AN ABSTRACT OF THE THESIS OF

Jonah A. Siekmann for the degree of Master of Science in Robotics presented on March 15, 2023.

Title: Robust Reference-Free Sim-to-Real Reinforcement Learning for Bipedal Locomotion

Abstract approved: ______________________________

Alan Fern

In recent years, model-free Deep Reinforcement Learning (RL) has become an increasingly popular alternative to more traditional model-based or optimization-based control methods in solving robotic legged locomotion. However, deploying RL in the real world can be a significant undertaking. Constructing reward functions which compel controllers to learn the desired behavior is not straightforward. For example, can a reward function which trains a controller to stand be easily modified to train it to walk or run? This thesis seeks to provide insights into training such controllers in ways that make desired behaviors easier to realize by parameterizing behaviors in a low-dimensional periodic space which covers the entire breadth of legged gaits. In addition, it explores the limits of the approach by training controllers to interact (in the real world) with challenging terrain conventionally assumed to be extremely difficult for bipedal robots.

Robust Reference-Free Sim-to-Real Reinforcement Learning for Bipedal Locomotion

by

Jonah A. Siekmann

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Presented March 15, 2023
Commencement June 2023

ACKNOWLEDGEMENTS

I would like to thank Kevin Green, Yesh Godse, Helei Duan, and Jeremy Dao for their advice, mentorship, and collaborative spirit which I have enjoyed during my time at the Dynamic Robotics Laboratory. I am grateful to Lorenzo Bermillo and John Warila for their clever insights and engineering know-how. Thank you to my advisors Alan Fern and Jonathan Hurst for their mentorship and support during my time as a graduate student.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

LIST OF TABLES

Chapter 1: Introduction

For several decades, model-based methods have enjoyed significant popularity in research into robotic bipedal locomotion [56, 58, 40, 22, 57, 6]. These methods are usually physically principled and benefit from a high degree of interpretability, which make them attractive solutions to engineers seeking easily understandable and modifiable control algorithms. However, robotic bipedal locomotion is known to be a dynamic and unstable phenomenon which is difficult to describe in simple physical terms due to complicated contact dynamics. Engineers must therefore choose a tradeoff between models which may be closer to physical reality but suffer from slower-than-real-time execution speeds, or simpler models which are cheap to execute but may fail to describe reality accurately outside of some narrow band of behavior.

A recent alternative to model-based control methods for robotic control problems, known as model-free deep Reinforcement Learning (RL), has come into prominence within the last decade [35, 34, 32, 5, 61, 52, 26, 28, 60]. Rather than make use of a reduced-order, physically principled model of the robot for control, it learns a control policy through interaction with a full-order simulation of the entire robot, which is then deployed on the real robot. Engineers must specify the desired behavior through the use of a reward function, which the control policy will attempt to maximize through trial and error over the course of months or even years of simulated experience. Thanks to advances in computing hardware and deep learning, this process can take as little as a few hours.

Further, the practical problem of conceiving of useful reward functions which train a control policy to accurately encapsulate the desired behavior remains hard. While simple reward functions are usually preferred, rewards which are not sufficiently specified may result in the policy learning physically unrealistic [54] or dangerous behaviors. On the other hand, complex reward functions may accurately capture a certain behavior without necessarily yielding insight into how to design reward functions which enable different behaviors. For example, creating a reward which teaches a robot to walk may not be easily modified to teach the robot how to skip.

In Chapter 3, we describe a probabilistic reward framework which allows us to create reward functions to learn the entire spectrum of 2-beat and 4-beat bipedal gaits. We are able to successfully train policies which can learn to stand, walk, run, hop, gallop, and skip, and even learn policies which can learn all behaviors and smoothly interpolate between them in real-time. This work was published in the International Conference on Robotics and Automation (2021) and nominated for the Best Paper award.

In Chapter 4, we explore the limits of the approach introduced in Chapter 3 by training a policy to ascend and descend stairs in the real world despite being completely blind (no exteroceptive input). By providing the policy the proprioceptive state of the Cassie robot and training it on hundreds of thousands of interactions walking over randomized staircases under randomized dynamics (while asking it to maximize the reward function introduced in Chapter 3), we can successfully learn such a policy with no fundamental alterations to the approach. This work was published in the Robotics: Science and Systems (2021) conference.

Chapter 2: Background

2.1 Reinforcement Learning

Reinforcement learning is a machine learning paradigm that seeks to train agents to maximize an expected reward through trial-and-error [50]. Reinforcement learning problems are often formalized as an agent interacting with a Markov decision process (MDP) in order to learn a behavior to maximize expected returns. This problem is typically presented in a manner in which an agent receives a state s_t at timestep t from the MDP, which the agent acts on based on its policy $\pi(a_t|s_t)$. The MDP's transition function receives the agent's action a_t and returns the next state s_{t+1} and reward r_t based on the action taken. The policy is often a stochastic policy, in which case it is a function $\pi(a|s)$ which takes in a state s and outputs the parameters of a distribution, usually the mean and standard deviation of a normal distribution. The reward $r = R(s, a)$ is a scalar signal that expresses how good a particular state-action pair is. The agent's goal is then to find an optimal policy that maximizes expected return $J(\pi)$,

$$J(\pi) = \mathbb{E}_\pi \left[\sum_{t=0}^{T} \gamma^t r_t \right]$$

where T is the horizon of an episode, and $\gamma \in [0, 1]$ is the discount factor.

2.2 Proximal Policy Optimization (PPO)

An effective solution to many RL problems is the family of policy gradient algorithms, in which the gradient of the expected return with respect to the policy parameters is computed and used to update the parameters through gradient ascent. PPO is a model-free policy gradient algorithm which samples data through interaction with the environment and optimizes a "surrogate" objective function. PPO introduces a modified objective function that adopts clipped probability ratios which forms a pessimistic estimate of the policy's performance [43]. It also addresses the problem of excessive policy updates by restricting changes that move the probability ratio,

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$$

too far away from 1. The probability ratio is a measure of how different the current policy is from the previous policy (the policy before the last update). The smaller the ratio the greater the difference. The "surrogate" objective function is then modified into the clipped objective:

$$L(\theta) = \mathbb{E}_t \left[min \left(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right]$$

where ϵ is a tunable hyperparameter that increases or decreases the bounds which constrain the probability ratio $r_t(\theta)$. Clipping the probability ratio discourages the policy from changing too much and taking the minimum results in using the lower, pessimistic bound of the unclipped objective. Thus any change in the probability ratio $r_t(\theta)$ is included when it makes the objective worse, and otherwise is ignored [43]. This can prevent the policy from changing too quickly and leads to more stable learning.

2.3 RNNs for Control Problems

RNNs have been successfully applied to many control domains using reinforcement learning, often resulting in performance superior to feedforward networks. [47] shows that using deep recurrent Q networks (DRQN) instead of conventional feedforward Q Networks in UAV navigation results in less collisions and more energy-efficient performance [47]. [20] also use DRQNs for Atari games, finding that DRQNs with a single observation are a viable alternative to DQNs with a history of states, and that DRQNs are more robust to partial observability that non-memory based agents. It has been shown that RNNs can store and recollect information for arbitrarily long amounts of time [23], as well as perform system identification as noted in [21] and further explored in [35]. Furthermore, RNNs can do so through gradient descent, without hand-tuning of hyper-parameters, in contrast to feedforward networks which require access to hand-picked fixed window of state histories [62].

Thus, while RNNs have been shown to be extremely successful in achieving superior performance to feedforward networks on a variety of robotic control tasks, some even on hardware [35], they have not yet been demonstrated for the task of real-world robotic bipedal locomotion. This task differs significantly from other control tasks, such as robotic arm manipulation, due to the significant underactuation of the system and complicated contact dynamics.

2.4 Dynamics Randomization

Dynamics randomization [35] [52] is the practice of randomizing physics parameters of the simulated environment in the hopes that training agents on a variety of possible dynamics will lead to better performance in the real world. [52] leverage this technique to learn quadruped locomotion from scratch in a physics simulator and then deploy the learned controller into hardware, while [35] use a similar system to train a robotic arm to manipulate objects in the real world. They improve the robustness of control policies by simulating latency as well as physical properties such as mass, joint center of mass, joint damping, and other similar parameters of the environment.

More formally, using notation from [35], the objective is to train a memory-based agent to perform manipulation tasks under the conditions set by the real world dynamics $p^*(s_{t+1}|s_t, a_t)$. However, sampling from these dynamics is not very time-efficient. Instead, the agent is trained across a wide range of possible dynamics by using a set of dynamics parameters μ, sampled from a multivariate uniform distribution, to parameterize the simulation dynamics $\hat{p}_\mu(s_{t+1}|s_t, a_t)$, so the objective is reframed as attempting to maximize the expected return over the distribution of dynamics parameters ρ_μ.

2.5 Motion Synthesis

The practice of synthesizing locomotion behaviors has been studied in a variety of fields. In character animation, kinematics-based approaches (i.e., those using motion capture data) are commonly used to synthesize locomotion [29] [42] [59], with some newer approaches using deep learning [63] to train neural networks to solve problems like adapting realistically to varying ground geometries [24] [48] or blending several types of actions into one realistic body motion [49]. Physics-based character animation, wherein the pose of a character is controlled through the application of torques and motions are simulated using a physics engine, has also come into prominence in recent years [3], though is notably more difficult to use effectively for complicated motions [17] when compared to kinematics-based methods.

Approaches which use reinforcement learning to synthesize locomotion bear heavy similarities to the aforementioned methods used in character animation. Much recent work uses trajectory-matching reward functions to train policies to imitate some reference trajectory (akin to a motion capture) while subjecting the policies to simulated physics, resulting in policies that behave realistically while imitating some reference trajectory [7] [36] [61] [37] [45]. Reinforcement learning has also been used for synthesizing locomotion without the use of reference trajectories, but these approaches often place little emphasis on subjective quality of behaviors, resulting in behaviors which maximize some objective while producing policies which are often inefficient, infeasible or unsafe to execute in the real world, and not usually visually pleasing [11] [54]. Methods which do prioritize physically realistic behavior without using a reference trajectory exist [19] [53] [26], but their reward functions are specific to single behaviors and are not trivial to extend to other behaviors.

Chapter 3: Sim-to-Real Learning of All Common Bipedal Gaits via Periodic Reward Composition

3.1 Introduction

Using reinforcement learning (RL) to learn all of the common bipedal gaits found in nature for a real robot is an unsolved problem. A key challenge of learning a specific locomotion gait via RL is to communicate the gait behavior through the reward function. In general, a specific gait can be viewed as a dynamic process that has a characteristic periodic structure, but is also able to flexibly adapt to moderate environment disturbances. This suggests two considerations when designing a gait reward function. First, the reward must be specific enough to produce the desired gait characteristic when optimized. Second, to account for the fact that there is uncertainty about the exact details of a gait in the context of specific terrain and dynamic conditions, the reward should not be overly constraining.

The common use of reference trajectories to specify gait-specific rewards, e.g., [63, 49, 61, 36**?**], partly addresses the first consideration above, but mostly ignores the second. In particular, a reference trajectory only captures a small part of the variation needed to realize a gait characteristic under varying conditions. Thus, attempting to adhere to such a trajectory can prevent learning a characteristic gait that is more robust and/or efficient, not to mention that deriving feasible reference trajectories for a

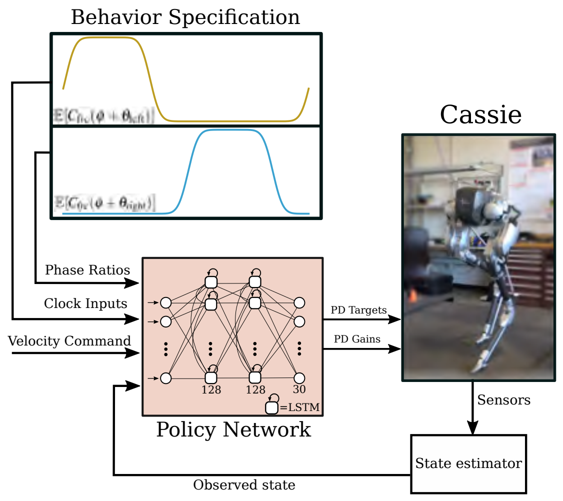


Figure 3.1: In this chapter, we present a reward design framework which makes it easy to learn policies which can stand, walk, run, gallop, hop, and skip on hardware. We condition the reward function based on a number of gait parameters, and also provide these parameters to the LSTM policy, which outputs PD joint position targets and PD gains to the robot.

particular desired gait can be very challenging in the first place.

Reference-free approaches to specifying reward functions for locomotion are often highly underspecified, for example, those used in the OpenAI Gym [11] locomotion benchmarks. With this starting point, achieving a specific gait characteristic requires iterations of heuristic reward-function adjustments, based on observed RL performance, until arriving at a desired behavior. This approach can be tedious when it works and is unreliable as a general framework. Other reference-free approaches structure the reward around a specific type of locomotion behavior [26] without being easily extended to other behaviors.

The first contribution of our work is to present a principled framework for designing

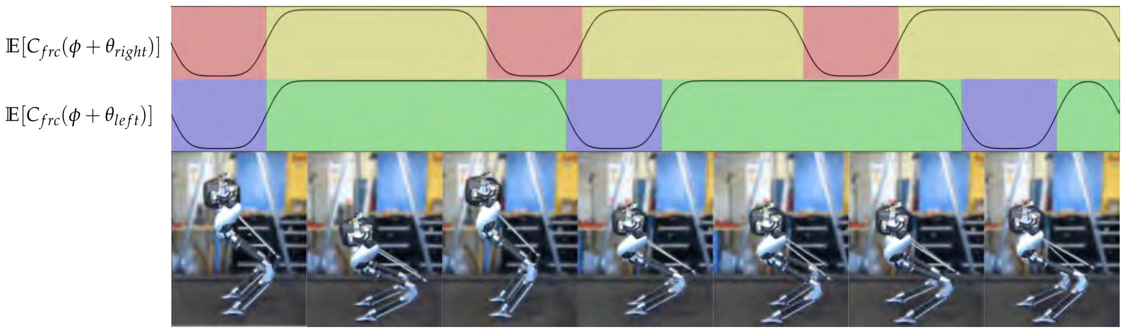


Figure 3.2: A series of images showing a neural network policy controlling Cassie and continuously transitioning from hopping to galloping to walking. We present a simple reward design paradigm which makes use of probabilistic intervals to apply cost functions at specific times, allowing policies to learn all common bipedal gaits exhibited by animals in nature.

reward functions that can naturally capture all of the periodic bipedal locomotion gaits. We are motivated by the fact that all common bipedal gaits can be defined by periodic *swing phases* (foot swinging in the air) and *stance phases* (foot planted on the ground) for each foot [16]. A fundamental distinction between swing and stance phases is the complementary presence and absence of foot forces and foot velocities for a given foot. We can create principled reward functions based on this observation by using the magnitudes of foot forces and velocities such that during a swing phase, forces are penalized while velocities are allowed, prompting the policy to learn to lift the foot. Thus, our framework describes gaits as a sequence of periodic phases, each of which rewards or penalizes a particular measurement of the physical system. Our hypothesis is that this framework will allow for a more natural specification of reward functions that sufficiently constrain RL to learn the desired gait characteristics, while allowing for flexible adaptation to specific environmental disturbances.

Our second contribution is to demonstrate this framework for sim-to-real RL of all common bipedal gaits, including walking, running, galloping, skipping, and hopping, without using a motion capture dataset or reference trajectories. We train policies for each of these behaviors in simulation and successfully demonstrate them on hardware. Further, by providing the framework's gait parameters as a command input to the policy, we are able to learn a multi-gait policy which can hop, gallop, run, and walk.

3.2 Learning Bipedal Gaits with Periodic Reward Composition

3.2.1 Reinforcement Learning Framework

We formulate our problem in the framework of reinforcement learning (RL) [51], for which we assume basic familiarity. The world is modeled as a discrete-time Markov Decision Process (MDP) with continuous state space S, continuous action space A, transition function $T(s, a, s')$, and reward function $R(s, t)$. Here $T(s, a, s')$ gives the probability density over the next state s' after taking action a in state s, and $R(s, t)$ gives the non-stationary reward for being in state s at time step t.

A control policy is a possibly stochastic mapping $\pi(a \mid s)$ from states to actions, which dictates behavior. Given a policy the expected T-horizon discounted return is given by $J(\pi) = \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t R(S_t, t)\right]$, where $\gamma \in [0, 1]$ is a discount factor and S_t is a random variable representing the state at time t when following policy π under transition dynamics T. The goal of RL is to learn a policy π that maximizes $J(\pi)$ based on trial-and-error training experience in the world. In this work, we follow a sim-to-real RL paradigm where training is done in simulation to identify a policy, which is then used in the real-world.

3.2.2 Periodic Reward Composition

Since our framework is targeted toward periodic behaviors, we index time via a *cycle time* ϕ variable, which repeatedly cycles over a normalized time period of $[0, 1]$ at discrete time steps rather than increasing monotonically. Accordingly, the non-stationary reward function $R(s, \phi)$ is periodic and defined in terms of ϕ rather than absolute time.

As motivated in the introduction, our framework specifies rewards in terms of compositions of rewards on periodic intervals. We define the reward as a biased sum of n *reward components* $R_i(s, \phi)$, where each component $R_i(s, \phi)$ captures a desired characteristic of the gait during a particular phase.

$$R(s, \phi) = \beta + \sum_i R_i(s, \phi)$$

Each reward component $R_i(s, \phi)$ is a product of a *phase coefficient* c_i, a *phase indicator* $I_i(\phi)$, and a real-valued *phase reward measurement* $q_i(s)$ (e.g. norm of a foot force).

$$R_i(s, \phi) = c_i \cdot I_i(\phi) \cdot q_i(s)$$

The phase coefficient c_i is a scalar whose sign determines the effect that the phase measurement $q_i(s)$ has on the total reward during cycle times when the reward component is active. The phase indicator function $I_i(\phi)$ is a binary-valued random variable denoting whether the target phase is active or not at cycle time ϕ. In this work, the distribution of each I_i is described by random variables A_i and B_i representing the start and end times of the period respectively. Since A_i and B_i represent intervals on a cycle, we use Von Mises distributions (approximations of the wrapped Normal distribution) described by the parameter tuple (a_i, b_i, κ), which gives the means a_i and b_i and shared variance parameter κ. The distribution of the binary phase indicator $I_i(\phi)$ is then simply:

$$P\left(I_i(\phi) = x\right) = \begin{cases} P(A_i < \phi < B_i) & \text{if } x = 1 \\ 1 - P(A_i < \phi < B_i) & \text{if } x = 0 \end{cases} \tag{3.1}$$

$$\text{where } P(A_i < \phi < B_i) = P(A_i < \phi)(1 - P(B_i < \phi))$$
$$A_i \sim \mathbf{\Phi}(2\pi a_i, \kappa) \text{ and } B_i \sim \mathbf{\Phi}(2\pi b_i, \kappa)$$
$$\mathbf{\Phi} \text{ is the Von Mises distribution}$$

Note that this formulation allows for uncertainty about the exact start and end of each phase to be captured via the variance parameter κ of the Von Mises distributions. This has a smoothing effect on the reward function at phase boundaries, which we have found to usefully encourage more stable and consistent learning. While we could directly use this probabilistic reward function $R(s, \phi)$ for RL training by sampling from the reward distribution at each step, we instead apply RL to the deterministic expectation of $R(s, \phi)$.

$$\mathbb{E}\left[R(s, \phi)\right] = \sum_i^n c_i \cdot \mathbb{E}[I_i(\phi)] \cdot q_i(s) + \beta$$

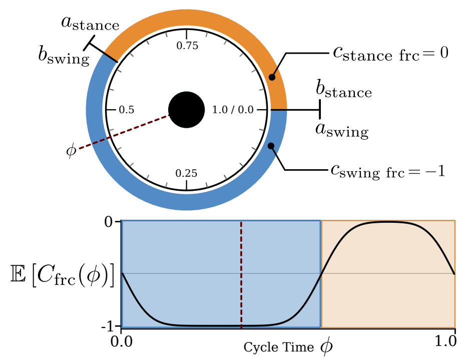


Figure 3.3: The circular intervals of two phases, swing and stance, are shown on a polar plot. The expected sum of the phase indicators and phase coefficients for foot force $C_{\text{frc}}(\phi)$ is shown below.

Due to the linearity of expectations, for any policy π the expected cumulative return $J(\pi)$ is the same regardless of whether we use stochastic rewards or their expectations.

3.2.3 Describing Bipedal Gaits

For simplicity, we begin by describing repeatedly lifting and placing a single foot, or equivalently, cycling between swing and stance phases with our framework. As our phase reward measurements, we select the norm of foot force $q_{\text{frc}}(s)$ and the norm of foot velocity $q_{\text{spd}}(s)$. During the swing phase we want to penalize foot forces and ignore foot velocities, so we choose $c_{\text{swing frc}} = -1$ and $c_{\text{swing spd}} = 0$. Similarly, we choose $c_{\text{stance spd}} = -1$ and $c_{\text{stance frc}} = 0$ to penalize foot velocities and ignore foot forces during the stance phase. We constrain the swing and stance phases to follow immediately after one another and together last the entire cycle time by defining a ratio $r \in (0, 1)$ and setting the intervals for both phases such that the swing phase lasts length r, while the stance phase lasts length $1 - r$ and starts directly afterwards. Finally, by choosing a common scale κ, we can define the indicator functions $I_{\text{frc}}(\phi)$ and $I_{\text{spd}}(\phi)$ by using Equation 3.1.

For convenience, we define $C_{\text{frc}}(\phi)$ and $C_{\text{spd}}(\phi)$ as the expected sum of the product of all the phase indicators and phase coefficients for the swing and stance phases:

$$\begin{aligned}\mathbb{E}\left[C_{\text{frc}}(\phi)\right] = &\quad c_{\text{swing frc}} \cdot \mathbb{E}[I_{\text{swing frc}}(\phi)] \\ &+ c_{\text{stance frc}} \cdot \mathbb{E}[I_{\text{stance frc}}(\phi)] \\ \mathbb{E}\left[C_{\text{spd}}(\phi)\right] = &\quad c_{\text{swing spd}} \cdot \mathbb{E}[I_{\text{swing spd}}(\phi)] \\ &+ c_{\text{stance spd}} \cdot \mathbb{E}[I_{\text{stance spd}}(\phi)]\end{aligned}$$

Refer to Fig.3.3 for a visual explanation of the phase start and end times, ϕ, and the expected value of C_{frc}. For more complicated behaviors, we find that visualizing $C_{\text{frc}}(\phi)$ can be useful for understanding how the reward function changes over the cycle to guide the learning of a particular behavior.

Putting the force and speed components together, the expected overall reward for repeatedly lifting and placing a single foot is

$$\begin{aligned}\mathbb{E}\left[R_{\text{unipedal}}(s, \phi)\right] = &\quad \mathbb{E}\left[C_{\text{frc}}(\phi)\right] \cdot q_{\text{frc}}(s) \\ &+ \mathbb{E}\left[C_{\text{spd}}(\phi)\right] \cdot q_{\text{spd}}(s)\end{aligned}$$

Bipedal gaits are behaviors where the left and right feet both follow the same sequence of phases described above, but offset relative to each other in phase time. For instance, in a walking behavior the timings of the swing and stance phases are shifted apart by half of the period length (one leg in swing, the other in stance), while a hopping behavior is one where both feet synchronously enter the swing and stance phases. To expand the simple behavior of lifting and placing a single foot into the full spectrum of bipedal gaits, we need only introduce two *cycle offset* parameters $\theta_{\text{left}}, \theta_{\text{right}}$ which define the exact timing shift between the identical sequence of behavioral phases for the left and right feet, and differentiate between the norms of left and right foot forces, $q_{\text{left frc}}(s),\ q_{\text{right frc}}(s)$ and norms of left and right foot velocities $q_{\text{left spd}}(s),\ q_{\text{right spd}}(s)$. The expected overall reward for a bipedal behavior is

$$\begin{aligned}\mathbb{E}[R_{\text{bipedal}}(s,\phi)] = \quad & \mathbb{E}[C_{\text{frc}}(\phi+\theta_{\text{left}})]\cdot q_{\text{left frc}}(s) \\ & +\mathbb{E}[C_{\text{frc}}(\phi+\theta_{\text{right}})]\cdot q_{\text{right frc}}(s) \\ & +\mathbb{E}[C_{\text{spd}}(\phi+\theta_{\text{left}})]\cdot q_{\text{left spd}}(s) \\ & +\mathbb{E}[C_{\text{spd}}(\phi+\theta_{\text{right}})]\cdot q_{\text{right spd}}(s)\end{aligned} \tag{3.2}$$

Introducing $\theta_{\text{left}}, \theta_{\text{right}}$ for two phase behaviors allows us to define reward functions for walking, galloping, and hopping. Additionally, by using four phases rather than two in each component, we can derive a skipping reward function, as shown in Fig. 3.4.

3.3 Method

Network Architecture and Action Space: For all policies, we use a Long Short-Term Memory neural network [23] with two recurrent hidden layers of size 128 each, and a simple linear output projection of size 30, corresponding to 10 desired joint positions, and 10 sets of PD gains, as seen in Fig. 4.1. The desired joint positions are added to a set of constant offsets corresponding to a neutral standing position, such that an output

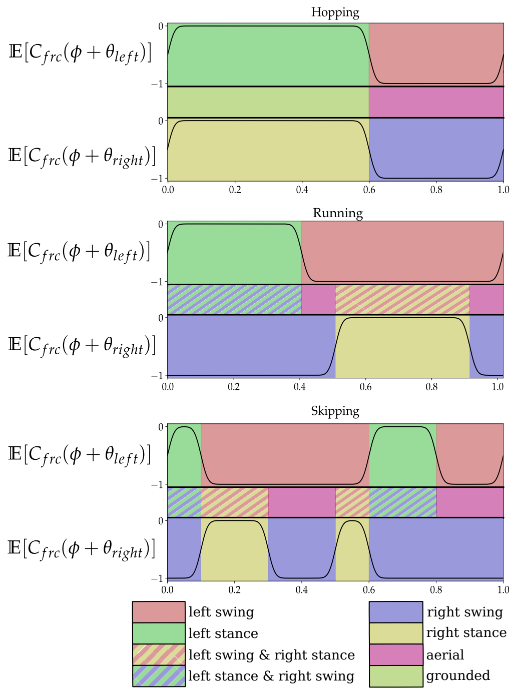


Figure 3.4: Plot of the expected value of the force phase coefficient for each foot for some example bipedal gaits. In hopping there is a $|\theta_{\text{left}} - \theta_{\text{right}}| \approx 0$ shift between the left and right feet. For walking and running, $|\theta_{\text{left}} - \theta_{\text{right}}| \approx 0.5$, with transitional shifts resulting in galloping. Four phases are necessary for describing skipping, as well as a $|\theta_{\text{left}} - \theta_{\text{right}}| \approx 0.5$ shift between the left and right feet. The shaded region between the phase coefficient plots show the hybrid phases from combining the left and right foot behaviors together.

vector of zeroes results in a standing pose. Similarly, the P gains and D gains are also summed with a fixed 'neutral' set of gains. The policy is evaluated at 40Hz, and the robot's PD controllers are run at 2000Hz. A similar system is used in [61] and [46], though without PD gain deltas.

State Space: To prevent the reinforcement learning environment from becoming a non-stationary Markov decision process, some information about the periodic reward function must be present in the state provided to the policy. Specifically, the policy must receive some sort of encoding of the current cycle time ϕ, an encoding of the cycle offset parameters, $\theta_{\text{left}}, \theta_{\text{right}}$, and information about the start and end times of the phases of the phase reward components.

In order to encode the current cycle time ϕ and the cycle offset parameters $\theta_{\text{left}}, \theta_{\text{right}}$, we condition the policies on two clock inputs:

$$p = \left\{\sin\left(2\pi(\phi + \theta_{\text{left}})\right), \sin\left(2\pi(\phi + \theta_{\text{right}})\right)\right\}$$

To encode information about the start and end times of the phases into the state, we derive a vector of *ratios* from the sequence of phase timings; each ratio represents the proportion out of the total period that a phase occupies. For instance, a phase j with start time $a_{i,j=0.3}$ and end time $b_{i,j} = 0.7$ should occupy 40% of the total period time (plus or minus some uncertainty); thus, we can derive a ratio $r_j = 0.4$. Each phase's ratio is calculated and provided to the policy.

Thus, the policy's input consists of:

$$X_t = \begin{cases} \hat{q},\ \hat{\dot{q}} & \text{robot state} \\ \dot{x}_{\text{desired}}, \dot{y}_{\text{desired}} & \text{desired velocity} \\ r, p & \text{phase ratios and clock inputs} \end{cases}$$

Where $\hat{q}, \hat{\dot{q}}$ are estimates of the pelvis orientation, rotational velocity, joint positions and joint velocities. $\dot{x}_{\text{desired}}$ and $\dot{y}_{\text{desired}}$ are speed commands randomized during training, and manually controlled by the user during evaluation.

Single Behavior Reward Formulation: We use the set of reward components from R_{bipedal} defined in Equation 3.2 to learn single-gait policies. In addition, we also use a variety of auxiliary cost components to further constrain the path of viable exploration towards stable locomotion and facilitate successful sim-to-real transfer as well as command the policy to match a desired orientation, forward speed, and sidespeed. These rewards do not need to vary as a function of time, and can be seen as a special case of the reward component framework, wherein the random variable c_i has only one phase, and thus one possible value (in this case, -1, to show that we wish to penalize these quantities q_i for the entire period).

$$\begin{aligned}
R_{\text{cmd}}(s) = \quad & (-1) \cdot q_{\dot{x}}(s) \\
+ \quad & (-1) \cdot q_{\dot{y}}(s) \\
+ \quad & (-1) \cdot q_{\text{orientation}}(s) \\
R_{\text{smooth}}(s) = \quad & (-1) \cdot q_{\text{action diff}}(s) \\
+ \quad & (-1) \cdot q_{\text{torque}}(s) \\
+ \quad & (-1) \cdot q_{\text{pelvis acc}}(s)
\end{aligned}$$

$$\mathbb{E}[R(s,\phi)] = \mathbb{E}[R_{\text{bipedal}}(s,\phi)] + R_{\text{smooth}}(s)] + R_{\text{cmd}}(s) + \beta \tag{3.3}$$

Where $q_{\dot{x}}$ and $q_{\dot{y}}$ are measures of error between the commanded velocity and the actual pelvis velocity, and $q_{\text{orientation}}$ is the negative exponent of quaternion difference between the pelvis orientation and an orientation which faces straight, which can be used to change the heading of the robot. $q_{\text{pelvis acc}}$ is a cost for aggressive pelvis motion, which helps reduce noise in the state estimator, while $q_{\text{action diff}}$ is a cost for aggressive actions and q_{torque} is an overall joint torque cost to encourage efficient motions. For general policies which can learn and transition between a continuum of bipedal gaits, we specify additional reward terms which are discussed in Section 4.4.

Multi-Behavior Reward Function:

When learning the continuum of 2-beat gaits, we find that policies often learn to stutter rather than hop with their feet together, which is not observed when training hopping standalone. To combat this, we introduce a *hop symmetry* term, a cost which becomes active when the cycle offsets of both legs match closely and punishes large distances between the feet in the sagittal and transverse planes, err_{sym},

$$q_{\textit{hop sym}}(s) = 1 - \exp(-\text{err}_{\text{sym}} \exp(-5\,|sin(2\pi(\theta_{\text{left}} - \theta_{\text{right}}))|))$$

To learn a standing behavior in addition to the rest of the 2-beat gaits, the reward function must be modified to reflect our wish for the policy to remain very still when commanded. First, we define the standing region of gait parameter space as one where the swing-to-stance phase ratio is close to 0.

We define $\omega = (1 + \exp(-50(r_{\text{swing}} - 0.15)))^{-1}$, which is a coefficient close to one during normal locomotion (when the swing and stance ratios are in the range [0.35, 0.7], and close to zero during standing (where the swing phase ratio is close to zero). Now we define an additional standing cost, which applies an additional action difference cost and a foot symmetry cost (similar to the hopping symmetry) when the command inputs are in the standing region.

$$q_{\text{standing cost}}(s) = 1 - \exp(-((\text{err}_{\text{sym}}) + 20 q_{\text{action diff}}(s)))$$

Now we define the full reward for a generic policy which includes standing:

$$\begin{aligned}
\mathbb{E}[R_{\text{multi}}(s,\phi)] = \quad & 0.400 \cdot \mathbb{E}[R_{\text{bipedal}}(s,\phi)] \\
+ \quad & 0.300 \cdot \mathbb{E}[R_{\text{cmd}}(s)] \\
+ \quad & 0.100 \cdot \mathbb{E}[R_{\text{smooth}}(s)] \\
+ \quad & 0.100 \cdot (\omega - 1) \cdot q_{\text{standing cost}}(s) \\
+ \quad & 0.100 \cdot (-1) \cdot q_{\text{hop sym}}(s) \\
+ \quad & 1
\end{aligned}$$

Quantity q_i	Detail
$q_{\text{left/right frc}}$	$1 - \exp(-\omega \lVert \text{raw\_foot\_frc} \rVert_2^2 / 100)$
$q_{\text{left/right spd}}$	$1 - \exp(-2 \cdot \omega \lVert \text{raw\_foot\_spd} \rVert_2^2)$
$q_{\dot{x}}$	$1 - \exp(-2 \cdot \omega \lvert \dot{x}_{\text{desired}} - \dot{x}_{\text{actual}} \rvert)$
$q_{\dot{y}}$	$1 - \exp(-2 \cdot \omega \lvert \dot{y}_{\text{desired}} - \dot{y}_{\text{actual}} \rvert)$
$q_{\text{orientation}}$	$1 - \exp(-3 \cdot (1 - ((\text{quat}_{\text{actual}})^T (\text{quat}_{\text{des}}))^2)$
$q_{\text{action diff}}$	$1 - \exp(-5 \cdot \lVert a_t - a_{t-1} \rVert)$
q_{torque}	$1 - \exp(-0.05 \cdot \lVert \tau \rVert)$
$q_{\text{pelvis acc}}$	$1 - \exp(-0.10 \cdot (\lVert \text{pelvis}_{\text{rot}} \rVert + \lVert \text{pelvis}_{\text{acc}} \rVert)$

Table 3.1

Where $\dot{x}_{\text{actual}}$ and $\dot{y}_{\text{actual}}$ are the current forward and lateral speeds of the pelvis. $\text{quat}_{\text{actual}}$ and quat_{des} are the actual pelvis orientation and the desired pelvis orientation, in quaternion format. a_t is the current timestep's action, and a_{t-1} is the previous timestep's action. τ is the net torque applied to all joints across the robot. $\text{pelvis}_{\text{rot}}$ is the rotational velocity of the pelvis, and $\text{pelvis}_{\text{acc}}$ is the translational acceleration of the pelvis. Note that certain terms are multiplied by ω, signifying that these terms should not be respected by the policy when $\omega \approx 0$ (i.e., the policy is somewhere inside the standing region of the gait parameter space), to prevent the policy from (for example) attempting to match a desired speed while standing. The choice of $1 - \exp(-|x|)$ as a kernel function was influenced by a desire to have a reward bounded by 0 and 1.

Dynamics Randomization: In order to facilitate successful sim-to-real transfer, we use dynamics randomization [35] [52] to expose the policies to a wide variety of possible real-world dynamics. Specifically, we randomize the execution rate of the policy, the mass and damping of the joints, the friction of the ground, the slope of the ground, and joint position encoder noise. Details on the ranges and distributions used to randomize these parameters can be found in Table 4.3. These parameters are randomized at the beginning of every rollout during training and remain constant throughout each rollout.

Parameter	Unit	Range
Joint damping	Nms/rad	$[0.3, 4.0]$ $\times$ default values
Joint mass	kg	$[0.5, 1.5]$ $\times$ default values
Ground Friction	–	$[0.35, 1.1]$
Ground Slope	rad	$[-0.03, 0.03]$
Joint Encoder Offset	rad	$[-0.05, 0.05]$

Table 3.2: The ranges for randomization of several dynamics parameters during training. We use a uniform distribution over the given ranges for all listed parameters.

Proximal Policy Optimization: To train our policies, we use a common model-free reinforcement learning algorithm known as Proximal Policy Optimization (PPO)

[43]. More specifically, we use the recurrent adaptation described in **??**, which samples batches of trajectories from a replay buffer, rather than batches of individual timesteps. In our case, we use a recurrent critic with exactly the same dimensions as the actor with the exception of the final linear output vector, which in the case of the critic is a scalar. We use a fixed exploration action noise, with standard deviation e^{-1}. In addition, we incorporate a mirror loss term [1] into our policy gradient algorithm with the aim of achieving more symmetric locomotion behavior.

3.4 Experimental Results

Training Details. Policies were trained using PPO with a batch size of 32 trajectories of up to 300 timesteps each, a learning rate of 0.0001 for both the actor and critic, a replay buffer of 50,000 samples, and 4 epochs per iteration. Training was terminated after 150,000,000 samples, which took between 24 and 36 hours per policy. We use the *cassie-mujoco-sim* [2] simulator, based on MuJoCo [55], to simulate Cassie during training.

Single-Gait Policy Results. We found that it was straightforward to use the probabilistic framework to train policies to learn standalone gaits, such as hopping, walking, running, or even skipping. These behaviors can be learned simply by holding the values of ratios r and cycle offset parameters $\theta_{\text{left}}, \theta_{\text{right}}$ to be constant throughout training (we refer the reader to Fig. 3.4 for examples of gait specifications). Videos of single-gait policies which learn skipping hopping, and walking can be found in our submission video.

Multi-Gait Policy Results By keeping the cycle offsets fixed and varying the phase ratios over some range during training, we are able to train policies that can learn to hop with more or less time in the air, and policies that can transition from walking to running. However, learning to transition between gaits by varying both the cycle offsets and phase ratios during training appears to be a challenging learning problem: policies which are trained in this fashion can end up asymmetrically walking instead of hopping, or learn other undesirable behaviors that resemble a fusion of all the different commanded gaits. To avoid these issues, we introduce additional reward terms called *transition penalties* to further distinguish each of the desired behaviors from each other during training. Transition penalties are behavior-specific costs which are only active when specific behaviors are being commanded. For hopping, which is commanded when cycle offsets are very close to each other, we activate a cost for constraining the feet positions to be close together. Similarly, by adding another transition penalty for standing, we can train a generic controller to transition between all steady state two-beat gaits and standing in place. Full details of the reward function for generic controllers are described in detail in section 3.3.

Outdoor Experiments We demonstrate the robustness of policies learned with our approach in a variety of outdoor experiments, shown in our submission video [1]. Given that the robot is effectively blind to the external world, the behaviors we observe in response to disturbances are surprisingly robust. We show the multi-gait policy hopping on and off a sidewalk, walking with one foot elevated on a curb, and running over small bumps in its path. The policy is also shown smoothly transitioning between all of the 2-

[1] youtu.be/4DnxV9lko_U

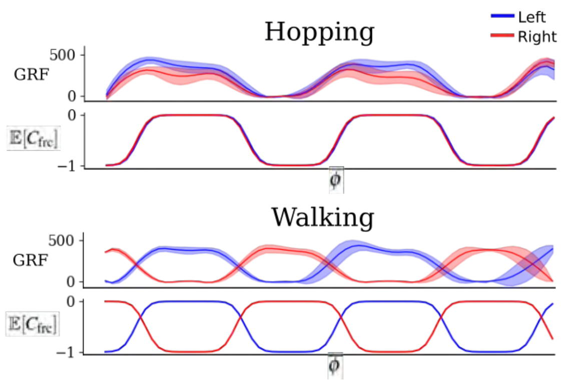


Figure 3.5: Comparison between the mean measured simulation ground reaction forces (GRFs) in newtons and the corresponding expected values of C_{frc} over multiple policies. When the expectation is close to -1, the policies refrain from applying foot forces. When the expectation is close to 0, the policies apply foot forces.

beat gaits while moving forward on a crowned road, and while turning in a small circle on a turf field. We also show a multi-gait policy descending down 5 steps of stairs while in a running gait. In the staircase and sidewalk tests, we observe the foot slipping off of small ledges and being compliant to the slope of the ground. This indicates a priority on force profile, as opposed to position only.

3.5 Conclusion

In this chapter, we introduced a probabilistic reward framework which allows for learning of all of the common bipedal gait behaviors observed in animals. This framework requires no reference trajectories, enabling policies to explore a rich space of possible interaction with the world during training without artificial constraints to some arbitrary trajectory through space. We showed that not only are we able to train policies to learn all common bipedal gaits individually, including walking, running, hopping, galloping, and skipping, but also that we can learn all 2-beat behaviors on single policies and transition between them continuously, even while in motion. Some questions which remain unanswered include how easily this framework might be applied to the space of possible quadrupedal gaits, or other bipedal morphologies. A modified version of this framework could also be applied to aperiodic motions to learn one-off behaviors.

Chapter 4: Blind Bipedal Stair Traversal via Sim-to-Real Reinforcement Learning

4.1 Introduction

In order to be useful in the real world, bipedal and humanoid robots need to be able to climb and descend stairs and stair-like terrain, such as raised platforms or sudden vertical drops, which are common features of human-centric environments. The ability to robustly navigate these environments is crucial to getting robots to work with and alongside humans safely. Achieving this level of robustness on a bipedal platform is no easy task; while other platforms such as quadrupedal robots benefit from inherent stability due to multiple points of contact with the ground at a given time and the ability to stop and stand like a table, bipedal robots such as Cassie rely entirely on dynamic stability (essentially always existing in a state of falling). On stair-like environments, this is especially apparent due to the difficulty of recovery from missteps with only two legs.

By contrast, robots with quadrupedal morphologies have been able to use proprioception alone to negotiate stairs [28, 9], and hexapedal robots have even been able to use open-loop control to ascend and descend stairs [31]. While planar bipedal robots have been shown to be able to reject disturbances like large unexpected dropsteps [33], the vast majority of approaches seeking to enable such robots to negotiate stairs in the

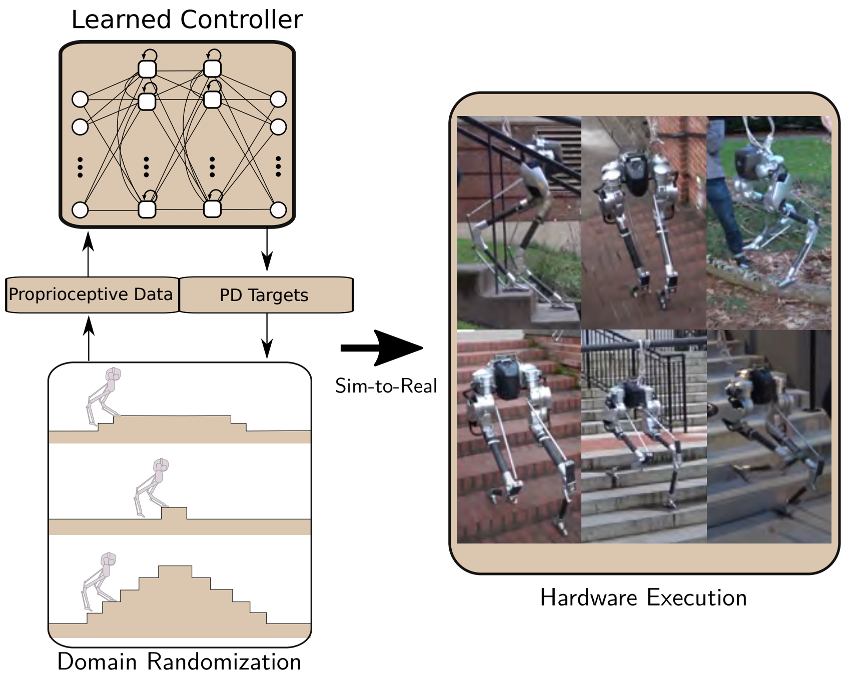


Figure 4.1: In this chapter, we investigate the limits of blind bipedal locomotion. We present a training pipeline which produces policies capable of blindly ascending and descending stairs in the real world. These policies learn proprioceptive reflexes to reject significant disturbances in ground height, resulting in highly robust behavior to many real-world environments.

real world require either accurate vision systems [18, 4, 30] or operation in a carefully controlled laboratory environment [12, 41, 14], meaning the robot is localized through a known start location or the stairs are designed in tandem with robot morphology.

However, robots must be able to operate outside of controlled laboratory conditions and handle the massive variety of conditions in the real world. This goal is not compatible with a complete reliance on exteroceptive sensors such as RGB and depth cameras for accurate terrain estimation, which introduce fragility to real world conditions [15]. For instance, cameras may be unreliable if exposed to occlusion, fog, or varying lighting conditions. Further, integrating a state-of-the-art computer vision system into a high-speed controller is technically difficult, especially on a computationally limited platform like a mobile robot. For practical purposes, underlying controllers should be as robust as possible while relying on as little information about the world as possible. Ideally, a bipedal robot should be able to traverse as much of the entire breadth of human environments as possible using proprioception, while relying on exteroceptive sensing for further efficiency and high-level planning (and being robust to mistaken perception). This begs the question: how robust can a blind bipedal robot be?

Reinforcement learning (RL) based approaches have begun to show significant promise at robust real-world legged locomotion [28, 60]. Unlike optimization or heuristic-based control methods which rely on prescribed ground contact schedules or force-based event detection, RL can produce control policies which learn proprioceptive reflexes and strategies for dealing with unexpectedly early or late contact and rough terrain through exposure to a variety of disturbances during training. However, the limits of this approach are unclear and prior work has not been demonstrated on the scale and variety of disturbances involved in stair-like terrain.

In this work, we show that robust proprioceptive bipedal control for complex stair-like terrain can be learned via an existing RL framework with surprisingly little modification. In particular, the only adjustment needed is the terrain randomization used during training, where we define a distribution over upward and downward going stairs including variation in height, width, and slope of the contact planes. Learning on this distribution allows for blind locomotion up and down unknown stairs as well as handling more general stair-like terrain characteristics, e.g. logs, curbs, dropoffs, etc. The learned controller is demonstrated in simulation and a variety of real-world settings. To our knowledge, this is the first demonstration of its kind and suggests the continued exploration into the limits of robust proprioceptive bipedal control.

4.2 Reinforcement Learning Formulation

We follow a sim-to-real reinforcement learning (RL) approach for learning bipedal locomotion and assume basic familiarity with RL [51]. In the general RL setting, at each discrete time step t the robot control policy π receives the current state s_t and returns an action a_t, which is applied and results in a transition to the next state s_{t+1}. The state transition dynamics are unknown to the robot and are governed by a combination of environmental conditions, such as terrain type, and the robot dynamics. In addition, during learning, each state transition is associated with a real-valued reward r_t. The reward is governed by the application goals to encourage the desired behavior during learning. The RL optimization objective considered in this work is to learn a policy through interaction with the environment that maximizes the expected cumulative discounted reward over a finite-horizon T. That is, find a policy π that maximizes: $J(\pi) = \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t R_t\right]$, where $\gamma \in [0,1]$ is the discount factor and R_t is a random variable representing the reward at time t when following π from a state drawn from an initial state distribution.

For complex environments, RL typically requires large amounts of training experience to identify a good policy. Further, for biped locomotion, the training will involve many falls and crashes, especially early in training. Thus, training from scratch in the real-world is not practical and we instead follow a sim-to-real RL paradigm. Training is done completely in a simulation environment, with dynamics randomization (see below), and the resulting policy is then used in the real-world.

In the remainder of this section, we detail the specific sim-to-real RL formulation used in this work, which follows the work in Chapter 3 on learning different biped gaits over flat terrain. Surprisingly, only minimal changes were required to enable policy learning for the much more complex stair-like terrains of this paper[1] In particular, the only major modification required was the randomized domain generation of stair-like rather than mostly flat terrain as discussed later in Section 4.3; no novel stair-specific reward terms were needed.

4.2.1 State Space

The state s_t that is input to the control policy at each time step includes three main components. First, the state contains information about the robot's instantaneous physical state, including the pelvis orientation in quaternion format, the angular velocity of the pelvis, the joint positions, and the joint velocities. The second component of s_t is composed of command inputs, which come from a human operator. These commands are subject to randomization during training to give the policies a wide breadth of experience attempting to traverse stairs over a variety of speeds and approach angles. Details of this randomization can be seen in Table 4.1.

The third component includes two cyclic clock inputs, each corresponding to a leg

[1]This was only discovered after a careful ablation analysis of our first success on stair-like terrain, which originally included seemingly necessary modifications to prior work, such as more complex reward functions and state features.

Command	Probability of Change	Range
Forward Speed	1/300	[-0.3m/s, 1.5m/s]
Sideways Speed	1/300	[-0.3m/s, 0.3m/s
Turn Rate	1/300	[-90deg/s, 90deg/s]

Table 4.1: At each timestep, each command input to the policy is subject to a 1/300 probability of being altered. When this occurs, a new command is sampled from a uniform distribution parameterized by the rightmost column. Given that maximum episode length is 300 discrete timesteps, this means each command will change once on average per episode.

of the robot, p:

$$p = \begin{cases} \sin\left(2\pi(\phi_t + 0.0)\right) \\ \sin\left(2\pi(\phi_t + 0.5)\right) \end{cases} \tag{4.1}$$

Here ϕ_t is a phase variable which increments from 0 to 1, then rolls back over to 0, keeping track of the current phase of the gait. The constant offsets 0.0 and 0.5 are phase offsets used to make sure that the left and right legs are always diametrically opposite of each other in terms of phase during locomotion.

4.2.2 Action Space

The output action a_t of the control policy at each time step (running at 40Hz) is an 11 dimensional vector with the first 10 entries corresponding to PD targets for the joints, each of which are fed into a PD controller for each joint operating at 2KHz. Prior work has found it advantageous to learn actions in the PD target space rather than directly learning the higher-rate actuation commands [38].

The final dimension of a_t is a clock delta δ_t (refer to 4.2.1 for information on clocks), which allows the policy to regulate the stepping frequency of the gait. Intuitively, this allows the controller to choose an appropriate stepping frequency for a particular gait, command, and terrain. Specifically, the phase variable ϕ in the state representation (Section 4.2.1) is updated at each timestep t by,

$$\phi_{t+1} = \text{fmod}(\phi_t + \delta_t, 1.0). \tag{4.2}$$

This delta is bounded in a way such that the policy can choose to regulate the gait cycle between 0.5x and 1.5x the nominal stepping frequency (which is approximately one gait cycle every 0.7 seconds). While this component is included in the control policy action, it does not appear to have a large impact on performance and the learned policy does not vary δ_t much in response to disturbances. We suspect that future ablation analysis will show that it is not important for performance on the real robot.[2]

4.2.3 Reward Function

We use the method introduced in Chapter 3 to specify our reward function. To briefly review this method, we desire a reward framework which allows for penalizing the policy for large magnitudes of some quantities of the environment at certain times, while permitting those quantities to be large at other times. We designate foot forces and foot velocities as two such quantities; punishing foot forces incentivizes the policy to lift the foot, while punishing foot velocities incentivizes the policy to place the foot. We add additional cost terms on top of these foundational reward terms, including a cost in-

[2]We leave this as a hypothesis here, since we have not had the resources to perform an ablation study.

centivizing the policy to match a translational velocity and orientation. We also employ costs which encourage smooth actions, energy efficiency, and to reduce pelvis shakiness. As in Chapter 3, we do not rely on expert reference trajectories to learn behaviors.

Reward Function

To briefly review the approach taken in Chapter 3, we wish to take advantage of the complementary nature of foot forces and foot velocities during locomotion to construct a reward function which will punish one and allow the other, and vice versa, at key intervals during the gait. We use a probabilistic framework to represent uncertainty around the timings of these intervals. More specifically, we make use of a binary-valued random indicator function $I_i(\phi)$ for each quantity q_i which we wish to penalize at some time during the gait cycle. This indicator function is likely to be 1 during the interval in which it is active, and likely to be 0 during intervals in which it is not active. The distribution of this binary-valued random function is defined via the Von Mises distribution; for a more comprehensive description, see [45]. In addition, rather than use the actual random variable in the reward we instead opt to use its expectation for more stable learning; see Fig. 4.2 for a plot of this expectation.

Our full reward function is as follows;

$$R(s, \phi) = 1 - \mathbb{E}[\rho(s, \phi)] \tag{4.3}$$

Which is to say, our reward is the difference of a bias and the expectation of a probabilistic penalty term $\rho(s, \phi)$ as described in Chapter 3. See Table 4.2 for detailed information on the exact quantities and weightings used.

Weight	Cost Component
0.140	$1 - \mathbb{E}[I_{\text{left force}}(\phi)] \cdot \exp(-.01\|F_l\|)$
0.140	$1 - \mathbb{E}[I_{\text{right force}}(\phi)] \cdot \exp(-.01\|F_r\|)$
0.140	$1 - \mathbb{E}[I_{\text{left velocity}}(\phi)] \cdot \exp(-\|v_l\|)$
0.140	$1 - \mathbb{E}[I_{\text{right velocity}}(\phi)] \cdot \exp(-\|v_r\|)$
0.140	$1 - \exp(-\epsilon_o)$
0.140	$1 - \exp(-\lvert\dot{x}_{\text{desired}} - \dot{x}_{\text{actual}}\rvert)$
0.078	$1 - \exp(-\lvert\dot{y}_{\text{desired}} - \dot{y}_{\text{actual}}\rvert)$
0.028	$1 - \exp(-5 \cdot \|a_t - a_{t-1}\|)$
0.028	$1 - \exp(-0.05 \cdot \|\tau\|)$
0.028	$1 - \exp(-0.1(\|\text{pelvis}_{\text{rot}}\| + \|\text{pelvis}_{\text{acc}}\|))$

Table 4.2: The cost terms which are summed together to compose the expected penalty, $\mathbb{E}[\rho(s, \phi)]$. Terms involving an expectation of a variable $I_i(\phi)$ vary over the course of the gait cycle, with the goal of penalizing foot forces and foot velocities at key intervals to teach the policy to lift and place the feet periodically in order to walk. Other terms exist for the sake of commanding the policy to move forward, backwards, or sideways, or turn the robot to face a desired heading. Finally, the remaining terms exist to reduce shaky behaviors which are unlikely to work well on hardware.

We define F_l and F_r as the vectors of translational forces applied to the left and right foot, and v_l and v_r similarly as the vectors of left and right foot velocities. To maintain a steady orientation, an orientation error ϵ_o is used, which is equal to,

$$\epsilon_o = 3(1 - \hat{q}^T q_{\text{body}})^2 + 10\left((1 - \hat{q}^T q_{\text{l}})^2 + (1 - \hat{q}^T q_{\text{r}})^2\right) \tag{4.4}$$

where q_{l} and q_{r} are the quaternion orientations of the left and right foot, q_{body} is the quaternion orientation of the pelvis, and $\hat{q}$ is a desired orientation (for our purposes, fixed to be always be facing straight ahead).

The quantities $\dot{x}_{\text{desired}}$ and $\dot{y}_{\text{desired}}$ correspond to a commanded translational speed, while $\dot{x}_{\text{actual}}$ and $\dot{y}_{\text{actual}}$ are the actual translational speed of the robot. The term $\text{pelvis}_{\text{rot}}$ represents the angular velocity while $\text{pelvis}_{\text{acc}}$ represents translational acceleration; these terms are used in the cost component to reduce the shakiness of locomotion behavior. The terms a_t and a_{t-1} refer to the current timestep's action and the previous timestep's action, and their use in the cost component is to encourage smooth behaviors. The term τ is the vector of net torques applied to the joints, and its use in the cost component is intended to encourage energy efficient gaits.

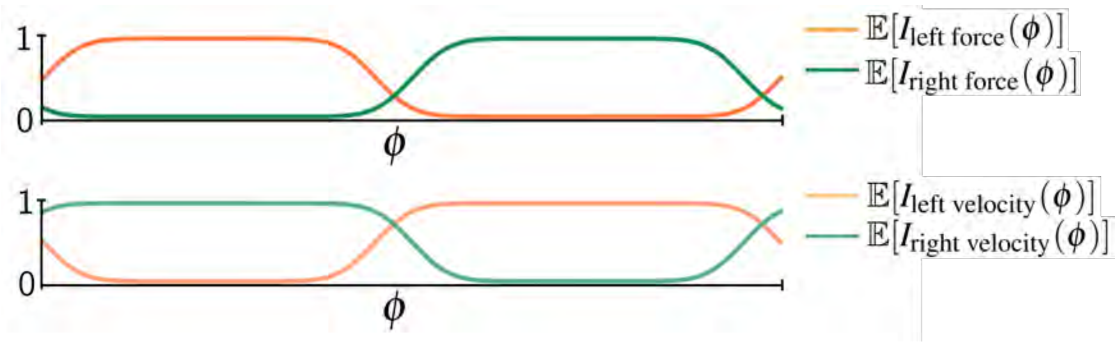


Figure 4.2: By alternatingly punishing foot forces during a 'stance' phase to teach the policy to lift the foot, and punishing foot velocities during a 'swing' phase to teach the policy to place the foot on the ground, we can construct a foundation on which to learn simple walking behavior. Following in the path of previous work, we define these cyclic coefficents as random indicator functions of the phase, and take their expectation.

4.2.4 Dynamics Randomization

In order to overcome any modeling errors that may be present in our simulated Cassie environment, we randomize several important quantities of the dynamics at the beginning of each episode during training as in previous work [34] and Chapter 3. These randomized parameters are listed in Table 4.3.

Parameter	Unit	Range
Joint damping	Nms/rad	$[0.5, 3.5] \times$ default values
Joint mass	kg	$[0.5, 1.7] \times$ default values
Ground Friction	–	$[0.5, 1.1]$
Joint Encoder Offset	rad	$[-0.05, 0.05]$
Execution Rate	Hz	$[37, 42]$

Table 4.3: To prevent overfitting to simulation dynamics and facilitate a smooth sim-to-real transfer, we employ dynamics randomization. The above ranges parameterize a uniform distribution for each listed parameters. Damping, mass, friction, and encoder offset are randomized at the beginning of each rollout, while execution rate is randomized at each timestep to mimic the effect of variable system delay on the real robot.

4.2.5 Policy Representation and Learning

We represent the control policy as an LSTM recurrent neural network [23], with two recurrent hidden layers of dimension 128 each. We opt to use a memory-enabled network because of previous work demonstrating a higher degree of proficiency in handling partially observable environments [**?**] [34] [45]. For ablation experiments, involving non-memory-based control policies, we use a standard feedforward neural network with two layers of dimension 300, with tanh activation functions, such that the number of parameters is approximately equal to that of the LSTM network.

For sim-to-real training of the policy, we use Proximal Policy Optimization (PPO) [43], a model-free deep RL algorithm. Specifically, we use a KL-threshold-termination variant, wherein each time the policy is updated, the KL divergence between the updated policy and the former policy is calculated and the update is aborted if the divergence is

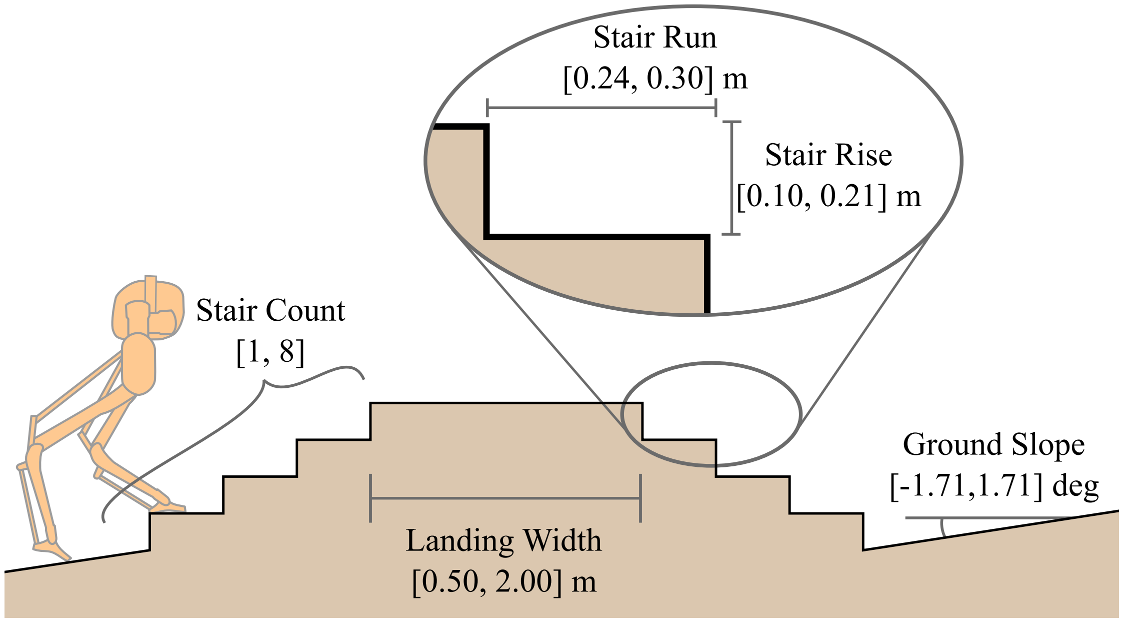


Figure 4.3: In order to ensure robustness over a variety of possible stair-like terrain, we randomize a number of parameters when generating stairs at the start of each episode in simulation. These parameters include the number of stairs, the height of each stair, the length of each stair, the length of the landing atop the stairs, and the slope of the ground immediately before and after the stairs.

too large. During training, we make use of a mirror loss term [1] in order to ensure that the control policy does not learn asymmetric gaits. For recurrent policies, we sample batches of episodes from a replay buffer as in [45], while for feedforward policies we sample batches of timesteps. Each episode is limited to be 300 timesteps, which corresponds to about 7.5 seconds of simulation time.

4.3 Terrain Randomization

Previous work on applying RL to Cassie has either trained on flat ground [60] [45] or on randomized slight inclines [Chapter 3]. Other work in applying deep RL has investigated employing a curriculum of rough terrains which become increasingly difficult as training progresses [28]. For the purpose of simplicity, we find that training on interactions with a randomized staircase without a curriculum is sufficient to learn robust behavior.

To this end, we train on a plane whose incline is randomized at the beginning of each rollout in the pitch and roll axes. This incline is between -0.03 radians and 0.03 radians. As part of the dynamics randomization, ground friction is randomized, increasing the potential difficulty of the environment. The starting position of the stairs are randomized at the beginning of each rollout, such that the episode can start with the policy already on top of the stairs, or with the stairs up to 10 meters in front of the policy. This is done in order to ensure that the policy is able to see lots of experience on flat or inclined ground, as well as on stairs.

The dimensions of the stairs are randomized within typical city code dimensions, with a per-step rise of between 10cm and 21cm, and a run of 24cm to 30cm. The number of stairs is also randomized, such that each set of stairs has between 1 and 8 individual steps. A small amount of noise ($\pm$ 1cm) is added to the rise and run of each step such that the stairs are never entirely uniform, to prevent the policy from deducing the precise dimensions of the stairs via proprioception and subsequently overfitting to perfectly uniform stairs.

4.4 Results

We trained four groups of policies, each containing five policies initialized with different random seeds. First, we trained a group of simple LSTM policies with stair terrain randomization; these are referred to in this section as **Stair LSTM**. To investigate the importance of memory, we trained a group of feedforward policies also with stair terrain randomization; we denote these **Stair FF**. We also trained a group of policies without stair terrain randomization, and denote these **Flat Ground LSTM**, to investigate the importance of the terrain randomization introduced in this work. The final group was trained with a simple additional binary input informing the policy whether or not stairs were present within one meter of the policy, referred to here as **Proximity LSTM**, in order to investigate the benefit of leaking information about the world to the policies.

Each policy was trained until 300 million timesteps were sampled from the virtual environment, simulated with MuJoCo [55]. Our selection of hyperparameters for the PPO algorithm includes a replay buffer size of 50,000 timesteps, a batch size of 64 trajectories for recurrent policies, and a batch size of 1024 timesteps for feedforward policies. Each replay buffer is sampled for up to five epochs, with optimization terminating early if the KL divergence reached the maximum allowed threshold of 0.02. We clear our replay buffer at the start of each iteration. We use the Adam [27] optimizer with a learning rate of 0.0005 for both the actor and critic, which are learned separately and do not share parameters.

4.4.1 Simulation

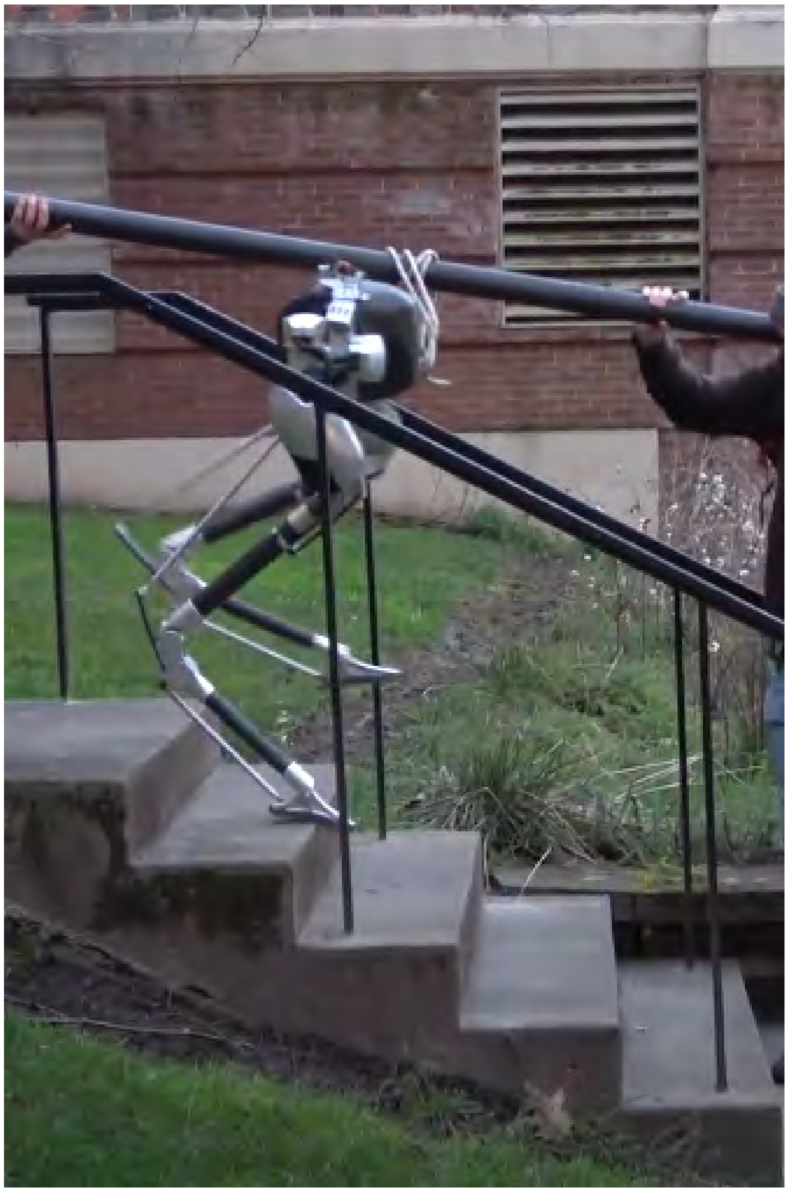


Figure 4.4: The learned policies exhibit a high degree of blind robustness to a variety of stair-like terrain, and can reliably ascend and descend stairs of typical dimensions found in human environments.

4.4.1.1 Probability of Successfully Ascending and Descending Stairs

To understand the importance of memory and terrain randomization, we evaluate three groups of policies on the task of successfully climbing and descending a set of stairs in simulation. We compare the performance of Stair FF, Stair LSTM, and Flat Ground LSTM policies on this task.

Specifically, we run 150 trials testing how often a policy is successfully able to climb a set of stairs with five steps, each with a tread of 17cm and a depth of 30cm (a typical real-world and relatively mild stair geometry). This should give us an estimate of how reliably each group of control policies can climb a flight of stairs that it approaches blindly. Success is defined as reaching the top of the flight of stairs without falling. We also apply this procedure for descending stairs, running 150 trials on stairs with the same dimensions, and record the rate at which each group of policies can reach the bottom without falling.

The results of these tests for three different training conditions is shown in figure 4.5. We note that the Stair LSTM policy has the highest overall probability of success. Nevertheless, the probability of success is dependent, in large part, on approach speed. The policies experience higher rates of failure at low speeds, where they may lack the momentum to propel themselves past poorly chosen foot placements. They also experience higher rates of failure at high speeds, possibly due to the more dynamic nature of high-speed gaits.

The Flat Ground LSTM policies, having never seen stair-like terrain during training, are unable to compensate and experiences a high rate of failure for both ascent

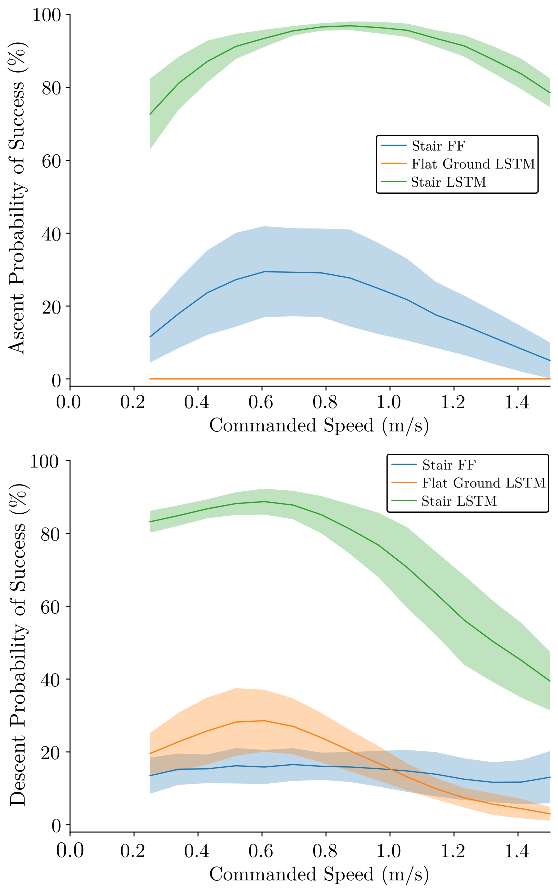


Figure 4.5: We evaluate the probability of successfully climbing and descending stairs without falling as a function of commanded speed between 0.25 m/s and 1.5 m/s over 150 trials. For Stair LSTM policies, there seems to be an optimal approach speed for climbing stairs and a separate optimal descent speed. Stair FF policies do not attain high performance, implying that memory could be an important component of the learned behavior. Flat Ground LSTM policies, having never encountered stairs in training, are virtually unable to climb stairs while finding some success in descending stairs without falling over.

and descent. The Stair FF policies, despite encountering stairs during training, are unable to learn an effective strategy for handling stairs, implying that memory may be an important mechanism for robustness to stair-like terrain.

4.4.1.2 Energy Efficiency Comparison

To understand the consequences of training with terrain randomization, we also compare the cost of transport between Flat Ground LSTM policies, Stair LSTM policies, and Proximity LSTM policies. The cost of transport (CoT) is a common measure of efficiency of legged robots, humans and animals. It is the energy used per distance, normalized by weight to be unitless. It is defined as

$$\text{CoT} = \frac{E_\text{m}}{Mgd}, \tag{4.5}$$

where E_m is the energy used by the motors, M is the total mass of the robot, g is the gravitational acceleration and d is the distance traveled. The energy used by Cassie is calculated using positive actuator work and resistive losses via

$$E_\text{m} = \int_0^T \Big(\sum_i max(\tau_i \cdot \omega_i,\ 0) + \frac{\omega_i^\text{max}}{P_i^\text{max}} \tau_i^2 \Big) dt. \tag{4.6}$$

Here τ_i is the torque applied to motor i and ω_i is its rotational velocity. We use two parameters to define the resistive losses in terms of torque, P_i^{max} is the maximum input power and ω_i^{max} is the maximum speed of motor i. The results of testing steady state CoT at 1 m/s on flat ground can be seen in Table 4.4. These calculations of CoT do not include the overhead power draw from computation and control electronics so they should not be used to compare between robots, only between control policies.

Policy Group	Mean CoT	Std. CoT
Proximity LSTM (stairs)	0.47	0.0086
Stair LSTM	0.46	0.0323
Proximity LSTM (flat)	0.39	0.0257
Flat Ground LSTM	0.38	0.0205

Table 4.4: Locomotion efficiency as measured by cost of transport (CoT) for walking at 1 m/s over flat ground in simulation between three groups of policies over all five random seeds. We note that policies not trained on stair terrain randomization tend to learn more energy efficient gaits, though some energy efficiency can be recovered by providing the stair-trained policies with a binary stair presence/absence input.

We find that Flat Ground LSTM policies learn the most energy efficient gaits for walking on flat ground. Stair LSTM policies learn less efficient flat-ground gaits in order to be robust to stairs; however, the stair proximity input to the Proximity LSTM can help to recover some of this lost energy efficiency by allowing the learned controller to switch between a stair-ready gait and a more energy efficient, flat-ground gait.

4.4.2 Behavior Analysis

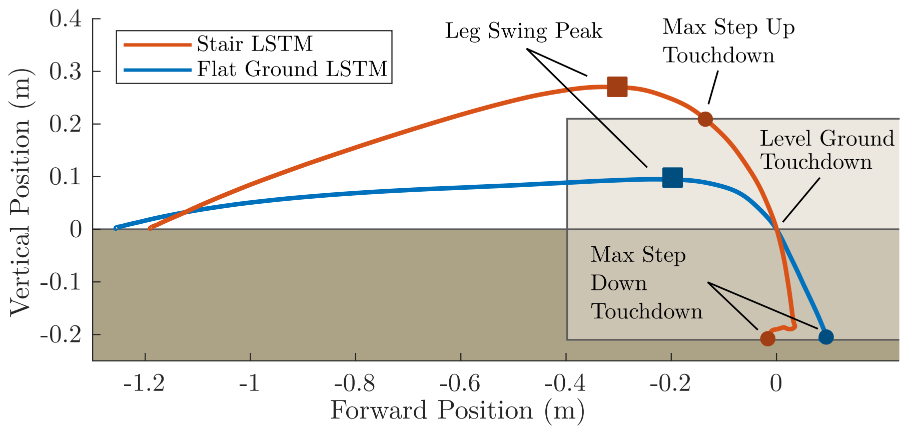
To understand the strategy adopted by the policy, we can benefit from taking the perspective of experimental biology.We specifically look at the behavior as the robot contacts the first step up or down after walking along flat ground. First we will analyze the swing leg motion to understand how the robot places its foot on step ups and step downs. Once the swing foot contacts a step up or down, the force applied by the foot on the ground during stance phase can be modulated to better prepare the robot for future steps. We analyze how the ground reaction force and total impulse varies in the case of step ups and step downs.

4.4.2.1 Swing Foot Motion

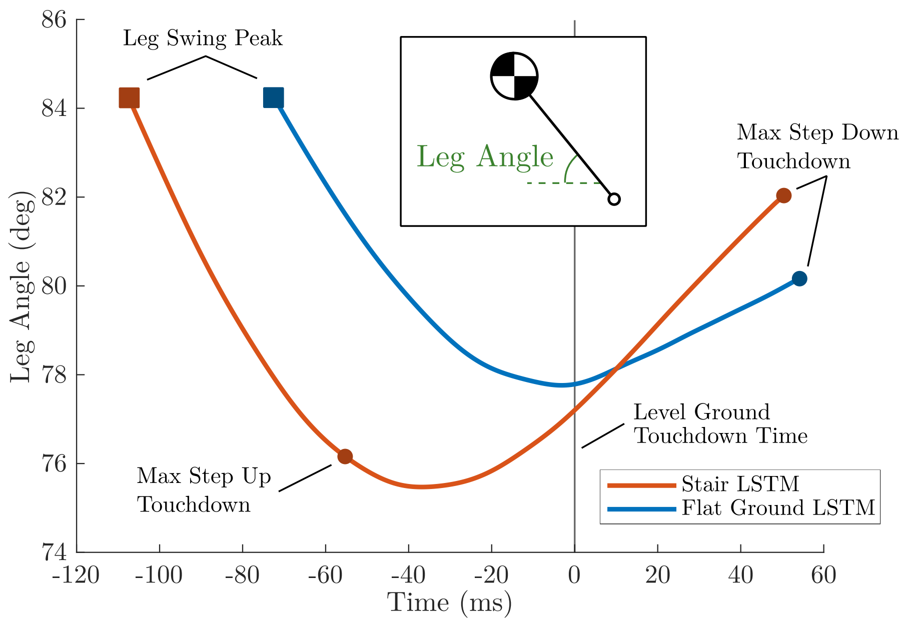
To understand the change the stair terrain makes in the foot swing path we compare the result of a Flat Ground LSTM policy and a Stair LSTM policy when they encounter a drop step. The foot swing path during a drop step lets us see where the policy would place the foot if it had encountered a step up or a step down. Fig. **??** shows the foot swing path of these two policies relative to the ground. We can see that the Stair LSTM policy takes a much higher step compared to the Flat Ground LSTM policy which gives it additional clearance so it can step up onto a large step. A second interesting observation is the steeper path of the swing foot for the Stair LSTM policy. The swing foot only moves forward 14 cm while it is in the height range where it may encounter the front face of a step up. We hypothesize this is a strategy that prevents the foot from stubbing the toe too hard on the front face of a stair and causing the robot to trip forward.

A second viewpoint to understand leg swing motion is to look at the leg swing retraction. In humans and in bipedal birds it is observed that the swing leg is swung backwards, relative to the body, towards the ground near the end of stance [39, 13]. This has the benefits of reducing the velocity of the foot relative to the ground and thus reducing the impact [10] as well as improving ground height disturbance rejection by automatically varying the leg touchdown angle [44].

Our training procedure does not explicitly incentivize the policy to exhibit these leg



Swing foot paths for the stair trained policy and the flat ground policy overlaid on example step ups and step downs.



The leg angle between the robot body and the swing foot as the foot descends toward touchdown.

Figure 4.6: A comparison of the swing foot motion of the Stair LSTM policy and the Flat Ground LSTM policy while locomoting at 1.0 m/s. There is a significant change in the leg swing policy as a result of training on randomized stairs. The most significant changes are higher foot clearance, a steeper foot descent and a faster leg angle retraction rate.

swing retraction behaviors, but we do see them emerge as shown in Fig. **??**. This figure shows the angle of the swing foot relative to the body between the peak of leg swing and contact with the maximum step down. The Stair LSTM policy has a faster leg retraction rate compared to the Flat Ground LSTM policy. With only this data we cannot say if this retraction profile is optimal or even if it is the cause of the improved performance on stairs. However, the fact that there is a significant change in the leg retraction profile as a result of training on stairs is an interesting observation.

4.4.2.2 Ground Reaction Forces

Once the robot's foot has touched down its control authority is limited due to the under-actuated nature of bipedal locomotion. However, the robot still has a significant amount of control through the ground reaction force. To understand how the Stair LSTM policy reacts to a 10 cm step up or down we plot the horizontal and vertical ground reaction forces in the sagittal plane in Fig. 4.7. At the beginning of stance there is a large spike in force that dwarfs the normal forces during stance. The force value during this spike is largely defined by the tuning of the simulation contact model so it is not of primary interest to understanding the behavior of the policy. The first interesting thing we see in subplot A is that the maximum nominal leg force is held relatively constant which is a predicted result of a well adjusted leg swing policy [8]. Second we see that the magnitude of the second hump in the double humped ground reaction forces is increased in the step down and decreased in the step up. In the horizontal forces (subplot B) we see an oscillating signal where the oscillations match the frequency of policy evaluation. We

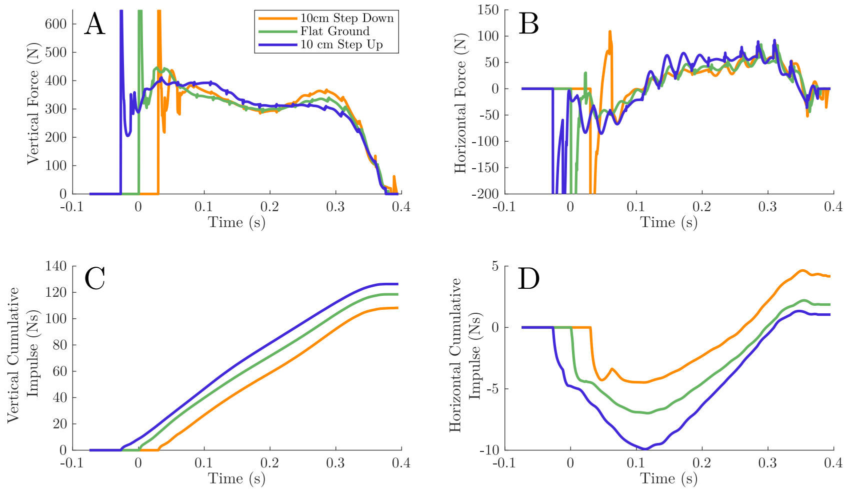


Figure 4.7: The ground reaction forces and cumulative impulses of a Stair LSTM policy when it encounters varying ground height. The peak vertical force (A) after the impact remain roughly equivalent while the force in the second half of stance is modulated. The horizontal force (B) shows oscillations that match the frequency of the learned policy execution rate. This may be the policy controlling the body's attitude. The total vertical impulse (C) shows the expected result of a larger impulse stepping up and a smaller one stepping down. The horizontal impulse (D) shows a result that is predicted by leg swing retraction. When stepping down the foot is shifted backwards relative to the body which results in net acceleration forward which is shown here by a positive horizontal impulse.

hypothesize that this is the policy working to control the attitude of the pelvis. Prioritizing body attitude over forward velocity would be similar to the explicit priorities during single stance in Virtual Model Control [25]. The lower two subplots (C and D) show the cumulative impulse in the vertical and forward directions throughout the stance phase. We can see that the step up applies a larger vertical impulse and the step down a smaller vertical impulse. This agrees with the intuition that the robot should apply a smaller vertical impulse to lower itself down a step compared to lifting itself up a step. The horizontal impulse tells us if the robot speeds up or slows down in the forward direction during the stance phase. We see that the step down results in a significantly larger forward impulse and the step up reduces the vertical impulse very slightly. This aligns with the predicted behavior from a well tuned swing leg retraction policy.

4.4.3 Hardware

The recurrent policies transferred to hardware without any notable difficulties. We were able to take the robot for a walk around a large university campus using a randomly selected Stair LSTM policy and attempt to climb the staircases we came across. We observed robust and error-correcting behavior, as well as successful and repeatable stair ascents and descents. In addition, we noted robustness to uneven terrain, logs, and curbs, none of which were modeled in training. The policy was similarly robust to inclines and deformable terrain, demonstrated by a walk through a wet grass field and up a small hill. These experiments can be seen in our submission video [3], and a still image of one such

[3][Web link to submission video: youtu.be/MPhEmC6b6XU]

experiment can be seen in Fig. 4.4.

In addition to testing one-off terrains all over the university campus, we ran ten trials ascending stairs, and ten trials descending stairs on an outdoor real-world staircase. We recorded an 80% success rate in ascending stairs using the selected Stair LSTM policy, and a 100% success rate in descending stairs. A full video of this trial can be seen in our attachment to this submission. We note that the learned behavior is robust to missteps, and can quickly recover from mistakes, though the policy is not completely infallible and will fall if it makes a particularly egregious error. This experiment can be seen in our supplemental video [4]. The blind, proprioceptive learned strategy appears to rely on a solid stair face; evaluating policies on slatted stairs in simulation resulted in a much higher failure rate, pointing to the limits of such an approach. Even when explicitly included in training, slatted stairs tended to trip up policies on ascent. By contrast, stairs with randomly inclined steps (e.g., ones where each step had a unique pitch and roll orientation) did not seem to be difficult for ascent or descent. Likewise, approaching and ascending stairs at an angle did not seem to be an issue for policies.

4.5 Conclusion

In this work, we have motivated the desirability of a highly robust but blind walking controller, and demonstrated that such a blind bipedal walking controller is capable of climbing a wide variety of real-world stairs. In addition, we note that producing such a controller requires very little modification to the training pipeline in Chapter 3, and in

[4][Web link to supplemental video: youtu.be/nuhHiKEtaZQ]

particular no stair-specific reward terms; simply adding stairs to the environment with no further information is sufficient for learning stair-capable control policies. An important requirement of this learned ability appears to be a memory mechanism of some kind, probably due to the partially observable nature of the task of walking through unknown terrain while blind. In future work, it will be interesting to investigate how vision can be most effectively used to improve the efficiency and/or performance of a blind bipedal robot. Further, this work has demonstrated surprising capabilities for blind locomotion and leaves open the question of where the limits lie.

Bibliography

[1] Farzad Adbolhosseini, Hung Yu Ling, Zhaoming Xie, Xue Bin Peng, and Michiel van de Panne. On Learning Symmetric Locomotion. In *Proc. ACM SIGGRAPH Motion, Interaction, and Games (MIG 2019)*, 2019.

[2] Agility Robotics. cassie-mujoco-sim, 2018.

[3] Shailen Agrawal, Shuo Shen, and Michiel van de Panne. Diverse motion variations for physics-based character animation. *Symposium on Computer Animation*, 2013.

[4] Amos Albert, Michael Suppa, and Wilfried Gerth. Detection of stair dimensions for the path planning of a bipedal robot. In *2001 IEEE/ASME International Conference on Advanced Intelligent Mechatronics. Proceedings (Cat. No. 01TH8556)*, volume 2, pages 1291–1296. IEEE, 2001.

[5] 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.

[6] Taylor Apgar, Patrick Clary, Kevin Green, Alan Fern, and Jonathan W. Hurst. Fast online trajectory optimization for the bipedal robot cassie. In *Robotics: Science and Systems*, 2018.

[7] Kevin Bergamin, Simon Clavet, Daniel Holden, and James Richard Forbes. Drecon: data-driven responsive control of physics-based characters. *ACM Transactions on Graphics (TOG)*, 38(6):1–11, 2019.

[8] Aleksandra V. Birn-Jeffery, Christian M. Hubicki, Yvonne Blum, Daniel Renjewski, Jonathan W. Hurst, and Monica A. Daley. Don't break a leg: running birds from quail to ostrich prioritise leg safety and economy on uneven terrain. *Journal of Experimental Biology*, 217(21):3786–3796, 2014.

[9] Gerardo Bledt, Matthew J Powell, Benjamin Katz, Jared Di Carlo, Patrick M Wensing, and Sangbae Kim. MIT Cheetah 3: Design and control of a robust,

dynamic quadruped robot. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2245–2252. IEEE, 2018.

[10] Yvonne Blum, Susanne W Lipfert, Juergen Rummel, and André Seyfarth. Swing leg control in human running. *Bioinspiration & biomimetics*, 5(2):026006, 2010.

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

[12] Stéphane Caron, Abderrahmane Kheddar, and Olivier Tempier. Stair climbing stabilization of the HRP-4 humanoid robot using whole-body admittance control. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 277–283. IEEE, 2019.

[13] Monica A. Daley and Andrew A. Biewener. Running over rough terrain reveals limb control for intrinsic stability. *Proceedings of the National Academy of Sciences*, 103(42):15681–15686, 2006.

[14] Giorgio Figliolini and Marco Ceccarelli. Climbing stairs with EP-WAR2 biped robot. In *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No. 01CH37164)*, volume 4, pages 4116–4121. IEEE, 2001.

[15] Michele Focchi, Romeo Orsolino, Marco Camurri, Victor Barasuol, Carlos Mastalli, Darwin G. Caldwell, and Claudio Semini. *Heuristic Planning for Rough Terrain Locomotion in Presence of External Disturbances and Variable Perception Quality*, volume 132, pages 165–209. Springer International Publishing, Cham, 2020.

[16] Zhenyu Gan, Yevgeniy Yesilevskiy, Petr Zaytsev, and C David Remy. All common bipedal gaits emerge from a single passive model. *Journal of The Royal Society Interface*, 15(146):20180455, 2018.

[17] Thomas Geijtenbeek, Nicolas Pronost, Arjan Egges, and Mark H Overmars. Interactive character animation using simulated physics.

[18] J-S Gutmann, Masaki Fukuchi, and Masahiro Fujita. Stair climbing for humanoid robots using stereo vision. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, volume 2, pages 1407–1413. IEEE, 2004.

[19] Tuomas Haarnoja, Sehoon Ha, Aurick Zhou, Jie Tan, George Tucker, and Sergey Levine. Learning to walk via deep reinforcement learning. *arXiv preprint arXiv:1812.11103*, 2018.

[20] Matthew Hausknecht and Peter Stone. Deep Recurrent Q-Learning for Partially Observable MDPs. In *AAAI Fall Symposium on Sequential Decision Making for Intelligent Agents (AAAI-SDMIA15)*, November 2015.

[21] Nicolas Heess, Jonathan J. Hunt, Timothy P. Lillicrap, and David Silver. Memory-based control with recurrent neural networks. *CoRR*, abs/1512.04455, 2015.

[22] Ayonga Hereid, Christian M. Hubicki, Eric A. Cousineau, Jonathan W. Hurst, and Aaron D. Ames. Hybrid zero dynamics based multiple shooting optimization with applications to robotic walking. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5734–5740, 2015.

[23] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[24] Daniel Holden, Taku Komura, and Jun Saito. Phase-functioned neural networks for character control. *ACM Transactions on Graphics*, 36(4):1–13, 2017.

[25] Jerry Pratt and Chee-Meng Chew and Ann Torres and Peter Dilworth and Gill Pratt. Virtual model control: An intuitive approach for bipedal locomotion. *The International Journal of Robotics Research*, 20(2):129–143, 2001.

[26] 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):eaau5872, 2019.

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

[28] Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning quadrupedal locomotion over challenging terrain. *Science Robotics*, 5(47):eabc5986, Oct 2020.

[29] Sergey Levine, Jack M Wang, Alexis Haraux, Zoran Popovic, and Vladlen Koltun. Continuous character control with low-dimensional embeddings. *ACM Transactions on Graphics (TOG)*, 31(4):1–10, 2012.

[30] Philipp Michel, Joel Chestnutt, Satoshi Kagami, Koichi Nishiwaki, James Kuffner, and Takeo Kanade. GPU-accelerated real-time 3D tracking for humanoid locomotion and stair climbing. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 463–469. IEEE, 2007.

[31] EZ Moore and M Buehler. Stable stair climbing in a simple hexapod robot. Technical report, McGill Research Centre for Intelligent Machines, 2001.

[32] 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. *arXiv preprint*, 2019.

[33] Hae-Won Park, Alireza Ramezani, and Jessy W Grizzle. A finite-state machine for accommodating unexpected large ground-height variations in bipedal robot walking. *IEEE Transactions on Robotics*, 29(2):331–345, 2012.

[34] 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 3803–3810, 2018.

[35] 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 3803–3810, May 2018.

[36] Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel van de Panne. DeepMimic. *ACM Transactions on Graphics*, 37(4):1–14, 2018.

[37] Xue Bin Peng, Glen Berseth, and Michiel Van de Panne. Terrain-adaptive locomotion skills using deep reinforcement learning. *ACM Transactions on Graphics (TOG)*, 35(4):1–12, 2016.

[38] Xue Bin Peng and Michiel van de Panne. Learning Locomotion Skills Using DeepRL: Does the Choice of Action Space Matter? *CoRR*, abs/1611.01055, 2016.

[39] KL Poggensee, MA Sharbafi, and A Seyfarth. Characterizing swing-leg retraction in human locomotion. In *Mobile Service Robotics*, pages 377–384. World Scientific, 2014.

[40] Siavash Rezazadeh, Christian Hubicki, Mikhail Jones, Andrew Peekema, Johnathan Van Why, Andy Abate, and Jonathan Hurst. Spring-Mass Walking With ATRIAS in 3D: Robust Gait Control Spanning Zero to 4.3 KPH on a Heavily Underactuated Bipedal Robot. Dynamic Systems and Control Conference, 10 2015. V001T04A003.

[41] Agility Robotics. Cassie: Dynamic Planning on Stairs.

[42] Alla Safonova, Jessica K Hodgins, and Nancy S Pollard. Synthesizing physically realistic human motion in low-dimensional, behavior-specific spaces. *ACM Transactions on Graphics (ToG)*, 23(3):514–521, 2004.

[43] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms, 2017.

[44] André Seyfarth, Hartmut Geyer, and Hugh Herr. Swing-leg retraction: a simple control model for stable running. *Journal of Experimental Biology*, 206(15):2547–2555, 2003.

[45] Jonah Siekmann, Srikar Valluri, Jeremy Dao, Lorenzo Bermillo, Helei Duan, Alan Fern, and Jonathan Hurst. Learning memory-based control for human-scale bipedal locomotion. In *Proceedings of Robotics: Science and Systems*, July 2020.

[46] Jonah Siekmann, Srikar Valluri, Jeremy Dao, Lorenzo Bermillo, Helei Duan, Alan Fern, and Jonathan Hurst. Learning Memory-Based Control for Human-Scale Bipedal Locomotion. 2020.

[47] A. Singla, S. Padakandla, and S. Bhatnagar. Memory-Based Deep Reinforcement Learning for Obstacle Avoidance in UAV With Limited Environment Knowledge. *IEEE Transactions on Intelligent Transportation Systems*, pages 1–12, 2019.

[48] Sebastian Starke, He Zhang, Taku Komura, and Jun Saito. Neural state machine for character-scene interactions. 2019.

[49] Sebastian Starke, Yiwei Zhao, Taku Komura, and Kazi Zaman. Local motion phases for learning multi-contact character movements. *ACM Transactions on Graphics*, 39(4), 2020.

[50] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[51] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[52] Jie Tan, Tingnan Zhang, Erwin Coumans, Atil Iscen, Yunfei Bai, Danijar Hafner, Steven Bohez, and Vincent Vanhoucke. Sim-to-Real: Learning Agile Locomotion For Quadruped Robots. In *Proceedings of Robotics: Science and Systems*, Pittsburgh, Pennsylvania, June 2018.

[53] Jie Tan, Tingnan Zhang, Erwin Coumans, Atil Iscen, Yunfei Bai, Danijar Hafner, Steven Bohez, and Vincent Vanhoucke. Sim-to-real: Learning agile locomotion for quadruped robots. In *Proceedings of Robotics: Science and Systems*, Pittsburgh, Pennsylvania, June 2018.

[54] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, Timothy P. Lillicrap, and Martin A. Riedmiller. Deepmind control suite. *CoRR*, abs/1801.00690, 2018.

[55] 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.

[56] Miomir Vukobratovic and Branislav Borovac. Zero-Moment Point - Thirty Five Years of its Life. *I. J. Humanoid Robotics*, 1:157–173, 03 2004.

[57] Miomir Vukobratović and Branislav Borovac. Zero-moment point—thirty five years of its life. *International journal of humanoid robotics*, 1(01):157–173, 2004.

[58] Eric R Westervelt, Jessy W Grizzle, and Daniel E Koditschek. Hybrid zero dynamics of planar biped walkers. *IEEE transactions on automatic control*, 48(1):42–56, 2003.

[59] Pierre-Brice Wieber and Christine Chevallereau. Online adaptation of reference trajectories for the control of walking systems. *Robotics and Autonomous Systems*, 54(7):559–566, 2006.

[60] Zhaoming Xie, Glen Berseth, Patrick Clary, Jonathan Hurst, and Michiel van de Panne. Feedback control for cassie with deep reinforcement learning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1241–1246. IEEE, 2018.

[61] Zhaoming Xie, Patrick Clary, Jeremy Dao, Pedro Morais, Jonanthan Hurst, and Michiel van de Panne. Learning locomotion skills for cassie: Iterative design and sim-to-real. volume 100 of *Proceedings of Machine Learning Research*, pages 317–329. PMLR, 2020.

[62] Wenhao Yu, Jie Tan, C. Karen Liu, and Greg Turk. Preparing for the Unknown: Learning a Universal Policy with Online System Identification. In *Proceedings of Robotics: Science and Systems*, Cambridge, Massachusetts, July 2017.

[63] He Zhang, Sebastian Starke, Taku Komura, and Jun Saito. Mode-adaptive neural networks for quadruped motion control. *ACM Transactions on Graphics*, 37(4):1–11, 2018.