk backwards, due to interference from the interleaved forwards learning.

6 Sim-to-real Transfer

The policies are trained in simulation and deployed on the physical robot, with hardware details provided in Appendix B.2. Here we identify a number of the key ingredients that we find to be critical for sim-to-real success in our setting.

System Identification Careful system identification is performed on the robot to ensure that dynamics properties such as the mass and inertia of each link is reasonably accurate. We specifically note the importance of the reflected inertia of motors, in particular for the knee joint, and which is not directly discussed in other recent work [2, 1, 24]. In the Mujoco simulator, the reflected inertia can be modeled using the armature attribute. For the Cassie simulation, setting the reflected inertia to 0 causes unstable dynamics, which makes learning impossible.

Despite careful system identification, modeling errors are unavoidable. A common approach for addressing these issues is to use dynamic randomization during training, where dynamic properties of the robot are sampled from some distributions centered around the measured values. This approach is found to be essential to achieve sim-to-real success for manipulation tasks [28, 29] and quadrupedal locomotion [2, 1]. Surprisingly, our policies can transfer to the physical robot without dynamic randomization during training. This is for a class of motions (dynamic bipedal walking on small feet) that are arguably more unstable than other recent sim-to-real results.

Actuator Dynamics Unmodeled actuator dynamics also poses a challenge for successful sim-to-real, including for direct model-based methods. Despite such methods having theoretical guarantees, manual tuning of control parameters remains critical to successful transfer to Cassie, e.g., [3]. One possible solution is to learn dedicated models for the actuators using data collected from the physical robots, e.g., [1, 24]. In our setting, we have thus far found this to be unnecessary. We hypothesize that the action noise injected during training gives additional robustness to the policy.

State Estimation We find that using state estimation during training is critical for successful sim-to-real transfer. While the simulation has access to the true state of the robot, the hardware requires estimating the state from noisy sensory measurements. Policies trained using the true state in simulation fail during test time in simulation, where the state estimation is used with simulated sensors. They also fail on the physical robot. When the state estimator is employed during training, the policies perform well at test time, in simulation and on the physical robot.

State Space Since the initial policy that we build on tracks a time-varying reference motion, the resulting polices are also time-varying. The time-indexed nature of the policies comes into play via the reference motions, which remain an input. However, since the reference motions remain unchanged except for the velocity of the pelvis, it can be replaced by a motion-phase (or clock) variable that is reset every walking cycle, in addition to providing the desired speed. We use distillation with DASS samples to train a new policy that is conditioned only on the state, phase, and desired speed. The resulting policy reliably runs in simulation and on the physical robot. We then attempt another distillation without the phase information. Interestingly, this policy performs well in simulation, but fails to take steps on the robot and eventually falls down. We speculate that this may be due in part to some states remaining ambiguous in the absence of any memory, e.g., when stepping in place, during double-stance the state that preceeds left-leg and right-leg step is identical.

Action Space Our policies produce a *residual* PD target, which is added to the reference motion pose to produce full PD-target joint angles. In order to test the benefit of this residual-action architecture, we experiment with using distillation and DASS to train a new *direct* policy that produces the full PD-target actions. As before, the policy still has access to the reference motion pose as an input. This policy performs well in simulation but fails on the robot. We speculate that this may occur because for states that are out-of-distribution, the residual-action architecture will still perform a general walking motion as driven by the reference motion, while the direct policy does not have this default behavior. It is also possible to use a residual PD target with a purely state-indexed policy, e.g., [24, 2]. However, this makes the problem non-Markovian since the open loop PD-target is time varying. Other state space features to explore include using state and action histories [1].

Sim-to-Real Performance The policies demonstrate the desired characteristic as specified during the refinement steps, such as more stable pelvis movement, less foot impact, and lifting the feet higher. Furthermore, the policies are robust to perturbations such as occasional steps onto uneven ground and pushes in various directions. Experiments also show that the learned policies do well at tracking the desired velocities (Appendix C).

7 Conclusion

We have presented an iterative design process that uses RL to learn robust locomotion policies for Cassie. The process is based on three well-defined building blocks, and aided by a data collection technique (DASS) that enable us to refine, combine, and compress policies. We demonstrate walking motions on Cassie with various styles and speeds. While the details of the specific RL algorithm always remain important, they are only part of the story for achieving successful sim-to-real results.

We show that learned policies can be robust without resorting to dynamics randomization. We investigate the critical components needed for the success of sim-to-real in our setting, including accurate system identification, inclusion of the state estimator, and choices of state-and-action spaces. We do not claim that these results are fully general, as different settings may yield different (and occasionally conflicting) insights, as we have already noted for dynamics randomization. An important direction for further investigation includes determining how best to achieve sim-to-real success with more general state-and-action spaces, e.g., removing time-related inputs and removing the need to work with residual actions.

Acknowledgments

This research was funded in part by NSERC RGPIN-2015-04843 and NSF award 1849343 S&AS:INT:Learning and Planning for Dynamic Locomotion.

References

[1] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter. Learning agile and dynamic motor skills for legged robots. *Science Robotics*, 4(26), 2019.

[2] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke. Sim-to-real: Learning agile locomotion for quadruped robots. *CoRR*, abs/1804.10332, 2018.

[3] Y. Gong, R. Hartley, X. Da, A. Hereid, O. Harib, J. Huang, and J. W. Grizzle. Feedback control of a cassie bipedal robot: Walking, standing, and riding a segway. *CoRR*, abs/1809.07279, 2018.

[4] D. Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In *NIPS*, 1988.

[5] S. Ross and D. Bagnell. Efficient reductions for imitation learning. In *Proceedings of the Thirteenth AISTATS*, pages 661–668. PMLR, 2010.

[6] S. Ross, G. J. Gordon, and J. A. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *AISTATS*, 2011.

[7] M. Laskey, J. Lee, R. Fox, A. D. Dragan, and K. Y. Goldberg. Dart: Noise injection for robust imitation learning. In *CoRL*, 2017.

[8] J. Merel, L. Hasenclever, A. Galashov, A. Ahuja, V. Pham, G. Wayne, Y. W. Teh, and N. Heess. Neural probabilistic motor primitives for humanoid control. *CoRR*, abs/1811.11711, 2018.

[9] J. Ho and S. Ermon. Generative Adversarial Imitation Learning. *arXiv e-prints*, art. arXiv:1606.03476, June 2016.

[10] S.-A. Chen, V. Tangkaratt, H.-T. Lin, and M. Sugiyama. Active Deep Q-learning with Demonstration. *arXiv e-prints*, art. arXiv:1812.02632, Dec. 2018.

[11] X. Chen, A. Ghadirzadeh, J. Folkesson, and P. Jensfelt. Deep reinforcement learning to acquire navigation skills for wheel-legged robots in complex environments. *CoRR*, abs/1804.10500, 2018.

[12] A. A. Rusu, S. G. Colmenarejo, Ç. Gülçehre, G. Desjardins, J. Kirkpatrick, R. Pascanu, V. Mnih, K. Kavukcuoglu, and R. Hadsell. Policy distillation. *CoRR*, abs/1511.06295, 2015.

[13] E. Parisotto, L. J. Ba, and R. Salakhutdinov. Actor-mimic: Deep multitask and transfer reinforcement learning. *CoRR*, abs/1511.06342, 2015.

[14] G. Berseth, C. Xie, P. Cernek, and M. van de Panne. Progressive reinforcement learning with distillation for multi-skilled motion control. In *ICLR*, 2018.

[15] K. Hirai, M. Hirose, Y. Haikawa, and T. Takenaka. The development of honda humanoid robot. In *Proceedings. 1998 IEEE International Conference on Robotics and Automation*, volume 2, pages 1321–1326 vol.2, May 1998.

[16] R. Tedrake, S. Kuindersma, R. Deits, and K. Miura. A closed-form solution for real-time zmp gait generation and feedback stabilization. In *IEEE-RAS International Conference on Humanoid Robots*, Seoul, Korea, 2015.

[17] T. Apgar, P. Clary, K. Green, A. Fern, and J. Hurst. Fast online trajectory optimization for the bipedal robot cassie. In *Robotics: Science and Systems*, 2018.

[18] X. Xiong and A. D. Ames. Coupling reduced order models via feedback control for 3d underactuated bipedal robotic walking. *IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)*, 2018.

[19] A. Hereid, O. Harib, R. Hartley, Y. Gong, and J. W. Grizzle. Rapid bipedal gait design using C-FROST with illustration on a cassie-series robot. *CoRR*, abs/1807.06614, 2018.

[20] M. Posa, S. Kuindersma, and R. Tedrake. Optimization and stabilization of trajectories for constrained dynamical systems. In *Proceedings of the International Conference on Robotics and Automation*, 2016.

[21] E. Schuitema, M. Wisse, T. Ramakers, and P. Jonker. The design of leo: A 2d bipedal walking robot for online autonomous reinforcement learning. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3238–3243, Oct 2010.

[22] T. Li, A. Rai, H. Geyer, and C. G. Atkeson. Using deep reinforcement learning to learn high-level policies on the ATRIAS biped. *CoRR*, abs/1809.10811, 2018.

[23] R. Tedrake, T. W. Zhang, and H. S. Seung. Stochastic policy gradient reinforcement learning on a simple 3d biped. *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, 3:2849–2854 vol.3, 2004.

[24] W. Yu, V. C. V. Kumar, G. Turk, and C. K. Liu. Sim-to-real transfer for biped locomotion. *CoRR*, abs/1903.01390, 2019.

[25] X. B. Peng, G. Berseth, K. Yin, and M. van de Panne. Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning. *ACM Transactions on Graphics (Proc. SIGGRAPH 2017)*, 36(4), 2017.

[26] W. Yu, G. Turk, and C. K. Liu. Learning symmetric and low-energy locomotion. *ACM Transactions on Graphics (Proc. SIGGRAPH 2018 - to appear)*, 37(4), 2018.

[27] N. Heess, D. TB, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, S. M. A. Eslami, M. A. Riedmiller, and D. Silver. Emergence of locomotion behaviours in rich environments. *CoRR*, abs/1707.02286, 2017.

[28] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–8, May 2018. doi:10.1109/ICRA.2018.8460528.

[29] OpenAI, M. Andrychowicz, B. Baker, M. Chociej, R. Józefowicz, B. McGrew, J. W. Pachocki, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, J. Schneider, S. Sidor, J. Tobin, P. Welinder, L. Weng, and W. Zaremba. Learning dexterous in-hand manipulation. *CoRR*, abs/1808.00177, 2018.

[30] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *NIPS*, 1999.

[31] Z. Xie, G. Berseth, P. Clary, J. Hurst, and M. van de Panne. Feedback control for cassie with deep reinforcement learning. In *Proc. IEEE/RSJ Intl Conf on Intelligent Robots and Systems (IROS 2018)*, 2018.

[32] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger. Deep reinforcement learning that matters. In *AAAI*, 2018.

[33] X. B. Peng, P. Abbeel, S. Levine, and M. van de Panne. Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Transactions on Graphics (Proc. SIGGRAPH 2018)*, 37(4), 2018.

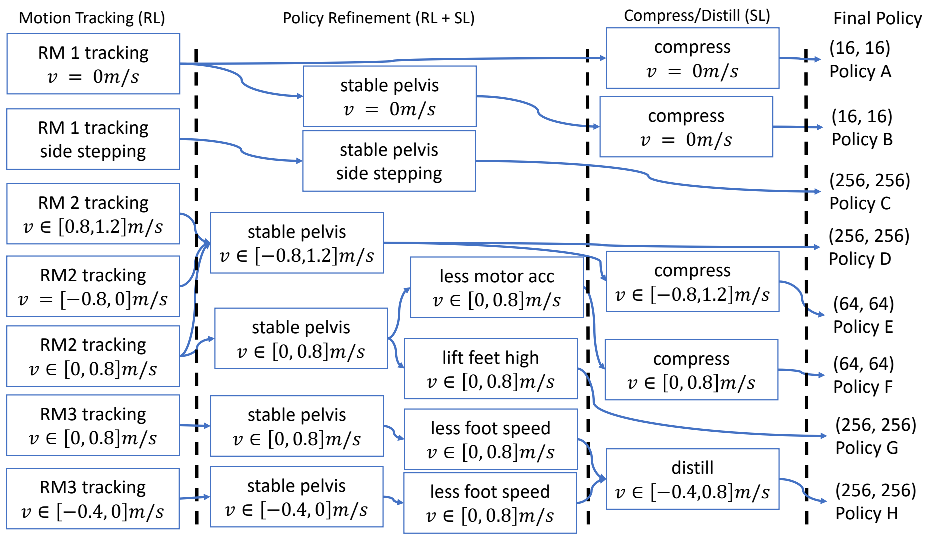
[34] E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, Oct 2012.

[35] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[36] A. Hereid, C. M. Hubicki, E. A. Cousineau, and A. D. Ames. Dynamic humanoid locomotion: A scalable formulation for hzd gait optimization. *IEEE Transactions on Robotics*, 34(2):370–387, April 2018. ISSN 1552-3098.

Appendix A List of Policies

We train several policies following our design process. A diagram of our design choices are shown in Figure. 5, and we demonstrate several policies running on the physical robots in the video.



Motion Tracking (RL)
Policy Refinement (RL + SL)
Compress/Distill (SL)
Final Policy
RM 1 tracking $v = 0m/s$
RM 1 tracking side stepping
RM 2 tracking $v \in [0.8,1.2]m/s$
RM2 tracking $v = [-0.8, 0]m/s$
RM2 tracking $v \in [0, 0.8]m/s$
RM3 tracking $v \in [0, 0.8]m/s$
RM3 tracking $v \in [-0.4, 0]m/s$
stable pelvis $v = 0m/s$
stable pelvis side stepping
stable pelvis $v \in [-0.8,1.2]m/s$
stable pelvis $v \in [0, 0.8]m/s$
stable pelvis $v \in [0, 0.8]m/s$
stable pelvis $v \in [-0.4, 0]m/s$
less motor acc $v \in [0, 0.8]m/s$
lift feet high $v \in [0, 0.8]m/s$
less foot speed $v \in [0, 0.8]m/s$
less foot speed $v \in [0, 0.8]m/s$
compress $v = 0m/s$
compress $v = 0m/s$
compress $v \in [-0.8,1.2]m/s$
compress $v \in [0, 0.8]m/s$
distill $v \in [-0.4,0.8]m/s$
(16, 16) Policy A
(16, 16) Policy B
(256, 256) Policy C
(256, 256) Policy D
(64, 64) Policy E
(64, 64) Policy F
(256, 256) Policy G
(256, 256) Policy H

Figure 5: Our policy design process. Several tracking-based policies are used as a starting point. DASS samples are passed from one policy to the next according to the arrows.

Appendix B Experiment Setup

B.1 Reinforcement Learning Setup

The state of the robot includes the height, orientation expressed as a unit quaternion, velocities, angular velocities and acceleration of the pelvis, joint angles and joint velocities of each joint. Combined with the reference motion, this gives us a $85D$ input vector. We use the commonly-adopted Gaussian Policy as output, where the neural network will output the mean of the policy and Gaussian noise is injected on top of the action during execution. The output and the reference motion are summed to produce target joints angles for a low level PD controller. Instead of making the covariance of the Gaussian policy a learnable parameter, we use a fixed covariance for our policy. We assume that the Gaussian distribution in each dimension is independent, with a standard deviation of ≈ 0.1 radians. A benefit of the fixed covariance is that because of the noise constantly injected into the system during training, the resulting policy will adapt itself to handle unmodeled disturbances during testing, as demonstrated in previous work [31, 33]. The network architecture is shown in Fig. 6.

The policy is trained with an actor-critic algorithm using a simulated model of Cassie with the MuJoCo simulator [34], with the gradient of the policy estimated using Proximal Policy Optimization [35]. The simulator includes a detailed model of the robot's rigid-body-dynamics, including the reflected inertia of the robot's motors, as well as empirically measured noise and delay for the robot's sensors and actuators.

The reference motions we work with are symmetric and the robot itself is nearly symmetric, and thus it is natural to enforce symmetry in the policies as well. We adopt a similar approach to [36], where we transform the input and output every half walking cycle to their symmetric form. During training, we apply reference state initialization and early termination techniques as suggested by Peng et al. [33], where each rollout is started from some states sampled from the reference motions and is terminated when the height of the pelvis is less than 0.4 meters, or whenever the reward for any given timestep falls below a threshold of 0.3.

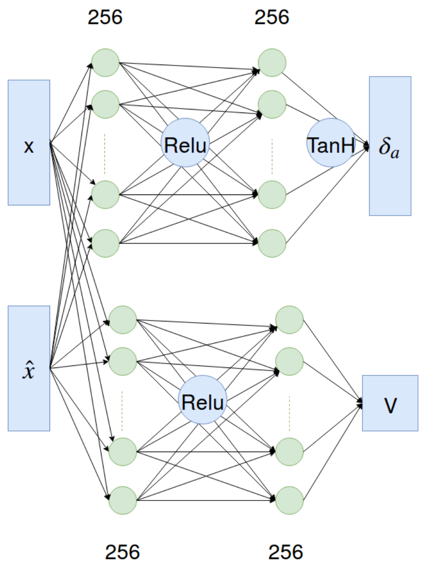


Figure 6: Network architecture used.

During training, the desire orientation of the pelvis is always facing the $+x$ direction, and the robot is initialize to be off the facing direction by $\theta \sim [-0.1\pi, 0.1\pi]$ radians to enable it to learn to turn. During test time, the orientation of the pelvis will be transformed by the desire heading commanded by the operator to allow the robot to turn to various directions.

B.2 Hardware Tests

We deploy a selection of trained policies on a physical Cassie robot. The state of the robot is estimated using sensor measurements from an IMU and joint encoders, which are fed into an Extended Kalman Filter to estimate parts of the robot's state, such as the pelvis velocity. This process runs at 2 kHz on an embedded computer directly connected to the robot's hardware. This information is sent over a point-to-point Ethernet link to a secondary computer on board the robot, which runs a standard Ubuntu operating system and executes the learned policy using the PyTorch framework. The policy updates its output joint PD targets once every 30 ms based on the latest state data and sends the targets back to the embedded computer over the Ethernet link. The embedded computer executes a PD control loop for each joint at the full 2 kHz rate, with targets updating every 30 ms based on new data from the policy execution.

Rapid deployment and testing is aided by the simulator using the same network-based interface as the physical robot, which means that tests can be moved from simulation to hardware by copying files to the robot's on board computer and connecting to a different address. The robot has a short homing procedure performed after powering on, and can be left on in between testing different policies. The same filtering and estimation code as used on hardware is used internally in the simulator, rather than giving the policy direct access to the true simulation state. The network link between two computers introduces an additional 1-2 ms of latency beyond running the simulator and policy on the same machine.

Appendix C Sim-to-Real Performance

Figure 7 shows the performance of the speed tracking. The same sequence of speed command is used in simulation for comparison. Although the speed has larger within-step variation on the physical robot as compared to the simulation, the average speeds match those commanded by the operator. This shows that although the simulation is not a perfect match to the physical robot, the policy is able to adapt and perform well.

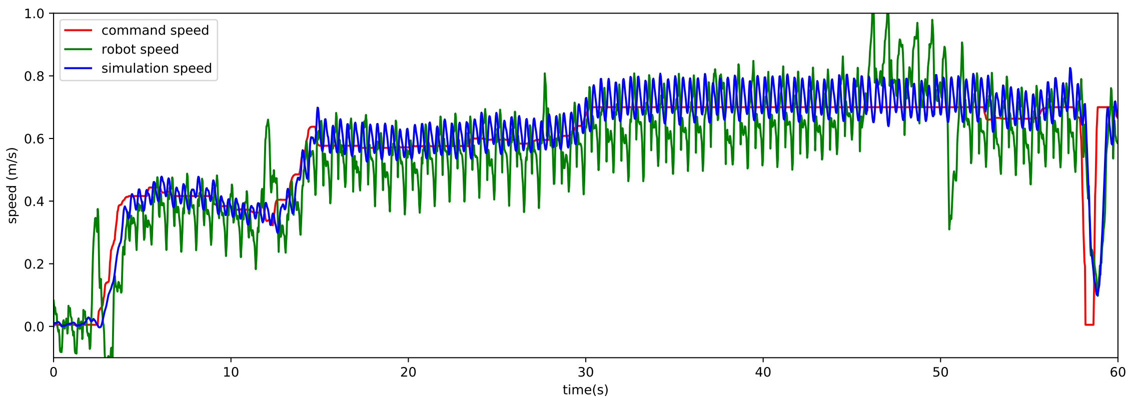


Figure 7: Speed tracking performance of a learned policy, in simulation vs on the physical robot.