Learning to Walk: Legged Hexapod Locomotion from Simulation to the Real World

by Maxime Kawawa-Beaudan

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

Professor Avideh Zakhor
Research Advisor

5/19/2022

(Date)

* * * * * * *

Professor Sergey Levine
Second Reader

(Date)

LEARNING TO WALK: LEGGED HEXAPOD LOCOMOTION FROM SIMULATION TO THE REAL WORLD

by

Maxime Kawawa-Beaudan

A thesis submitted in partial satisfaction of the requirements for the degree of

Master of Science in Electrical Engineering and Computer Science

in the Graduate Division of the University of California, Berkeley

Spring 2022

Abstract

This thesis presents methods for training intelligent robotic systems to navigate challenging, diverse environments. In particular it focuses on legged locomotion for hexapods rather than well-studied bipedal or quadrupedal robots. We employ hierarchical reinforcement learning to integrate proprioception with a low-level gait controller, and train on-policy algorithms entirely in simulation before transferring to a real robot. We develop command-conditioned policies in PyBullet that learn to walk at commanded target velocities, and switch to Isaac Gym to train policies with multi-objective rewards both in rough and flat terrains. A comparison is established between gaits with motion priors and prior-free gaits. Both methods successfully surmount joist obstacles typically seen in attics in the real world. We propose novel approaches to the sim-to-real problem designed to address limitations of affordable hardware. We demonstrate methods in the Isaac Gym simulation for integrating vision.

Contents

List of Tables

List of Figures

Acknowledgments

None of this work would be possible without the support of my parents, who throughout my life cleared innumerable roadblocks from my path – those I know about, and those they dealt with far before I ever came upon them, so that I never even knew they existed.

Thank you to Berkeley for bringing the best out of me, demanding a work ethic I never otherwise would have learned, and most of all, for putting me on the right street corners at the right times to meet the people who have become my best friends.

Thank you to these friends for giving me my humor, my smile, and my favorite memories. I'm grateful that you never let me use my work as an excuse to stay in; I owe many of my best stories to the nights I said, "Yes," despite my hesitations. Please read this thesis, however, as proof that I do in fact do research.

Thank you to Professor Avideh Zakhor for dispensing invaluable guidance over the past two years, for giving me autonomy in my work, and for granting me ownership over my projects. Special thanks to Ashish Kumar and Tingnan Zhang for guiding me through the weeds of robotics research.

Chapter 1

Background

1.1 What's So Great About Legs?

Legged robots are powerful for the same reasons that many animals have evolved systems of legged locomotion. Legged systems, whether biological or robotic, are versatile and agile. They can navigate rough terrain, surmount imposing barriers, and traverse obstacle courses far more easily than can their tracked and wheeled robotic counterparts.

The unconscious nature of biological motor control gives the illusion that legged locomotion is a simple task. In fact, the dynamics of legged movement are complex. Legged systems in motion are unstable. A body's joints have limitations with respect to achievable angles, safely applicable torques, and maximum velocities. The joints connect masses each with their own inertial properties and stability conditions. These constraints must be respected to avoid collapse. Even in the simplest environments, walking involves evaluating potential footholds for load-bearing fitness, balancing body mass against gravity and ground contact forces, and planning sequences of joint movements to reach safe foot configurations.

Many methods in traditional robotics decompose legged movement explicitly into these or equivalent substeps. For example, [11] uses an optical terrain scanner to map the robot's surroundings, segments this map into a grid, estimates the fitness of each grid square, and identifies potential footholds. A complex series of hand-coded rules govern this process, as well as the process of moving each foot to its target position.

1.2 Learning From Action

As the field of artificial intelligence matured, neural-network-based systems proved their worth for a broad class of traditional problems. Robotics is no exception. As universal function approximators, neural networks can be trained to replace many previously hand-coded robotic systems – for instance, they can learn the dynamics of a robot's actuators without any engineered guidance [7].

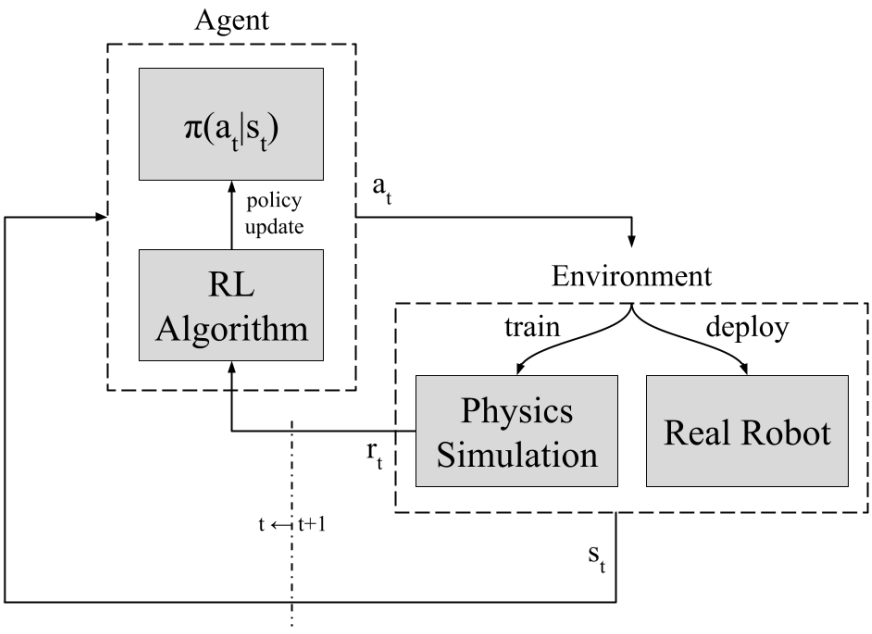


Figure 1.1: Block diagram of the reinforcement learning paradigm. a_t, s_t, r_t are, respectively, the actions, states, and rewards at timestep t.

There is a catch: training neural networks takes generally on the order of hundreds of thousands or millions of steps. At each step the parameters of the neural network are adjusted slightly to improve the network's performance. For problems in robotics, this may mean having a robot attempt to walk thousands of times – a training regimen which could take days, weeks, or months to complete and would likely damage the hardware beyond repair.

A faster alternative is to build an accurate simulation in which physically realistic models of robots learn policies for motion which are then transferred to robots in the real world [13]. These simulations, training multiple robots in parallel at faster than real time, allow simulated robots to acquire hours or days worth of simulated experience in minutes of wall-clock time.

There is another catch: many neural-network-based tasks have clearly defined target outputs for a given input. For example, a neural network is given a collection of images and asked to determine for each image whether it contains a cat. The target output, or ground truth, is "Yes" if the image contains a cat and "No" otherwise.

Many robotics applications, however, involve sequential decision-making tasks. At any given timestep, there is no ground truth – no pre-determined right answer, no known sequence of motor positions that result in the best outcome. Take the example task of a robot opening a door. There are many ways for a robot to turn a door knob, none of them any more "correct" than another at any single slice of time, and success – whether the door is open or

closed – is determined only after many consecutive movements.

A reasonable sequence of actions is also not universally appropriate. A robot cannot learn one series of actions to turn all door knobs from all intial positions, like a video game character acting out a pre-programmed animation. When a robot takes an action, the environment changes in response. The door may move when touched; the knob may slip out of the robot's grasp when turned too quickly. This is key to the difficulty of sequential decision making. A cat classifier neural network does not have to contend with the fact that saying "Yes" to one image may change the contents of the next image.

For these reasons and more, problems in robotics are increasingly approached as tasks in reinforcement learning, a subfield of artificial intelligence studying methods for learning from action. As seen in Figure 1.1, in reinforcement learning, decision-making algorithms called agents take actions a_t at each timestep t. They use a decision-making rule called a policy, denoted by π, to decide which actions a_t to take given observations of their environments represented by the state s_t at time t. These actions a_t in turn affect their environments, and from the environment they receive a reward r_t which indicates the fitness of their chosen action. At this point one timestep or "transition" is complete and $t \leftarrow t + 1$.

The agents' incentive is not to match some ground truth at each timestep but to maximize a cumulative reward over sequences of timesteps.

Formally, the environment makes up a Markov Decision Process, consisting of a state space $\mathcal{S}$, action space $\mathcal{A}$, transition operator $\mathcal{T}$, and reward function r where:

$$\mathcal{T}_{i,j,k} = p(s_{t+1} = i | s_t = j, a_t = k) \; \forall s_{t+1} \in \mathcal{S}, s_t \in \mathcal{S}, a_t \in \mathcal{A} \tag{1.1}$$

$$r : \mathcal{S} \times \mathcal{A} \to \mathbb{R} \tag{1.2}$$

The objective is to find a policy $\pi_\theta(a_t|s_t)$, parameterized by θ, which outputs $p(a_t|s_t)$ and maximizes the cumulative reward:

$$J(\theta) = \mathbb{E}_\pi \left[\sum_t r_t \right] \tag{1.3}$$

1.3 Motivation

The E-ROBOT Prize, put on by the U.S. Department of Energy, aims to source innovative solutions from academia to dangerous or arduous building industry tasks. These problems broadly include sensing, inspection, mapping, and retrofitting tasks. U.C. Berkeley's Video and Image Processing Lab's hexapod robot for exploring tight and inaccessible spaces placed as a finalist in the first phase of the competition. The goal of this project is to design a robot which can autonomously navigate such confined spaces. By partially automating the inspection process and eliminating arduous parts of the job such as the one depicted in Figure 1.2, the project proposes a way of making retrofitting safer for workers. The overarching goal of automating inspection and retrofitting is to make buildings more energy efficient.

Figure 1.2: Current solution to our motivating use case: technician inspecting tight crawl space with only the feet visible.

In this report, we use reinforcement learning to learn a policy for legged robotic locomotion in tight environments such as attics with joists.

1.4 Related Work

The reinforcement learning approaches relevant to this work can be broken down into two categories: those that learn free gaits and those that learn periodic gaits. Periodic gait controllers, as in [8, 11, 19, 3, 14], take advantage of the fact that gaits of legged systems are most often highly coordinated and periodic in nature.

The system in [8], depicted in Figure 2.1, is a good exemplar of a periodic controller. It consists of a trajectory generator which implements a parameterized family of functions. These functions are chosen to mimic the basic periodic motions of the joints as appropriate for the task. For example, for the task of walking, a trajectory generator implementing sine curves could be used, as leg joints follow roughly sinusoidal trajectories during walking. The parameters of this trajectory generator would be the amplitude and frequency of the sine curve. At each timestep a neural network policy learns to predict the parameters of the trajectory generator, as well as small residuals which are added to the output of the trajectory generator to produce the final joint angles.

Free gait controllers such as [10] learn locomotion policies from scratch, without any pre-defined trajectory underlying their motions. These controllers can be harder to learn as

they have no knowledge of gaits supplied implicitly through the trajectory generator, but can also be more robust and versatile, as they are not beholden to this underlying pattern of motion.

In [8] a quadruped robot is trained to match desired velocities, and its reward is a function of the distance between the real and target velocity. This is sufficient for proving the method on an example task, but for learning efficient gaits, most methods incorporate physics-derived or bioenergetic rewards [4, 10]. These rewards can include joint velocities, joint torques, smoothness of motion, and more.

To navigate non-flat terrains, it becomes important to have some form of perception. Much of the recent work in this area uses a hierarchical reinforcement learning approach [2, 9, 16, 20]. A high-level policy is trained to map from an observation space of images to potential paths, abstract latent features, or desired foot placements. A low-level policy is trained to map the outputs of the high-level policy to joint torques or angles, depending on the capabilities of the robot's hardware. The joint policy is typically trained in simulation and deployed later on hardware, with attention paid to minimizing the sim-to-real gap. For the low-level system design, several papers [9, 20] use periodic gaits as in [8].

1.5 Contributions

The outline of this report is as follows: In Chapter 2 we present our work developing velocity command-conditioned policies in PyBullet. In Chapter 3, we present our work in Isaac Gym, developing policies able to optimize multi-objective rewards to learn viable policies in both rough and flat environments. We demonstrate in simulation our model's capability to be integrated with vision, and present a method for bridging the sim-to-real gap on cheap hardware. In particular we demonstrate the effectiveness of a two-stage approach in which a policy is first trained with access to privileged simulated observations to optimize multi-objective rewards, and then fine-tuned with access only to a more limited set of observations available on hardware.

In Chapter 4 we present our results with illustrative rollouts of our policies in both simulation and the real world, and provide observations of each policy's performance. We show that periodic motions arise organically from prior-free policies both in simulation and the real world. We demonstrate that policies trained on rough terrain can successfully surmount joist obstacles in the real world. In Chapter 5 we discuss potential avenues for continued work.

Chapter 2

Command-Conditioned Approach

This chapter presents our work in PyBullet, developing velocity command-conditioned policies which successfully transfer to real hardware and respond to target velocity commands.

2.1 Learning Gaits with Motion Priors

Broadly this report explores methods for learning robust legged locomotion policies for hexapods. To this end, we train both periodic and free gaits. For our periodic gait we adopt the system in [8], with a trajectory generator providing a predefined prior on the motion of each joint. Figure 2.1 shows the general structure of a motion-prior gait policy. A policy π is trained to take in observations s_t, which may be sensor readings or proprioceptive

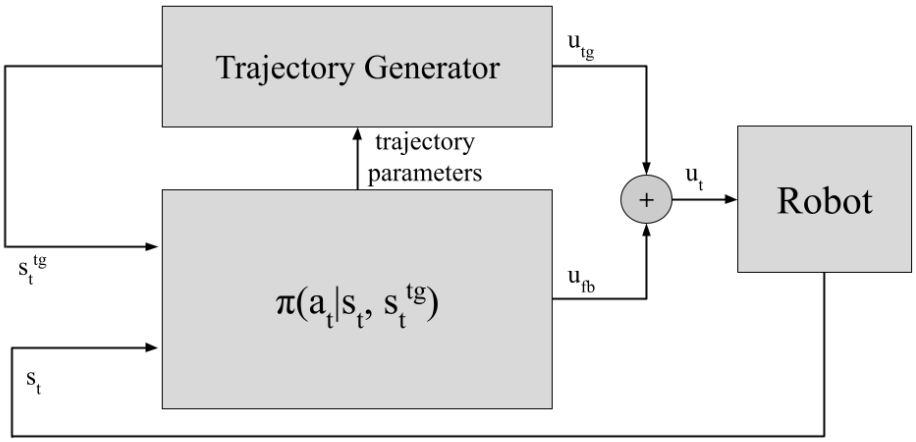


Figure 2.1: General structure of a motion-prior gait policy, where s_t^{tg} is the trajectory generator state at time t, s_t is the environmental state, a_t are the policy predictions, u_{fb} are the residuals, u_{tg} are the trajectory generator actions, and u_t are the combined actions.

readings, and observations from the trajectory generator s_t^{tg}, and to predict actions a_t at each time step t. These actions are split into parameters for the trajectory generator as well as residuals u_{fb} added directly to the trajectory generator output u_{tg} to create actions u_t which are actuated in the environment.

We train in PyBullet, a physics simulation engine chosen for its proven capacity to rapidly simulate the complex dynamics of legged motion for learning problems. Our aim is to use PyBullet to learn policies which can take as input a limited set of proprioceptive observations and output sequences of actions to move the robot forward at linear velocities matching the commanded velocity.

2.2 Simplified Control Problem

As in [8] we first deploy our motion prior approach with a simplified control problem. This allows us to verify the implementation of the approach and its validity in a synthetic setting. In the synthetic control problem, a policy attempts to predict the position of a point in $\mathcal{R}^2$ which moves in the x and y dimensions with time t. The trajectory generator from Figure 2.1 takes the form of a figure-eight curve in two dimensions shown in Figure 2.2.

$$u_{tg}(t) = \begin{bmatrix} x_{tg}(t) \\ y_{tg}(t) \end{bmatrix} \tag{2.1}$$

$$x_{tg}(t) = \alpha_x \sin(2\pi t) \tag{2.2}$$

$$y_{tg}(t) = \frac{\alpha_y}{2} \sin(2\pi t) \cos(2\pi t) \tag{2.3}$$

where $x_{tg}(t)$ and $y_{tg}(t)$ are the actions along the x-axis and y-axis respectively as prescribed by the trajectory generator at time t. α_x, α_y are the amplitudes of x_{tg}, y_{tg} respectively and are the parameters of the trajectory generator. In this synthetic problem these actions represent nothing except for the position of a point in $\mathcal{R}^2$.

The trajectory generator tracks and increments t at each step as its internal state. The policy π predicts the two parameters of this trajectory generator, α_x, α_y, shown in Equations 2.2 and 2.3, as well as small residuals $x_{fb}(t), y_{fb}(t)$

$$u_{fb}(t) = \begin{bmatrix} x_{fb}(t) \\ y_{fb}(t) \end{bmatrix} \tag{2.4}$$

depicted in Figure 2.1, at each time step. As shown in Figure 2.1 the policy receives as observations the trajectory generator's state $s_{tg}(t)$, as well as

$$s_t = \begin{bmatrix} x_{gt}(t) \\ y_{gt}(t) \end{bmatrix} \tag{2.5}$$

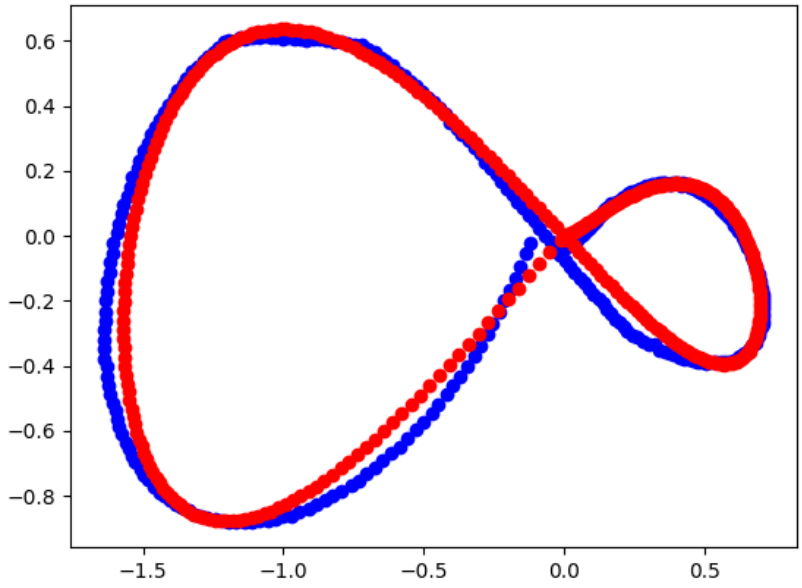


Figure 2.2: Demonstration of the synthetic control problem. The horizontal axis is the x action axis and the vertical axis is the y action axis. The red curve corresponds to the ground truth $x_{gt}(t), y_{gt}(t)$ and the blue curve corresponds to the trained policy predictions $x(t), y(t)$. In a perfectly trained system $\forall t\ (x_{tg}(t), y_{tg}(t)) = (x(t), y(t))$.

where $(x_{gt}(t), y_{gt}(t))$ are values from a figure-eight curve generated randomly before training. The goal in this synthetic problem is to learn to predict the parameters of this random figure-eight curve, which we take to be our ground truth.

The final predicted action, represented as u_t in Figure 2.1, is:

$$x(t) = x_{tg}(t) + x_{fb}(t) \tag{2.6}$$
$$y(t) = y_{tg}(t) + y_{fb}(t) \tag{2.7}$$

The reward is the Euclidean distance between the ground truth $(x_{gt}(t), y_{gt}(t))$ and this output $(x(t), y(t))$. We achieve final rewards comparable with those shown in [8] and our rewards converge stably through training. As depicted in Figure 2.2, the policy learns to closely approximate the ground truth function at each time step.

2.3 Robotic Control Problem

Hexapod robots have six legs and three joints per leg, as depicted in Figure 2.3. For our command-conditioned robotic control problem, we use a trajectory generator of the following form:

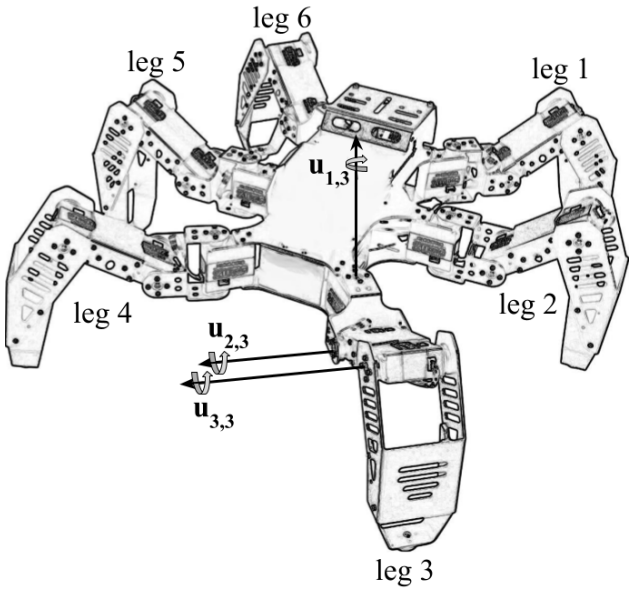


Figure 2.3: Action-joint correspondences. Each LX-224 servo actuator can actuate to ± 120 degrees from its depicted position.

$$u_{j,i} = \alpha_j \cos(\phi_{leg_i}) \tag{2.8}$$
$$\tag{2.9}$$

where $u_{j,i}$ is the position of the j-th joint in the hexapod's i-th leg, ϕ_{leg_i} is the i-th leg's phase, and α_j is the j-th joint trajectory's amplitude. With this formulation we define one trajectory $u_{j,*}$ for each of the joints $j \in [1, 3]$ in a leg, shared among all six legs i. The joint correspondences are illustrated in 2.3. At each timestep the policy predicts the parameters $\{f_{tg}, \alpha_1, \alpha_2, \alpha_3\}$ where f_{tg} is a frequency parameter allowing the policy to modulate the period of each joint's trajectory, and α_j is the amplitude of the j-th joint's trajectory. The policy also predicts an 18-dimensional residual u_{fb} added directly to the trajectory generator output. We clip these residuals to the range $[-0.1, 0.1]$, f_{tg} to $[0, 1.25]$, and a_j to $[-2.094, 2.094]$ since 2.094 radians is the maximum achievable angle for our motors. The trajectory generator maintains an internal clock ϕ_t, defined below in radians:

$$\phi_t = \phi_{t-1} + 2\pi f_{tg} \Delta T \tag{2.10}$$

where ΔT is the timestep of the policy, in our case 30Hz, and f_{tg} is predicted by the policy at each time step, allowing the policy to increase or decrease the rate at which the clock advances to achieve target velocities. Each leg maintains its own phase:

State Element	Constituents	Units
Base attitude	Euler roll, pitch	Radians
3-axis angular velocity	$\omega_x, \omega_y, \omega_z$	Radians / s
3-axis linear acceleration	a_x, a_y, a_z	m/s^2
Trajectory generator phase	$\sin(\phi_t), \cos(\phi_t) \in \mathbb{R}^2$	Unitless
Command velocity	v_t^T	m/s

Table 2.1: Constituent elements of the state s_t in our PyBullet setup.

$$\phi_{leg_i} = \phi_t + \Delta\phi_{leg_i} \tag{2.11}$$

where $\Delta\phi_{leg_i}$ is fixed and defined by the desired gait. For a tripod gait, three legs share $\Delta\phi_{leg_i} = 0$ and three legs share $\Delta\phi_{leg_i} = \pi/2$. The final joint position predicted by the model is then:

$$u_{j,i} = \alpha_j \cos(\phi_{leg_i}) + u_{fb_{j,i}} \tag{2.12}$$

The state s_t observed at each time step by our policy consists of the elements enumerated in Table 2.1, and the learned policy maps from s_t to actions a_t.

The policy is trained with a single reward term: $r_t = v_m e^{\frac{(v_t^T - v_x)^2}{v_m}}$ where v_m is the maximum command velocity, and v_x is the actual linear velocity of the base in the x direction. This reward incentivizes the policy to match the command velocity as closely as possible.

We train for 5×10^4 rollouts with 25 seconds per rollout and the simulation running at 30Hz. This gives us around 4×10^7 steps per training run. We successfully learn a simple policy network, with 2 hidden layers each of size 10, which responds to the command velocity. We transfer our low-level policy running at 30Hz to a real hexapod robot, the Hiwonder SpiderPi. The onboard IMU, an MPU-6050, gives 3-axis angular velocity gyroscope measurements and 3-axis linear acceleration measurements from the accelerometer.

The videos of our real-world results are depicted in Figure 2.4, demonstrating the policy's capacity to respond to velocity commands. In Figure 2.4(a) the target velocity is 0.4 for the entire length of the rollout and in Figure 2.4(b) the target velocity is zero for the first half of the rollout and 0.4 for the second half. The robot demonstrates slight, slow movement for the first half of this rollout, then speeds up as expected – there is some motion at a zero-motion command due to variance in the policy prediction and imperfect command response with our simple reward function.

(a) (b)

Figure 2.4: Video demonstrations of the policy trained with the method described in this chapter deployed on the HiWonder Spiderpi hexapod. (a) A video demo with target velocity of 0.4 is available here. (b) Another video demo with target velocities of 0 and 0.4 is available here.

Chapter 3

Command-Free Approach

In this chapter we present our work in NVIDIA's Isaac Gym training policies on rough and flat terrain with a multi-objective reward function that incentivizes forward movement in a straight line while respecting the limitations of the real actuators and incurring minimal damage to the hardware. We describe our experimental setup for policies with and without priors and present our solution to the limitations of our affordable hardware.

3.1 Simulation Setup

In order to scale up the complexity of our system, both in terms of our reward functions and the diversity of our rough environment, we find it necessary to switch from PyBullet to NVIDIA's Isaac Gym. This allows us to simulate many robot environments in parallel, letting us train much faster and with larger batch sizes for stability in learning with multi-objective rewards. We adopt the same basic setup as in [15]. At each timestep, we feed the state s_t to an observation encoder which outputs a vector z_t, which is then passed in to the policy, as illustrated in Figure 3.3. In our case the observation encoder is a 1-layer LSTM with hidden size 64. The recurrent nature of the encoder allows it to remember past observations and incorporate salient information into z_t. The state consists of the elements enumerated in Table 3.1 concatenated into a single vector.

We train our agents in two environments: the first is a flat plane; the second is a rough terrain consisting of randomly generated joist courses arranged to naturally give rise to a learning curriculum as in [15]. An example of such terrain is deplicted in Figure 3.1. In the rough environment but not the flat environment we terminate episodes if the robot's yaw exceeds $\frac{\pi}{3}$ to incentivize agents to scale obstacles rather than turning away from them. We train 4,000 agents at once with a policy frequency of $\Delta T = 25$Hz. As in [15], for robustness to sim-to-real differences we add uniform random noise to all of our state elements. This random noise is scaled as indicated in Table 3.1. We also randomize the friction coefficient of each robot in the range $[0.5, 1.25]$, and apply pushes in random directions to each robot's base mass every 8 seconds to improve robustness and stability of the learned gait.

State Element	Constituents	Units	Noise Range
Base attitude	Euler roll, pitch, yaw	Radians	[-0.1, 0.1]
Joint positions	$q_t \in \mathbb{R}^{18}$	Radians	[-0.01, 0.01]
Previous actions	$a_t \in \mathbb{R}^{18}$	Radians	[0, 0]
Trajectory generator phase*	$\sin(\phi_t), \cos(\phi_t) \in \mathbb{R}^2$	Unitless	[0, 0]
Terrain height measurements**	$h[x][y] \in \mathbb{R}^{81}$	Meters	[-0.1, 0.1]

Table 3.1: Constituent elements of the state s_t in our Isaac Gym setup. Noise range represents the magnitude of uniform random noise added to the observations to improve the robustness of the policy. *: only when using a gait with prior; **: only when using a policy with perception, with x, y forming a square grid with resolution 0.1m extending 0.4m in each direction from the robot.

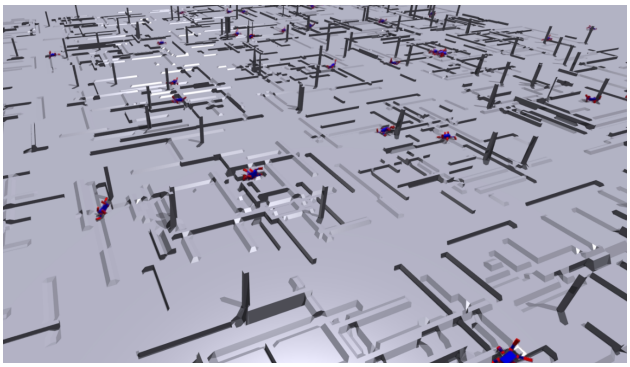


Figure 3.1: Simulated rough terrain of joists.

We penalize contacts on the base link and the two links of each leg closest to the base, the coxa and the femur. This encourages the agent to keep the base raised, avoiding damage to the hardware in the real world, and also discourages it from learning degenerate policies lying motionless on the terrain to avoid incurring negative rewards.

Our reward term is the weighted sum of the elements in Table 3.2. We use this reward function for all experimental setups – rough terrain, flat terrain, with prior, without prior, and with perception. This reward was shaped over many iterations with a great deal of experimentation, beginning with only the linear velocity in body x term to incentivize forward motion. The linear velocity in body y and angular velocity yaw terms were added to disincentivize sideways motion and turning to keep the agent on a straight line. Ground impact was added after initial real-world rollouts revealed the agent's tendency to let its feet collide harshly with the ground, damaging the hardware. The action rate, action magnitude, torque, and joint acceleration penalties were all added as regularizers to constrain the actions to behave in ways that were deployable on real hardware, instead of learning on-off control schemes that would burn out the motors or command impossible joint positions.

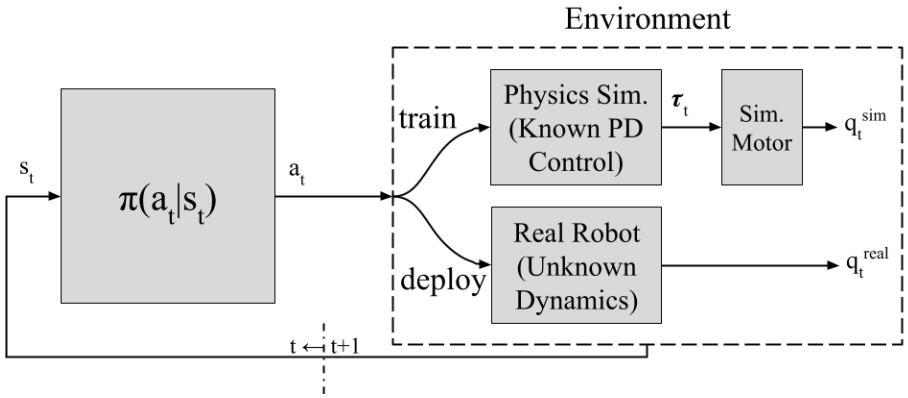


Figure 3.2: Block diagram for prior-free gaits where τ_t are the simulated torques, q_t^{sim} are the joint positions in simulation and q_t^{real} are the joint positions on the real robot.

3.2 Learning Prior-Free Gaits

A motion prior implicitly encodes information about efficient patterns of movement, making the learning problem easier. But it also constrains the resultant motion to be a transformation of the underlying motion prior. For traversing diverse obstacles, free gaits are preferable. Consider how legged animals move – when moving at constant speed they adopt periodic gaits. But when confronted with obstacles they may move their legs entirely independently with no discernable pattern.

Some existing work [10, 4] has shown that through reinforcement learning, efficient free

Reward Term	Expression	Weight
Linear velocity in body x	$clip(v_x, min = -0.4, max = 0.4)$	1e3
Linear velocity in body y	v_y^2	-1e2
Angular velocity: yaw	ω_z^2	-1e0
Ground impact	$\|f_t - f_{t-1}\|^2$	-1e-1
Collision penalty	$\mathbb{1}$\{coxa, femur, or base contacting terrain\}	-1e0
Action rate	$\|a_t - a_{t-1}\|^2$	$5 \cdot$ -1e-1
Action magnitude	$\|a_t\|^2$	-1e-1
Torques	$\|\tau\|^2$	1e-2 $\cdot$ -1e-1
Joint acceleration	$\hat{\ddot{q}}^2 = \frac{\dot{q}_t - \dot{q}_{t-1}}{\Delta t}^2$	-1e-5
Joint limit penalty	$clip(q_{min} - q_t, max = 0) + clip(q_t - q_{max}, min = 0)$	-1e0

Table 3.2: Constituent elements of the multi-objective reward r_t in our Isaac Gym experimental setup.

gaits can be learned without priors. To learn prior-free gaits, we do away with the trajectory generator from Figure 2.1. The policy directly predicts the joint positions q_t as depicted in Figure 3.2. In simulation, actions a_t are used as position targets for a known and tuned proportional-derivative (PD) controller which outputs torques τ_t. These torques are then applied to the simulated actuators, which model the physical constraints such as stiffness and friction to produce q_t^{sim}, the actual joint positions attained at time t in simulation.

3.3 Learning Perception for Motion

In the rough environment we also train a policy with perception, which adds to the state 81 terrain height observations sampled from a $0.8m \times 0.8m$ centered grid around the robot with spacing of $0.1m$.

Deploying the policy with perception on the real robot is outside of the scope of this thesis, but remains an avenue for future work.

3.4 Bridging the Sim-to-Real Gap

Joint positions on our robot are written to and read from all 18 joints through a shared serial port. This prohibits parallel access, so all joints are read and written to in serial, with wait times in between to clear the bus. One loop reading the position of all the joints can take multiple seconds, making it impossible to use them as inputs to a policy running at 25Hz. The joint positions q_t must therefore also be omitted from the states in simulation, since we have no access to joint position feedback on the real robot. Most other methods [16, 10, 4, 9] include q_t in their states, and indeed we find that including q_t is critical for performance across almost all settings. As in [16] we find that the yaw readings from our IMU in hardware are noisy and drift quickly.

We adopt an idea similar to that in [10] to adapt a policy learned with one set of observations to allow deployment with a different set of observations. While [10] uses this method to enable rough estimation of environmental conditions we use it to bridge the gap between performance-critical observations available only in simulation and the more limited observation set available in the real world. Our approach is depicted in Figure 3.3: in the first stage we train the system end-to-end using q_t in our state s_t. In the second stage we treat the trained observation encoder and policy as experts and fine tune a student observation encoder $\hat{E}$ and student policy $\hat{\pi}$ using as input only the more limited observation set $\hat{s}_t$ available on real hardware. We fine tune starting with the trained policy weights, and supervise the fine tuned encoder's $\hat{z}_t$ with the original encoder's z_t. We do the same for actions. In other words, we train the fine tuned encoder and policy to recover the same z_t and a_t from the more limited observation set by optimizing the following loss:

$$\mathcal{L}_{phase\_ii} = \mathcal{L}_{encoding} + \mathcal{L}_{behavioral} = MSE(z_t, \hat{z}_t) + MSE(a_t, \hat{a}_t) \tag{3.1}$$

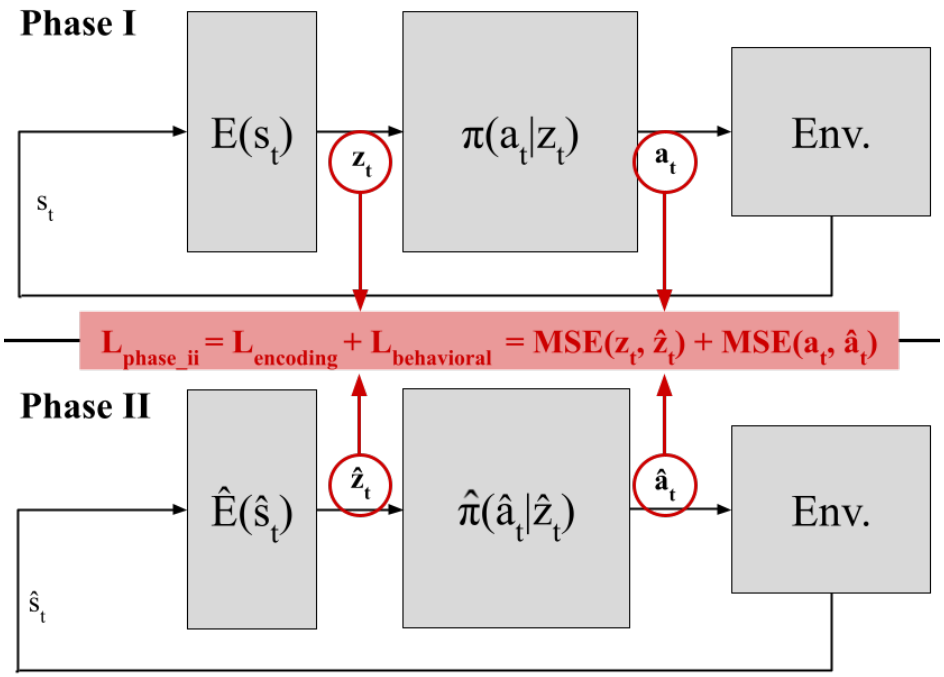


Figure 3.3: Block diagram for the two-stage training scheme. E represents an LSTM observation encoder. π represents the policy. Env. is the simulated environment. The mean-squared-error (MSE) terms are summed to form the loss minimized by the Phase II neural networks.

For all methods we use action scaling, which is to say we let

$$a_t^{unscaled} \leftarrow \pi(a_t|s_t)$$
$$a_t^{clipped} \leftarrow \text{clip}(a_t^{unscaled}, \text{min} = -s * a_{max}, \text{max} = s * a_{max})$$
$$a_t^{scaled} \leftarrow \frac{a_t^{clipped}}{s}$$

where s is the action scale and a_{max} is the maximum allowable joint angle, around 2.094 radians, for the method without prior, and the maximum allowable trajectory parameter value for the method with prior. For the method without prior we adopt the action scale 4. For the method with prior, we adopt the action scale 64 – this allows the unscaled actions to occupy a wider range, which makes regressing the second phase more stable.

Chapter 4

Experiments and Results

In this chapter we present our findings in both simulation and in the real world for the approach outlined in Chapter 3. We demonstrate the tradeoff our policy makes between various terms in the multi-objective reward function in Chapter 3, and the behaviors in simulation and the real world that result from these tradeoffs.

To disambiguate our different approaches, we provide in Table 4.1 a list of policies trained and tested in this chapter with corresponding shorthand notations.

Description	**Shorthand Notation**
Blind, prior-free policy trained on flat terrain	PF-FT-B
Blind, prior-free policy trained on rough terrain	PF-RT-B
Blind, with-prior policy trained on flat terrain	WP-FT-B
Blind, with-prior policy trained on rough terrain	WP-RT-B
Perceiving, prior-free policy trained on rough terrain	PF-RT-P

Table 4.1: Abbreviations of training setups. Policies with FT are trained on flat terrain while policies with RT are trained on rough terrain. Policies with PF are prior-free and those with WP are with-prior. Policies with B are blind, policies with P are perceiving.

4.1 Training Setup

We train our policy from Chapter 3 in NVIDIA's Isaac Gym [13], which allows us to train thousands of robots in parallel, amortizing the overhead of the physics simulation over many environments. We write our own URDF representation of the robot, included in Appendix A, after measuring link dimensions and link weights on our platform, the Hiwonder SpiderPi hexapod. We use a simulation timestep of 0.005 seconds and step the simulation 8 times per policy step – in other words, the policy runs at $1/(0.005s \times 8) = 25$Hz and in simulation we

apply the actions a_t for 8 steps of 0.005 seconds each. Each simulation step further uses 4 physics engine substeps of duration $0.005/4 = 0.00125$ seconds.

For our core reinforcement learning algorithm we use the open-source implementation of Proximal Policy Optimization from [15]. Our agent is an actor-critic where both the actor and critic are Multi-Layer Perceptrons with hidden layers of size 128, 64, and 32. We use Exponential Linear Unit activations between the layers. The learning rate is adaptive as in [15] and we train for 8,000 iterations, where each iteration consists of 24 timesteps for all agents. 4,000 robots simulated in parallel results in 96,000 transitions per iteration which we divide into 4 minibatches of size 24,000. We train exclusively on a single GPU, an NVIDIA GeForce RTX 3080 with 10.5GB of memory.

4.2 Problem: Sim-to-Real Action Gap

We deploy our policies on a low-cost $600 hexapod, the Hiwonder SpiderPi, which uses inexpensive $19 LX-224 servo motors from the same manufacturer. [15, 10, 4] and other existing methods deploy policies on robots with price tags of tens of thousands of dollars. The actuators of such robots typically allow for reading and writing of PD controller gains and have real-time joint position, joint velocity, and even torque feedback.

Our motors provide a maximum of 1.96 N·m of torque and their velocities cannot exceed 5.24 rad/s. The onboard microcontroller sets the position of the joints by directly modulating the motor voltage using pulse width modulation. As shown in Figure 3.2, there is no proportional-derivative controller or higher level of abstraction above this – proportional-derivative control is performed at the circuit level inside each motor. The gains are unknown and cannot be changed. The DC motor drives a gear train providing high but unknown stiffness. In short, we have no model of the torques the real motors will exert in response to a given action, and therefore no way of simulating them reliably.

[7] proposes a method for using a neural network to model such unknown dynamics, but requires torque feedback from the motor allowing one to build a dataset of joint positions and resultant torques $\mathcal{D} = \{\mathcal{T}_t, q_t\}_{t=1}^{T}$. Our motors have no velocity or torque feedback. [16] proposes a method based on an ideal DC motor model for approximating the motor dynamics, but requires knowledge of the torque constant and other parameters not provided for our motors. These factors make effort control, in which actions are mapped to torques in simulation using an accurate actuator model, difficult.

Furthermore, our robot is light and its links therefore have low inertia. Even if accurate effort control were possible it would need to be simulated with very small timesteps, increasing our compute time multiplicatively.

As a result we use position control, which is less sensitive to physical accuracy, in simulation. Our policy's predicted actions are interpreted as PD position targets. We also find that increasing the number of physics engine substeps per simulation step from 1 to 4 is critical to simulating physics accurately. Without this alteration, for example, the legs collapse

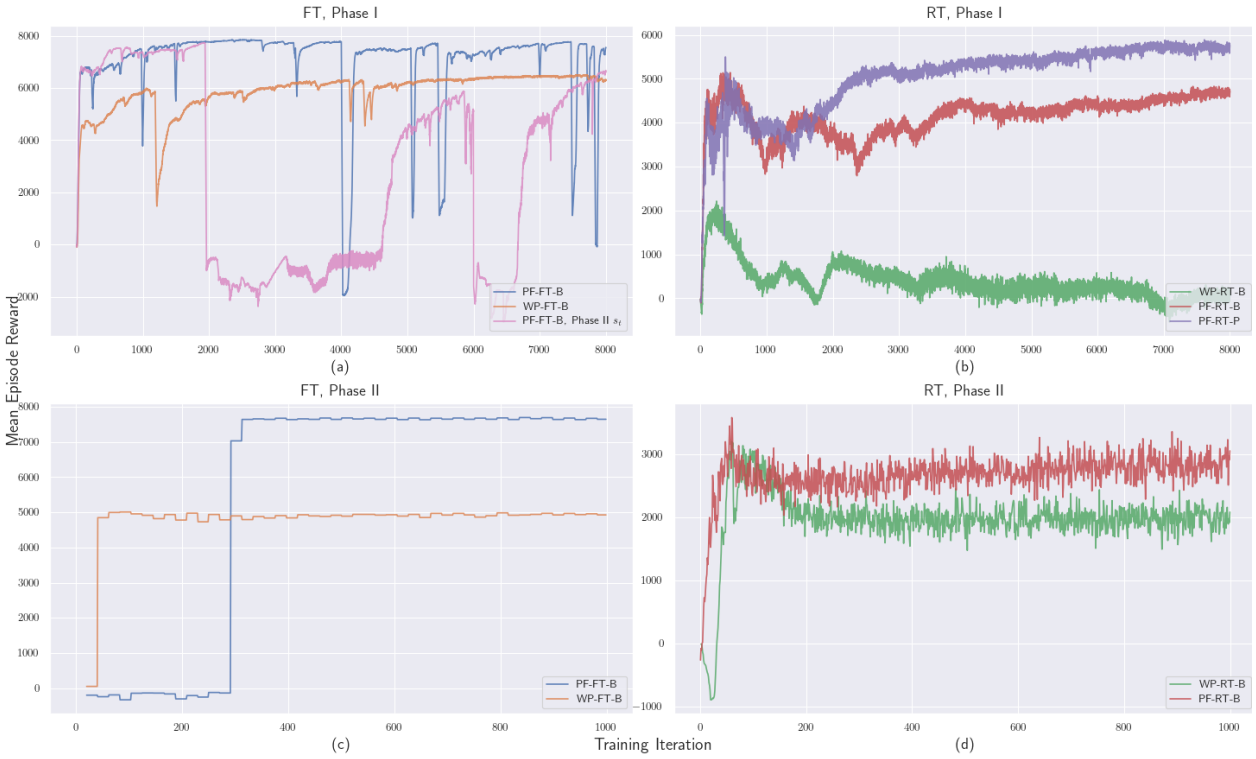


Figure 4.1: Learning curves for the two-phase training scheme. We do not train a second phase for the perceiving policies because we do not transfer them to the real robot in the scope of this work. Total rewards during training for the first phase for (a) FT and (b) RT policies. Total rewards during training for the second phase for (c) FT and (d) RT policies.

under the weight of the base mass, while in real life the robot holds its own weight without a problem.

Videos of all our results can be found on this project page.

4.3 Learning in Simulation

Here we present our cumulative and individual reward terms through training.

Figure 4.1 demonstrates the total rewards yielded by the two-stage approach described in Figure 3.3. Figure 4.1(a) compares the first phase rewards achieved by PF-FT and WP-FT methods; Figure 4.1(c) shows the rewards recovered in the second phase from these policies. As evidenced in the blue and orange curves in Figures 4.1(a), in the FT environment, performance is similarly high and rewards converge at the same rate for PF and WP methods. However, in the RT environment, adding a prior causes first phase final rewards to drop from around 5,000 to around 0, as evidenced in the green curve in Figure 4.1(b). In the RT environment reward recovery is less than 100%, but the majority of the reward can

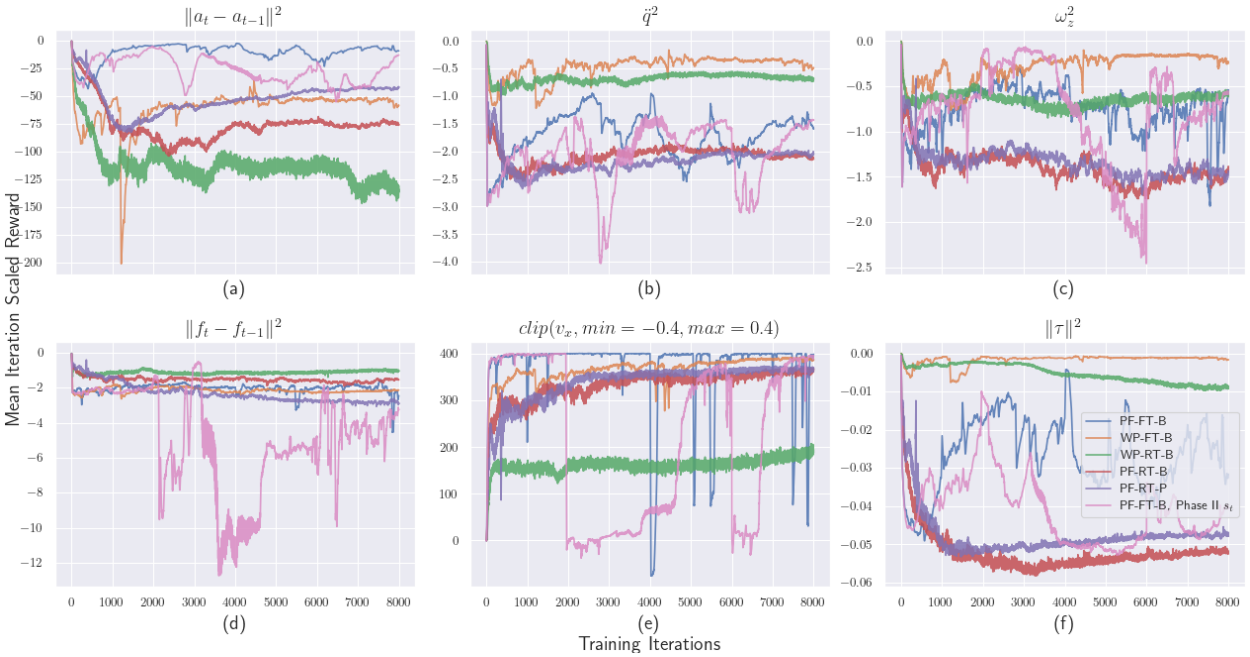


Figure 4.2: Learning curves for selected individual reward terms for each of the methods. (a) shows the action rate penalty, which discourages rapid, non-smooth changes in action between adjacent time steps, (b) shows the joint acceleration penalty, (c) shows the angular velocity in the yaw dimension penalty, (d) shows the ground impact penalty, (e) shows the forward movement reward, and (f) shows the torque penalty.

be recovered. For example, the red curve in Figure 4.1(b) correspoinding to the PF-RT-B policy achieves final rewards of around 5,000 and 3,000 from the first and second phases respectively.

Figure 4.1(a) also presents results in pink from a first phase trained with access only to second phase observations – that is, without q_t and yaw. This is a baseline against which we compare our two-stage method rewards to establish its efficacy. As seen in Figure 4.1(a) in pink, this baseline's rewards are highly unstable, collapsing for long periods of time to large negative values, and failing to converge. This demonstrates that access to q_t is critical to task performance. The final reward at iteration 8,000 for this method is around 6,000.

However, training first with q_t included in the state as shown in blue in Figure 4.1(a) allows us to reach final rewards at iteration 8,000 of around 8,000. Training the second phase using only state elements accessible on the real robot allows us in the FT environment to recover the full performance of using q_t, a final reward of around 8,000, without any of the stability problems as in the first phase trained with these observations.

Figure 4.2 illustrates the sub-reward terms making up the total reward during training. As seen, the dominant contribution to the total reward is the forward movement sub-reward. Figure 4.2(e) demonstrates that the difference between the RT-WP and RT-PF methods –

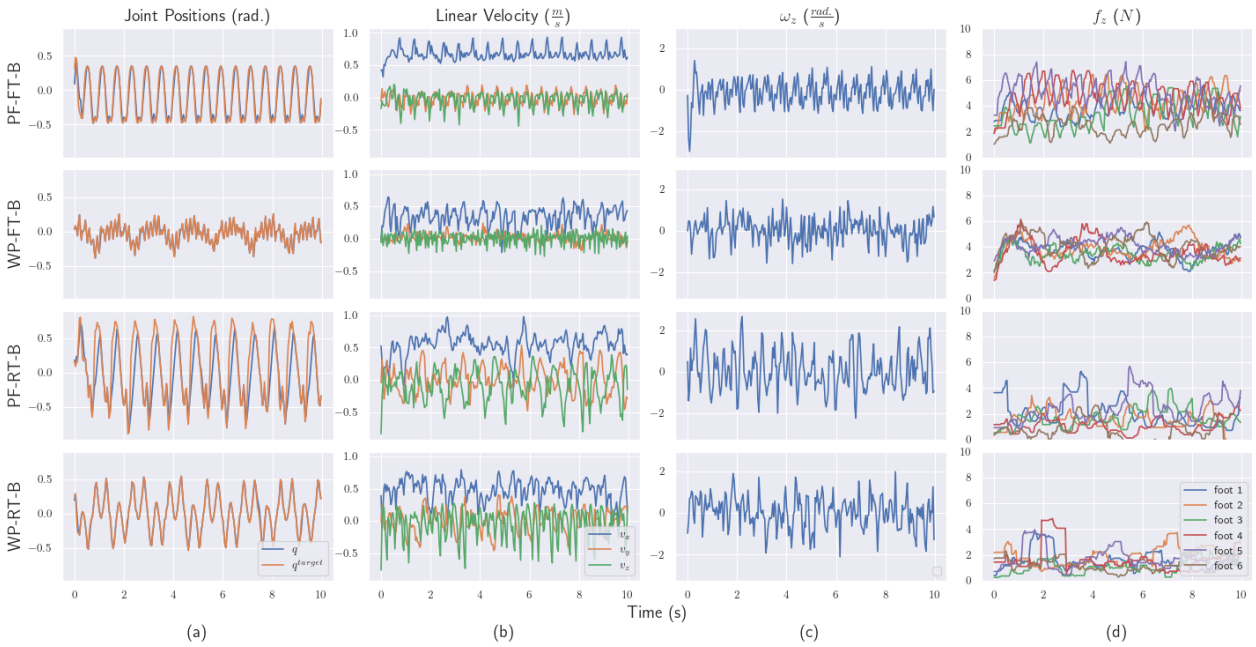


Figure 4.3: Measurements in simulation of rollouts of the trained policies. (a) joint positions q_t versus PD targets q_t^{target}; (b) linear velocity of the base in 3 principal directions; (c) angular velocity of the base about the yaw axis; (d) foot contact forces in the z vertical dimension. Note that joint positions q_t versus PD targets q_t^{target} are reported for the back left leg's first joint only, for ease of viewing. Foot contact forces in z dimension f_z are smoothed with a 1-second long moving average.

shown in green and red, respectively – lies mainly in this forward movement sub-reward. Specifically, the final forward movement sub-reward for the RT-WP method is around 200, while for the RT-PF method it is around 350. This is to be expected since periodic gaits are not ideal for surmounting obstacles.

4.4 Learned Behavior in Simulation

In Figures 4.3 and 4.4 we present the resultant behaviors in simulation of each of our trained policies for flat and rough terrain, with or without motion priors. Figure 4.3(a) shows the target and achieved joint positions q_t and q_t^{target} for one leg joint, Figure 4.3(b) shows the base linear velocities v_x, v_y, v_z along 3 principal axes, Figure 4.3(c) shows the angular velocity ω_z in the yaw dimension of the base, and Figure 4.3(d) shows the contact forces in the z-dimension experienced by the agent's feet. From Figure 4.3(a) we observe that, even without periodic motion priors, our prior-free methods learn that optimal movement is periodic in nature. In fact, comparing the first and second rows of Figure 4.3(a) demonstrates that the FT-PF method learns a smoother periodic behavior than the FT-WP method, though the

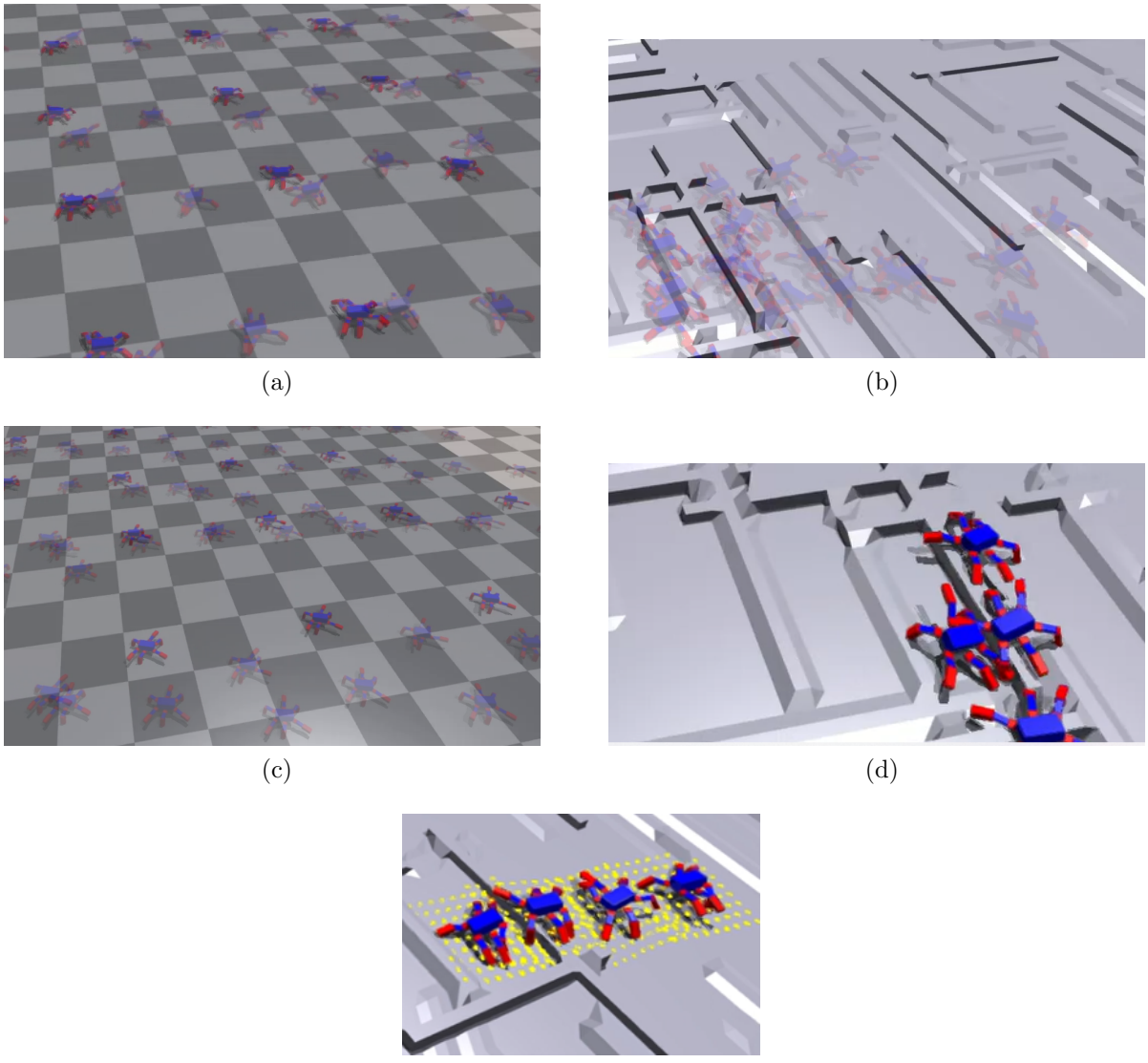
swing amplitude of FT-WP is smaller and easier on hardware than that of FT-PF. Figures 4.3(b) and 4.3(c) also reveal how much more perturbation the base experiences in RT versus FT. Both linear velocity and angular velocity fluctuate more and are less periodic for RT policies than FT policies.

Figure 4.4 includes a variety of simulation videos for the policies under consideration. The video in Figure 4.4(a) illustrates how the PF-FT-B policy learns a gait that resembles a wave gait, using the front, middle, and back pairs of legs on either side of the body in a periodic sequence. The video in Figure 4.4(b) shows that without any perception, the PF-RT-B policy can be thought of as learning a behavior that works on average in the rough terrain, and adapts to the changes in attitude when encountering obstacles. It has higher foot clearance and average base height than PF-FT-B, allowing it to surmount joists. Figure 4.4(c) shows that the WP-FT-B policy learns a rapid foot-tapping behavior that works in simulation, allowing it to shuffle forward at a constant rate with a stable base. As seen in Figure 4.2 this behavior yields lower rewards than PF-FT-B even with similar forward movement rewards because it incurs a high action rate penalty. Figure 4.4(d) demonstrates how the WP-RT-B learns a coordinated, periodic gait that successfully overcomes joist obstacles at a slower rate than the prior-free gait. However, it frequently gets stuck on the leading edge of a joist and requires multiple gait cycles to surmount this leading edge. While for WP-RT-B, trajectories of the legs are tied together, for PF-RT-B, each leg can learn its own behavior. The front two legs for PF-RT-B learn a probing behavior which allows the agent to catch the top face of a joist and surmount it. Figure 4.4(e) shows that while the PF-RT-B learns behavior that works on average, the PF-RT-P policy can adapt to the specific terrain it faces due to its perception capabilities. PF-RT-P stumbles on joists less frequently and notably keeps the base far more stable, with a nearly constant base height even when surmounting obstacles.

4.5 Real World Behavior

We successfully transfer our WP-FT-B, PF-FT-B, PF-RT-B, and WP-RT-B gaits to a real hexapod robot, the Hiwonder SpiderPi. The onboard IMU, an MPU-6050, provides 3-axis angular velocity gyroscope measurements and 3-axis linear acceleration measurements from the accelerometer. We use the MPU-6050's onboard sensor fusion to convert these raw measurements to projected gravity, linear acceleration less gravity, and Euler angles.

We find yaw to be subject to drift, and linear acceleration to be unreliable in simulation due to finite differencing. We use projected gravity measurements to derive Euler angles, which tend to be more interpretable, so we use these in our policy and discard projected gravity. Our measurements after sensor fusion over real rollouts from each of our four policies can be examined in Figure 4.5. They demonstrate the smoothness of our observations, particularly attitude, after sensor fusion. They also reflect in Figures 4.5(a) and 4.5(c) the same periodicity emerging as in simulation across methods with and without priors. Figures 4.5(a) and 4.5(c) illustrate how little perturbation the base experiences under policy WP-FT-B, due to its largely stand-still behavior.



(a) (b) (c) (d) (e)

Figure 4.4: Screenshots from simulation rollout videos. (a) A simulated rollout of the PF-FT-B method can be seen in this video. (b) A simulated rollout of the PF-RT-B method can be seen in this video. (c) A simulated rollout of the WP-FT-B method can be seen in this video. (d) A simulated rollout of the WP-RT-B method can be seen in this video. (e) A simulated rollout of the PF-RT-P method can be seen in this video, with points at which the robot receives terrain height measurements marked in yellow.

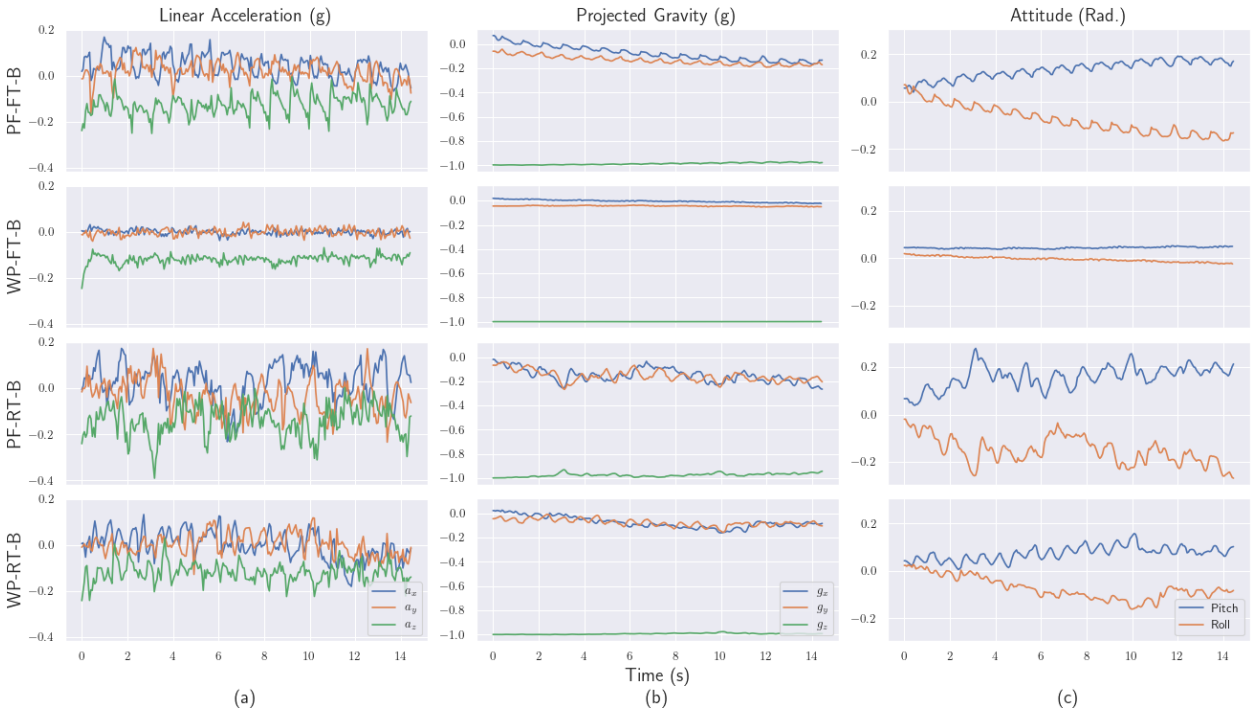
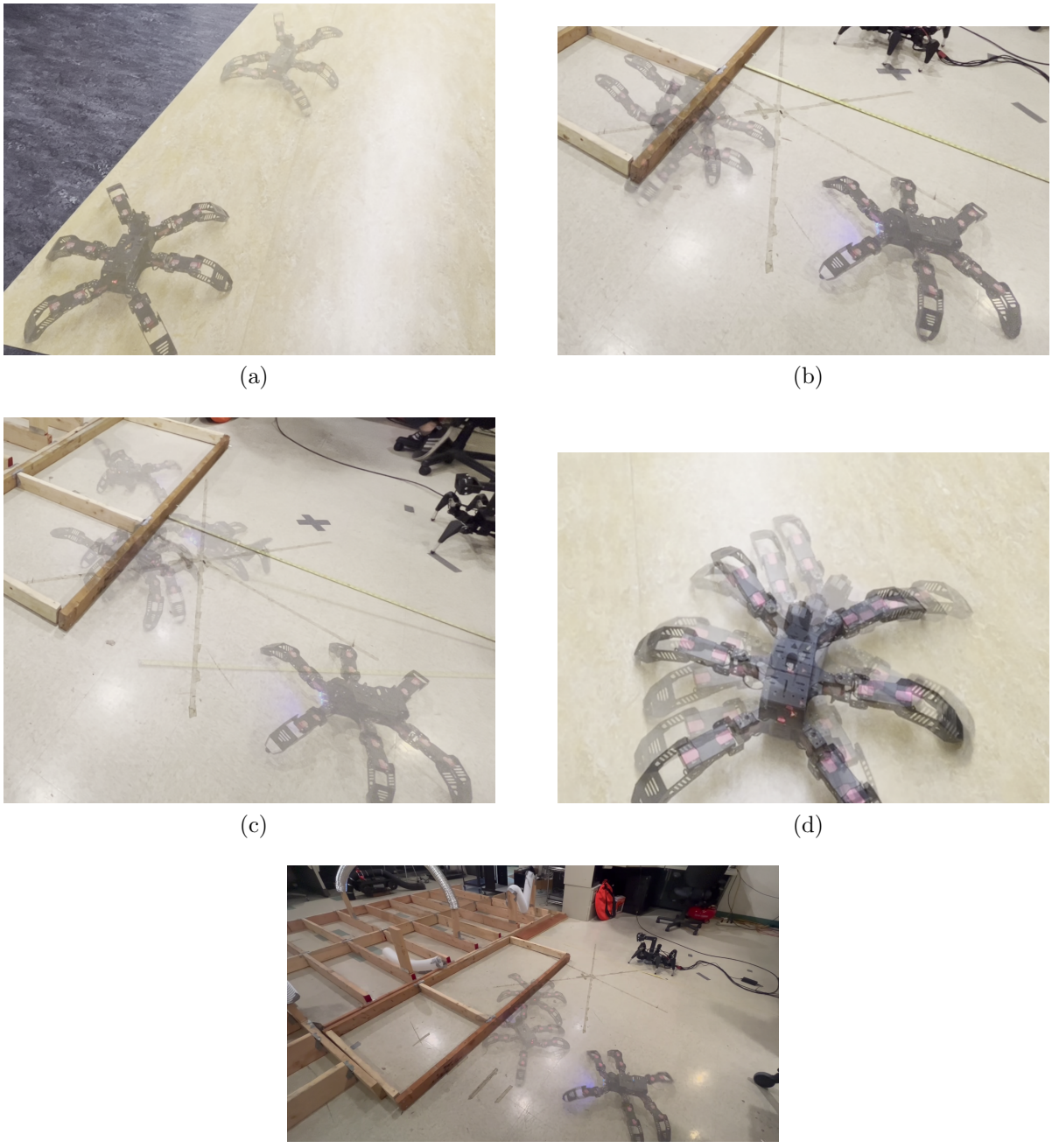


Figure 4.5: Sensor observations from real world rollouts from four different policies. Note that the policy takes as input only the attitude. (a) linear acceleration of the base in the three principal dimensions; (b) projected gravity vector in 3 dimensions; (c) attitude reported from the IMU.

Figure 4.6 shows video demonstrations of our policies deployed in the real world. Figure 4.6(a) reveals that the behavior on real hardware for PF-FT-B closely resembles the behavior in simulation, proving our effective sim-to-real transfer scheme. Due to wear and tear on the hardware the robot drifts to the right over the course of the rollout. Figure 4.6(b) shows that the PF-FT-B policy fails to surmount any obstacles. This is expected because obstacles are out of distribution for this policy, which is not trained to be robust to non-flat terrain. Figure 4.6(c) illustrates that the PF-RT-B policy successfully surmounts 3 joist obstacles in just 15 seconds, demonstrating the high-stepping and high base height behavior that we observe in simulation. This can be mostly although not entirely attributed to wear and tear on our testing hardware, as even in simulation, there is an apparent turning tendency. Figure 4.6(d) validates that the WP-FT-B policy's rapid foot tapping behavior fails to transfer to the real world and heats up the motors significantly. This is likely due to the differences between contact and friction dynamics in simulation and the real world – in simulation, low friction with the flat plane allows the feet to shuffle forward. In real life, the rubber-tipped feet grip the ground more strongly, and this shuffling results in stand-still behavior. Figure 4.6(e) proves that the WP-RT-B policy transfers from simulation to real successfully. Rough terrain incentivizes the robot to learn a gait with high foot clearance, as foot shuffling will not surmount joists. This behavior almost surmounts obstacles in the real world, getting the



(a) (b) (c) (d) (e)

Figure 4.6: Demonstrates of our policies deployed on robots in the real world. (a) Videos of the PF-FT-B policy deployed on flat terrain can be viewed in this video, this video, and this video. (b) Videos of the PF-FT-B policy deployed on rough terrain in this video. (c) A video demonstration of the PF-RT-B policy deployed on rough terrain can be viewed in this video. We observe a right-hand turning behavior not reflected in simulation due to wear on the testing hardware. (d) The WP-FT-B policy deployed on flat terrain is demonstrated in this video. (e) The WP-RT-B policy deployed on rough terrain is demonstrated in this video.

Approach	**Time in Seconds to Surmount First Joint**	**Number of Joints Surmounted**
WP-FT-B	N/A	0
WP-RT-B	12	1
PF-FT-B	N/A	0
PF-RT-B	5	3

Table 4.2: High-level summary of real-world results on the joist obstacles task.

base and all but two legs clear of the obstacle. But as expected from simulation results and the rewards achieved in Figure 4.1 this policy does worse than PF-RT-B.

We place the robot in front of a series of joist obstacles and let it run for 15 seconds, recorrding the number of joists it surmounts and how much time elapses before it surmounts the first joist. These results are summarized in Table 4.2. While the PF-RT-B policy works best, surmounting 3 joists in just 15 seconds, the WP-RT-B policy manages to almost surmount a joist. This is expected given our simulation rollouts and far higher rewards for PF-RT-B than WP-RT-B in Figure 4.1. As expected, both FT policies get stuck on the joist task.

In summary, on flat terrain our best policy is PF-FT-B, which is predictable from observing the simulated rapid foot movement behavior yielded by WP-FT-B and the simulated periodic gait with long strides yielded by PF-FT-B. On rough terrain our best policy is PF-RT-B, which is again predictable from the rewards in Figure 4.1 and from the simulated rollouts which demonstrated the WP-RT-B policy's tendency to get stuck on the leading edges of obstacles.

Chapter 5

Discussion and Conclusions

We have shown that our policies can be deployed on real hexapod robots in both flat and non-flat terrains. We have also demonstrated that even with significant cost constraints and unreliable and limited observations on the real robot, we can deploy a policy trained entirely in simulation without any fine-tuning. We present a two-stage method for training with privileged observations in simulation and fine-tuning for more limited real-world observations.

In general we find that methods with motion priors fare worse than methods without priors, especially in the case of rough terrains. This makes sense intuitively, as obstacle-surmounting gaits are determined ad-hoc and responsively. It is difficult to compare results from our work in PyBullet regarding command-conditioned gaits and our command-free gaits in Isaac. However, we can safely say that obstacles are out of distribution for the gaits trained in PyBullet with motion priors, and that these policies are fit only for flat terrain. PyBullet gaits also had no incentive to avoid damage to the hardware or reduce energy consumption through smooth gaits. Our Isaac PF-FT-B recovered a periodic gait organically while optimizing for all of these constraints. In Isaac we are also able to simulate 4,000 robots simultaneously as compared to a single robot in PyBullet. Therefore our Isaac Gym agents have far more experience than our PyBullet agents. For these reasons in flat terrain our best-performing policy is PF-FT-B, followed by our command-conditioned policy from PyBullet, followed by WP-FT-B. In rough terrain our best-performing policy is the PF-FT-B policy, followed by the WP-FT-B policy.

More broadly we find that reinforcement learning occupies a vast design space. RL requires tuning many hyperparameters, from algorithm parameters such as learning rates and policy architectures to simulation parameters such as timesteps, substeps, and PD gains, to design parameters such as state elements and reward shaping. It also requires optimizing many low-level details on real hardware, and making these choices in a complementary fashion between simulation and real hardware.

Once these many hyperparameters are tuned, however, RL allows one to iterate through policies and develop complexity quickly, changing the task and environment to learn robust policies.

In future work we aim to train command-conditioned policies using Isaac Gym. We also

aim to deploy a policy with perception in the real world. The sim-to-real gap for depth images is large, but the sim-to-real gap for observation encodings z_t of depth images may be small – one possible approach could be to regress z_t of simulated depth images against z_t of simulated height map observations and deploy this policy. Another could be to implement methods constructing elevation maps from depth images on the real robot.

Bibliography

[1] Marko Bjelonic, Navinda Kottege, Timon Homberger, Paulo Borges, Philipp Beckerle, and Margarita Chli. "Weaver: Hexapod Robot for Autonomous Navigation on Unstructured Terrain". In: *Journal of Field Robotics* 35 (June 2018), pp. 1063–1079. DOI: 10.1002/rob.21795.

[2] Alberto Camacho, Jacob Varley, Deepali Jain, Atil Iscen, and Dmitry Kalashnikov. "Disentangled Planning and Control in Vision Based Robotics via Reward Machines". In: *Deep Reinforcement Learning Workshop at Neural Information Processing Systems* (2020). arXiv: 2012.14464 [cs.RO].

[3] Petr Čížek, Diar Masri, and Jan Faigl. "Foothold placement planning with a hexapod crawling robot". In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2017, pp. 4096–4101. DOI: 10.1109/IROS.2017.8206267.

[4] Zipeng Fu, Ashish Kumar, Jitendra Malik, and Deepak Pathak. "Minimizing Energy Consumption Leads to the Emergence of Gaits in Legged Robots". In: *Conference on Robot Learning (CoRL)* (2021).

[5] Siddhant Gangapurwala, Mathieu Geisert, Romeo Orsolino, Maurice Fallon, and Ioannis Havoutis. "RLOC: Terrain-Aware Legged Locomotion using Reinforcement Learning and Optimal Control". In: *IEEE Transactions on Robotics* (2022). DOI: 10.48550/ARXIV.2012.03094. URL: https://arxiv.org/abs/2012.03094.

[6] David Hoeller, Lorenz Wellhausen, Farbod Farshidian, and Marco Hutter. "Learning a State Representation and Navigation in Cluttered and Dynamic Environments". In: *IEEE Robotics and Automation Letters 2021* (2021). arXiv: 2103.04351 [cs.RO].

[7] Jemin Hwangbo, Joonho Lee, Alexey Dosovitskiy, Dario Bellicoso, Vassilios Tsounis, Vladlen Koltun, and Marco Hutter. "Learning agile and dynamic motor skills for legged robots". In: *Science Robotics* 4.26 (2019). DOI: 10.1126/scirobotics.aau5872. URL: https://doi.org/10.1126/scirobotics.aau5872.

[8] Atil Iscen, Ken Caluwaerts, Jie Tan, Tingnan Zhang, Erwin Coumans, Vikas Sindhwani, and Vincent Vanhoucke. "Policies Modulating Trajectory Generators". In: *Proceedings of The 2nd Conference on Robot Learning* 87 (2019), pp. 916–926. arXiv: 1910.02812 [cs.RO].

[9] Deepali Jain, Atil Iscen, and Ken Caluwaerts. "From Pixels to Legs: Hierarchical Learning of Quadruped Locomotion". In: *4th Conference on Robot Learning (CoRL)* (2020). DOI: 10.48550/ARXIV.2011.11722. URL: https://arxiv.org/abs/2011.11722.

[10] Ashish Kumar, Zipeng Fu, Deepak Pathak, and Jitendra Malik. "RMA: Rapid Motor Adaptation for Legged Robots". In: *Robotics: Science and Systems* (2021). DOI: 10.48550/ARXIV.2107.04034. URL: https://arxiv.org/abs/2107.04034.

[11] S.H. Kwak and R.B. McGhee. "Rule-based motion coordination for a hexapod walking machine". In: *Advanced Robotics* 4.3 (1989), pp. 263–282. DOI: 10.1163/156855390X00297. eprint: https://doi.org/10.1163/156855390X00297. URL: https://doi.org/10.1163/156855390X00297.

[12] Tianyu Li, Roberto Calandra, Deepak Pathak, Yuandong Tian, Franziska Meier, and Akshara Rai. "Planning in Learned Latent Action Spaces for Generalizable Legged Locomotion". In: *IEEE Robotics and Automation Letters* (2020). DOI: 10.48550/ARXIV.2008.11867. URL: https://arxiv.org/abs/2008.11867.

[13] Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, and Gavriel State. *Isaac Gym: High Performance GPU-Based Physics Simulation For Robot Learning*. 2021. arXiv: 2108.10470 [cs.RO].

[14] Wenjuan Ouyang, Haozhen Chi, Jiangnan Pang, Wenyu Liang, and Qinyuan Ren. "Adaptive Locomotion Control of a Hexapod Robot via Bio-Inspired Learning". In: *Frontiers in Neurorobotics* 15 (2021). DOI: 10.3389/fnbot.2021.627157. URL: http://dx.doi.org/10.3389/fnbot.2021.627157.

[15] Nikita Rudin, David Hoeller, Philipp Reist, and Marco Hutter. "Learning to Walk in Minutes Using Massively Parallel Deep Reinforcement Learning". In: *Conference on Robot Learning* (2021). DOI: 10.48550/ARXIV.2109.11978. URL: https://arxiv.org/abs/2109.11978.

[16] 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: *Robotics: Science and Systems* (2018). DOI: 10.48550/ARXIV.1804.10332. URL: https://arxiv.org/abs/1804.10332.

[17] Joanne Truong, Denis Yarats, Tianyu Li, Franziska Meier, Sonia Chernova, Dhruv Batra, and Akshara Rai. "Learning Navigation Skills for Legged Robots with Learned Robot Embeddings". In: *Intelligent Robots and Systems (IROS)* (2021). arXiv: 2011.12255 [cs.RO].

[18] Vassilios Tsounis, Mitja Alge, Joonho Lee, Farbod Farshidian, and Marco Hutter. "DeepGait: Planning and Control of Quadrupedal Gaits using Deep Reinforcement Learning". In: *IEEE International Conference on Robotics and Automation (ICRA)* (2020). arXiv: 1909.08399 [cs.RO].

[19] Binrui Wang, Xiaohong Cui, Jianbo Sun, and Yanfeng Gao. "Parameters optimization of central pattern generators for hexapod robot based on multi-objective genetic algorithm". In: *International Journal of Advanced Robotic Systems* 18.5 (2021). DOI: 10.1177/17298814211044934.

[20] Wenhao Yu, Deepali Jain, Alejandro Escontrela, Atil Iscen, Peng Xu, Erwin Coumans, Sehoon Ha, Jie Tan, and Tingnan Zhang. "Visual-Locomotion: Learning to Walk on Complex Terrains with Vision". In: *5th Annual Conference on Robot Learning*. 2021. URL: https://openreview.net/forum?id=NDYbXf-DvwZ.

Appendix A

Custom URDF Robot Representation

```
<?xml version="1.0" ?>
<robot name="pexod" xmlns:xacro="http://www.ros.org/wiki/xacro">
  <!-- <xacro:include filename="$(find pexod_description)/urdf/pexod_control.xacro" /> -
  <!-- MATERIALS -->
  <material name="Blue">
    <color rgba="0 0 1 1"/>
  </material>
  <material name="Red">
    <color rgba="1 0 0 1"/>
  </material>
  <material name="Green">
    <color rgba="0 1 0 1"/>
  </material>
  <material name="Yellow">
    <color rgba="1 1 0 1"/>
  </material>
  <material name="LightGrey">
    <color rgba="0.6 0.6 0.6 1.0"/>
  </material>
  <!-- END OF MATERIALS -->
  <!-- XACRO MACROS FOR VISUALS AND COLLISIONS -->
  <!-- END OF XACRO MACROS -->
  <!-- TORSO -->
  <link name="base_link">
    <visual>
      <origin rpy="0 0 0" xyz="0 0 0"/>
      <geometry>
        <box size="0.239 0.117 0.095"/>
      </geometry>
```

```
    <material name="Blue"/>
  </visual>
  <collision>
    <origin rpy="0 0 0" xyz="0 0 0"/>
    <geometry>
<!-- END OF LEG LINKS/JOINTS -->
      <box size="0.239 0.117 0.095"/>
    </geometry>
  </collision>
  <inertial>
    <!-- CENTER OF MASS -->
    <origin rpy="0 0 0" xyz="0 0 0"/>
    <!-- <mass value="0.01"/> -->
    <mass value="1.396"/>
    <!-- box inertia: 1/12*m(y^2+z^2), ... -->
    <inertia ixx="0.003" ixy="0" ixz="0" iyy="0.008" iyz="0" izz="0.008"/>
  </inertial>
</link>
<!-- XACRO MACRO FOR LEGS LINKS/JOINTS -->
<joint name="body_leg_2" type="revolute">
  <parent link="base_link"/>
  <child link="leg_2_1"/>
  <limit effort="1.961" lower="-2.0944" upper="2.0944" velocity="5.236"/>
  <origin rpy="0 0 -0.785" xyz="0.119 0.0585 -0.025"/>
  <!-- <xacro:if value="${index == 5 or index == 4 or index == 3}">
              <axis xyz="0 0 1"/>
          </xacro:if>
          <xacro:if value="${index == 2 or index == 1 or index == 0}"> -->
  <axis xyz="0 0 1"/>
  <!-- </xacro:if> -->
  <dynamics damping="0" friction="0"/>
</joint>
<link name="leg_2_1">
  <visual>
    <origin rpy="1.57079632679 0 0" xyz="0 0.0225 0"/>
    <geometry>
      <box size="0.055 0.02 0.045"/>
      <!-- <cylinder length="0.04" radius="0.02"/> -->
    </geometry>
    <material name="Red"/>
  </visual>
```

```
    <collision>
      <origin rpy="1.57079632679 0 0" xyz="0 0.0225 0"/>
      <geometry>
        <box size="0.055 0.02 0.045"/>
      </geometry>
    </collision>
    <inertial>
      <!-- CENTER OF MASS -->
      <origin rpy="1.57079632679 0 0" xyz="0 0.0225 0"/>
      <mass value="0.01"/>
      <!-- box inertia: 1/12*m(y^2+z^2), ... -->
      <inertia ixx="2.021e-6" ixy="0" ixz="0" iyy="4.208e-6" iyz="0" izz="2.854e-6"/>
    </inertial>
  </link>
  <joint name="leg_2_1_2" type="revolute">
    <parent link="leg_2_1"/>
    <child link="leg_2_2"/>
    <limit effort="1.961" lower="-2.0944" upper="2.0944" velocity="5.236"/>
    <origin rpy="0 0 0" xyz="0 0.045 0"/>
    <axis xyz="1 0 0"/>
    <dynamics damping="0" friction="0"/>
  </joint>
  <link name="leg_2_2">
    <visual>
      <origin rpy="1.57079632679 0 0" xyz="0 0.045 0"/>
      <geometry>
        <box size="0.044 0.02 0.09"/>
        <!-- <cylinder length="0.09" radius="0.02"/> -->
      </geometry>
      <material name="Blue"/>
    </visual>
    <collision>
      <origin rpy="1.57079632679 0 0" xyz="0 0.045 0"/>
      <geometry>
        <box size="0.044 0.02 0.09"/>
      </geometry>
    </collision>
    <inertial>
      <!-- CENTER OF MASS -->
      <origin rpy="1.57079632679 0 0" xyz="0 0.045 0"/>
      <mass value="0.124"/>
      <!-- box inertia: 1/12*m(y^2+z^2), ... -->
```

```
    <inertia ixx="0.00008783" ixy="0" ixz="0" iyy="1.037e-4" iyz="0" izz="2.414e-5"/>
  </inertial>
</link>
<joint name="leg_2_2_3" type="revolute">
  <parent link="leg_2_2"/>
  <child link="leg_2_3"/>
  <limit effort="1.961" lower="-2.0944" upper="2.0944" velocity="5.236"/>
  <origin rpy="0 0 0" xyz="0 0.09 0"/>
  <axis xyz="-1 0 0"/>
  <dynamics damping="0" friction="0"/>
</joint>
<link name="leg_2_3">
  <visual>
    <origin rpy="0.785 0 0" xyz="0 0.0441941738 -0.0441941738"/>
    <geometry>
      <box size="0.055 0.04 0.125"/>
      <!-- <cylinder length="0.125" radius="0.025"/> -->
    </geometry>
    <material name="Red"/>
  </visual>

  <collision>
    <origin rpy="0.785 0 0" xyz="0 0.0441941738 -0.0441941738"/>
    <geometry>
      <box size="0.055 0.04 0.125"/>
    </geometry>
  </collision>

  <inertial>
    <!-- CENTER OF MASS -->
    <origin rpy="0.785 0 0" xyz="0 0.0441941738 -0.0441941738"/>
    <mass value="0.01"/>
    <!-- box inertia: 1/12*m(y^2+z^2), ... -->
    <inertia ixx="1.435e-5" ixy="0" ixz="0" iyy="1.554e-5" iyz="0" izz="0.00000385"/>
  </inertial>
</link>

<joint name="joint_2_eef" type="fixed">
  <parent link="leg_2_3"/>
  <child link="dummy_eef_2"/>
  <origin rpy="0.785 0 0" xyz="0 0.0883883476 -0.0883883476"/>
</joint>
```

```
  <link name="dummy_eef_2">
<visual>
      <geometry>
        <sphere radius="0.005"/>
      </geometry>
      <material name="Green"/>
    </visual>
    <inertial>
      <!-- CENTER OF MASS -->
      <origin rpy="0 0 0" xyz="0 0 0"/>
      <mass value="0"/>
      <inertia ixx="0" ixy="0" ixz="0" iyy="0" iyz="0" izz="0"/>
    </inertial>
  </link>

  <joint name="body_leg_3" type="revolute">
    <parent link="base_link"/>
    <child link="leg_3_1"/>
    <limit effort="1.961" lower="-2.0944" upper="2.0944" velocity="5.236"/>
    <origin rpy="0 0 0.785" xyz="0.119 -0.0585 -0.025"/>
    <!-- <xacro:if value="${index == 5 or index == 4 or index == 3}">
                <axis xyz="0 0 1"/>
            </xacro:if>
            <xacro:if value="${index == 2 or index == 1 or index == 0}"> -->
    <axis xyz="0 0 -1"/>
    <!-- </xacro:if> -->
    <dynamics damping="0" friction="0"/>
  </joint>
  <link name="leg_3_1">
    <visual>
      <origin rpy="1.57079632679 0 0" xyz="0 -0.0225 0"/>
      <geometry>
        <box size="0.055 0.02 0.045"/>
        <!-- <cylinder length="0.04" radius="0.02"/> -->
      </geometry>
      <material name="Red"/>
    </visual>
    <collision>
      <origin rpy="1.57079632679 0 0" xyz="0 -0.0225 0"/>
      <geometry>
        <box size="0.055 0.02 0.045"/>
```

```
    </geometry>
  </collision>
  <inertial>
    <!-- CENTER OF MASS -->
    <origin rpy="1.57079632679 0 0" xyz="0 -0.0225 0"/>
    <mass value="0.01"/>
    <!-- box inertia: 1/12*m(y^2+z^2), ... -->
    <inertia ixx="2.021e-6" ixy="0" ixz="0" iyy="4.208e-6" iyz="0" izz="2.854e-6"/>
  </inertial>
</link>
<joint name="leg_3_1_2" type="revolute">
  <parent link="leg_3_1"/>
  <child link="leg_3_2"/>
  <limit effort="1.961" lower="-2.0944" upper="2.0944" velocity="5.236"/>
  <origin rpy="0 0 0" xyz="0 -0.045 0"/>
  <axis xyz="-1 0 0"/>
  <dynamics damping="0" friction="0"/>
</joint>
<link name="leg_3_2">
  <visual>
    <origin rpy="1.57079632679 0 0" xyz="0 -0.045 0"/>
    <geometry>
      <box size="0.044 0.02 0.09"/>
      <!-- <cylinder length="0.09" radius="0.02"/> -->
    </geometry>
    <material name="Blue"/>
  </visual>
  <collision>
    <origin rpy="1.57079632679 0 0" xyz="0 -0.045 0"/>
    <geometry>
      <box size="0.044 0.02 0.09"/>
    </geometry>
  </collision>
  <inertial>
    <!-- CENTER OF MASS -->
    <origin rpy="1.57079632679 0 0" xyz="0 -0.045 0"/>
    <mass value="0.124"/>
    <!-- box inertia: 1/12*m(y^2+z^2), ... -->
    <inertia ixx="0.00008783" ixy="0" ixz="0" iyy="1.037e-4" iyz="0" izz="2.414e-5"/>
  </inertial>
</link>
<joint name="leg_3_2_3" type="revolute">
```

```
    <parent link="leg_3_2"/>
    <child link="leg_3_3"/>
    <limit effort="1.961" lower="-2.0944" upper="2.0944" velocity="5.236"/>
    <origin rpy="0 0 0" xyz="0 -0.09 0"/>
    <axis xyz="1 0 0"/>
    <dynamics damping="0" friction="0"/>
  </joint>
  <link name="leg_3_3">
    <visual>
      <origin rpy="-0.785 0 0" xyz="0 -0.0441941738 -0.0441941738"/>
      <geometry>
        <box size="0.055 0.04 0.125"/>
        <!-- <cylinder length="0.125" radius="0.025"/> -->
      </geometry>
      <material name="Red"/>
    </visual>

    <collision>
      <origin rpy="-0.785 0 0" xyz="0 -0.0441941738 -0.0441941738"/>
      <geometry>
        <box size="0.055 0.04 0.125"/>
      </geometry>
    </collision>

    <inertial>
      <!-- CENTER OF MASS -->
      <origin rpy="-0.785 0 0" xyz="0 -0.0441941738 -0.0441941738"/>
      <mass value="0.01"/>
      <!-- box inertia: 1/12*m(y^2+z^2), ... -->
      <inertia ixx="1.435e-5" ixy="0" ixz="0" iyy="1.554e-5" iyz="0" izz="0.00000385"/>
    </inertial>
  </link>

  <joint name="joint_3_eef" type="fixed">
    <parent link="leg_3_3"/>
    <child link="dummy_eef_3"/>
    <origin rpy="-0.785 0 0" xyz="0 -0.0883883476 -0.0883883476"/>
  </joint>

  <link name="dummy_eef_3">
  <visual>
      <geometry>
```

```
      <sphere radius="0.005"/>
    </geometry>
    <material name="Green"/>
  </visual>
  <inertial>
    <!-- CENTER OF MASS -->
    <origin rpy="0 0 0" xyz="0 0 0"/>
    <mass value="0"/>
    <inertia ixx="0" ixy="0" ixz="0" iyy="0" iyz="0" izz="0"/>
  </inertial>
</link>

<joint name="body_leg_0" type="revolute">
  <parent link="base_link"/>
  <child link="leg_0_1"/>
  <limit effort="1.961" lower="-2.0944" upper="2.0944" velocity="5.236"/>
  <origin rpy="0 0 0.785" xyz="-0.119 0.0585 -0.025"/>
  <!-- <xacro:if value="${index == 5 or index == 4 or index == 3}">
            <axis xyz="0 0 1"/>
        </xacro:if>
        <xacro:if value="${index == 2 or index == 1 or index == 0}"> -->
  <axis xyz="0 0 1"/>
  <!-- </xacro:if> -->
  <dynamics damping="0" friction="0"/>
</joint>
<link name="leg_0_1">
  <visual>
    <origin rpy="1.57079632679 0 0" xyz="0 0.0225 0"/>
    <geometry>
      <box size="0.055 0.02 0.045"/>
      <!-- <cylinder length="0.04" radius="0.02"/> -->
    </geometry>
    <material name="Red"/>
  </visual>
  <collision>
    <origin rpy="1.57079632679 0 0" xyz="0 0.0225 0"/>
    <geometry>
      <box size="0.055 0.02 0.045"/>
    </geometry>
  </collision>
  <inertial>
    <!-- CENTER OF MASS -->
```

```
    <origin rpy="1.57079632679 0 0" xyz="0 0.0225 0"/>
    <mass value="0.01"/>
    <!-- box inertia: 1/12*m(y^2+z^2), ... -->
    <inertia ixx="2.021e-6" ixy="0" ixz="0" iyy="4.208e-6" iyz="0" izz="2.854e-6"/>
  </inertial>
</link>
<joint name="leg_0_1_2" type="revolute">
  <parent link="leg_0_1"/>
  <child link="leg_0_2"/>
  <limit effort="1.961" lower="-2.0944" upper="2.0944" velocity="5.236"/>
  <origin rpy="0 0 0" xyz="0 0.045 0"/>
  <axis xyz="1 0 0"/>
  <dynamics damping="0" friction="0"/>
</joint>
<link name="leg_0_2">
  <visual>
    <origin rpy="1.57079632679 0 0" xyz="0 0.045 0"/>
    <geometry>
      <box size="0.044 0.02 0.09"/>
      <!-- <cylinder length="0.09" radius="0.02"/> -->
    </geometry>
    <material name="Blue"/>
  </visual>
  <collision>
    <origin rpy="1.57079632679 0 0" xyz="0 0.045 0"/>
    <geometry>
      <box size="0.044 0.02 0.09"/>
    </geometry>
  </collision>
  <inertial>
    <!-- CENTER OF MASS -->
    <origin rpy="1.57079632679 0 0" xyz="0 0.045 0"/>
    <mass value="0.124"/>
    <!-- box inertia: 1/12*m(y^2+z^2), ... -->
    <inertia ixx="0.00008783" ixy="0" ixz="0" iyy="1.037e-4" iyz="0" izz="2.414e-5"/>
  </inertial>
</link>
<joint name="leg_0_2_3" type="revolute">
  <parent link="leg_0_2"/>
  <child link="leg_0_3"/>
  <limit effort="1.961" lower="-2.0944" upper="2.0944" velocity="5.236"/>
  <origin rpy="0 0 0" xyz="0 0.09 0"/>
```

```
    <axis xyz="-1 0 0"/>
    <dynamics damping="0" friction="0"/>
  </joint>
  <link name="leg_0_3">
    <visual>
      <origin rpy="0.785 0 0" xyz="0 0.0441941738 -0.0441941738"/>
      <geometry>
        <box size="0.055 0.04 0.125"/>
        <!-- <cylinder length="0.125" radius="0.025"/> -->
      </geometry>
      <material name="Red"/>
    </visual>

    <collision>
      <origin rpy="0.785 0 0" xyz="0 0.0441941738 -0.0441941738"/>
      <geometry>
        <box size="0.055 0.04 0.125"/>
      </geometry>
    </collision>

    <inertial>
      <!-- CENTER OF MASS -->
      <origin rpy="0.785 0 0" xyz="0 0.0441941738 -0.0441941738"/>
      <mass value="0.01"/>
      <!-- box inertia: 1/12*m(y^2+z^2), ... -->
      <inertia ixx="1.435e-5" ixy="0" ixz="0" iyy="1.554e-5" iyz="0" izz="0.00000385"/>
    </inertial>
  </link>

  <joint name="joint_0_eef" type="fixed">
    <parent link="leg_0_3"/>
    <child link="dummy_eef_0"/>
    <origin rpy="0.785 0 0" xyz="0 0.0883883476 -0.0883883476"/>
  </joint>

  <link name="dummy_eef_0">
  <visual>
      <geometry>
        <sphere radius="0.005"/>
      </geometry>
      <material name="Green"/>
    </visual>
```

```
    <inertial>
      <!-- CENTER OF MASS -->
      <origin rpy="0 0 0" xyz="0 0 0"/>
      <mass value="0"/>
      <inertia ixx="0" ixy="0" ixz="0" iyy="0" iyz="0" izz="0"/>
    </inertial>
  </link>

  <joint name="body_leg_5" type="revolute">
    <parent link="base_link"/>
    <child link="leg_5_1"/>
    <limit effort="1.961" lower="-2.0944" upper="2.0944" velocity="5.236"/>
    <origin rpy="0 0 -0.785" xyz="-0.119 -0.0585 -0.025"/>
    <!-- <xacro:if value="${index == 5 or index == 4 or index == 3}">
                <axis xyz="0 0 1"/>
            </xacro:if>
            <xacro:if value="${index == 2 or index == 1 or index == 0}"> -->
    <axis xyz="0 0 -1"/>
    <!-- </xacro:if> -->
    <dynamics damping="0" friction="0"/>
  </joint>
  <link name="leg_5_1">
    <visual>
      <origin rpy="1.57079632679 0 0" xyz="0 -0.0225 0"/>
      <geometry>
        <box size="0.055 0.02 0.045"/>
        <!-- <cylinder length="0.04" radius="0.02"/> -->
      </geometry>
      <material name="Red"/>
    </visual>
    <collision>
      <origin rpy="1.57079632679 0 0" xyz="0 -0.0225 0"/>
      <geometry>
        <box size="0.055 0.02 0.045"/>
      </geometry>
    </collision>
    <inertial>
      <!-- CENTER OF MASS -->
      <origin rpy="1.57079632679 0 0" xyz="0 -0.0225 0"/>
      <mass value="0.01"/>
      <!-- box inertia: 1/12*m(y^2+z^2), ... -->
      <inertia ixx="2.021e-6" ixy="0" ixz="0" iyy="4.208e-6" iyz="0" izz="2.854e-6"/>
```

```
    </inertial>
  </link>
  <joint name="leg_5_1_2" type="revolute">
    <parent link="leg_5_1"/>
    <child link="leg_5_2"/>
    <limit effort="1.961" lower="-2.0944" upper="2.0944" velocity="5.236"/>
    <origin rpy="0 0 0" xyz="0 -0.045 0"/>
    <axis xyz="-1 0 0"/>
    <dynamics damping="0" friction="0"/>
  </joint>
  <link name="leg_5_2">
    <visual>
      <origin rpy="1.57079632679 0 0" xyz="0 -0.045 0"/>
      <geometry>
        <box size="0.044 0.02 0.09"/>
        <!-- <cylinder length="0.09" radius="0.02"/> -->
      </geometry>
      <material name="Blue"/>
    </visual>
    <collision>
      <origin rpy="1.57079632679 0 0" xyz="0 -0.045 0"/>
      <geometry>
        <box size="0.044 0.02 0.09"/>
      </geometry>
    </collision>
    <inertial>
      <!-- CENTER OF MASS -->
      <origin rpy="1.57079632679 0 0" xyz="0 -0.045 0"/>
      <mass value="0.124"/>
      <!-- box inertia: 1/12*m(y^2+z^2), ... -->
      <inertia ixx="0.00008783" ixy="0" ixz="0" iyy="1.037e-4" iyz="0" izz="2.414e-5"/>
    </inertial>
  </link>
  <joint name="leg_5_2_3" type="revolute">
    <parent link="leg_5_2"/>
    <child link="leg_5_3"/>
    <limit effort="1.961" lower="-2.0944" upper="2.0944" velocity="5.236"/>
    <origin rpy="0 0 0" xyz="0 -0.09 0"/>
    <axis xyz="1 0 0"/>
    <dynamics damping="0" friction="0"/>
  </joint>
  <link name="leg_5_3">
```

```
    <visual>
      <origin rpy="-0.785 0 0" xyz="0 -0.0441941738 -0.0441941738"/>
      <geometry>
        <box size="0.055 0.04 0.125"/>
        <!-- <cylinder length="0.125" radius="0.025"/> -->
      </geometry>
      <material name="Red"/>
    </visual>

    <collision>
      <origin rpy="-0.785 0 0" xyz="0 -0.0441941738 -0.0441941738"/>
      <geometry>
        <box size="0.055 0.04 0.125"/>
      </geometry>
    </collision>

    <inertial>
      <!-- CENTER OF MASS -->
      <origin rpy="-0.785 0 0" xyz="0 -0.0441941738 -0.0441941738"/>
      <mass value="0.01"/>
      <!-- box inertia: 1/12*m(y^2+z^2), ... -->
      <inertia ixx="1.435e-5" ixy="0" ixz="0" iyy="1.554e-5" iyz="0" izz="0.00000385"/>
    </inertial>
  </link>

  <joint name="joint_5_eef" type="fixed">
    <parent link="leg_5_3"/>
    <child link="dummy_eef_5"/>
    <origin rpy="-0.785 0 0" xyz="0 -0.0883883476 -0.0883883476"/>
  </joint>

  <link name="dummy_eef_5">
  <visual>
      <geometry>
        <sphere radius="0.005"/>
      </geometry>
      <material name="Green"/>
    </visual>
    <inertial>
      <!-- CENTER OF MASS -->
      <origin rpy="0 0 0" xyz="0 0 0"/>
      <mass value="0"/>
```

```
    <inertia ixx="0" ixy="0" ixz="0" iyy="0" iyz="0" izz="0"/>
  </inertial>
</link>

<joint name="body_leg_1" type="revolute">
  <parent link="base_link"/>
  <child link="leg_1_1"/>
  <limit effort="1.961" lower="-2.0944" upper="2.0944" velocity="5.236"/>
  <origin rpy="0 0 0" xyz="0 0.0915 -0.025"/>
  <!-- <xacro:if value="${index == 5 or index == 4 or index == 3}">
            <axis xyz="0 0 1"/>
        </xacro:if>
        <xacro:if value="${index == 2 or index == 1 or index == 0}"> -->
  <axis xyz="0 0 1"/>
  <!-- </xacro:if> -->
  <dynamics damping="0" friction="0"/>
</joint>
<link name="leg_1_1">
  <visual>
    <origin rpy="1.57079632679 0 0" xyz="0 0.0225 0"/>
    <geometry>
      <box size="0.055 0.02 0.045"/>
      <!-- <cylinder length="0.04" radius="0.02"/> -->
    </geometry>
    <material name="Red"/>
  </visual>
  <collision>
    <origin rpy="1.57079632679 0 0" xyz="0 0.0225 0"/>
    <geometry>
      <box size="0.055 0.02 0.045"/>
    </geometry>
  </collision>
  <inertial>
    <!-- CENTER OF MASS -->
    <origin rpy="1.57079632679 0 0" xyz="0 0.0225 0"/>
    <mass value="0.01"/>
    <!-- box inertia: 1/12*m(y^2+z^2), ... -->
    <inertia ixx="2.021e-6" ixy="0" ixz="0" iyy="4.208e-6" iyz="0" izz="2.854e-6"/>
  </inertial>
</link>
<joint name="leg_1_1_2" type="revolute">
  <parent link="leg_1_1"/>
```

```
    <child link="leg_1_2"/>
    <limit effort="1.961" lower="-2.0944" upper="2.0944" velocity="5.236"/>
    <origin rpy="0 0 0" xyz="0 0.045 0"/>
    <axis xyz="1 0 0"/>
    <dynamics damping="0" friction="0"/>
  </joint>
  <link name="leg_1_2">
    <visual>
      <origin rpy="1.57079632679 0 0" xyz="0 0.045 0"/>
      <geometry>
        <box size="0.044 0.02 0.09"/>
        <!-- <cylinder length="0.09" radius="0.02"/> -->
      </geometry>
      <material name="Blue"/>
    </visual>
    <collision>
      <origin rpy="1.57079632679 0 0" xyz="0 0.045 0"/>
      <geometry>
        <box size="0.044 0.02 0.09"/>
      </geometry>
    </collision>
    <inertial>
      <!-- CENTER OF MASS -->
      <origin rpy="1.57079632679 0 0" xyz="0 0.045 0"/>
      <mass value="0.124"/>
      <!-- box inertia: 1/12*m(y^2+z^2), ... -->
      <inertia ixx="0.00008783" ixy="0" ixz="0" iyy="1.037e-4" iyz="0" izz="2.414e-5"/>
    </inertial>
  </link>
  <joint name="leg_1_2_3" type="revolute">
    <parent link="leg_1_2"/>
    <child link="leg_1_3"/>
    <limit effort="1.961" lower="-2.0944" upper="2.0944" velocity="5.236"/>
    <origin rpy="0 0 0" xyz="0 0.09 0"/>
    <axis xyz="-1 0 0"/>
    <dynamics damping="0" friction="0"/>
  </joint>
  <link name="leg_1_3">
    <visual>
      <origin rpy="0.785 0 0" xyz="0 0.0441941738 -0.0441941738"/>
      <geometry>
        <box size="0.055 0.04 0.125"/>
```

```
      <!-- <cylinder length="0.125" radius="0.025"/> -->
    </geometry>
    <material name="Red"/>
  </visual>

  <collision>
    <origin rpy="0.785 0 0" xyz="0 0.0441941738 -0.0441941738"/>
    <geometry>
      <box size="0.055 0.04 0.125"/>
    </geometry>
  </collision>

  <inertial>
    <!-- CENTER OF MASS -->
    <origin rpy="0.785 0 0" xyz="0 0.0441941738 -0.0441941738"/>
    <mass value="0.01"/>
    <!-- box inertia: 1/12*m(y^2+z^2), ... -->
    <inertia ixx="1.435e-5" ixy="0" ixz="0" iyy="1.554e-5" iyz="0" izz="0.00000385"/>
  </inertial>
</link>

<joint name="joint_1_eef" type="fixed">
  <parent link="leg_1_3"/>
  <child link="dummy_eef_1"/>
  <origin rpy="0.785 0 0" xyz="0 0.0883883476 -0.0883883476"/>
</joint>

<link name="dummy_eef_1">
<visual>
    <geometry>
      <sphere radius="0.005"/>
    </geometry>
    <material name="Green"/>
  </visual>
  <inertial>
    <!-- CENTER OF MASS -->
    <origin rpy="0 0 0" xyz="0 0 0"/>
    <mass value="0"/>
    <inertia ixx="0" ixy="0" ixz="0" iyy="0" iyz="0" izz="0"/>
  </inertial>
</link>
```

```
<joint name="body_leg_4" type="revolute">
  <parent link="base_link"/>
  <child link="leg_4_1"/>
  <limit effort="1.961" lower="-2.0944" upper="2.0944" velocity="5.236"/>
  <origin rpy="0 0 0" xyz="0 -0.0915 -0.025"/>
  <!-- <xacro:if value="${index == 5 or index == 4 or index == 3}">
              <axis xyz="0 0 1"/>
          </xacro:if>
          <xacro:if value="${index == 2 or index == 1 or index == 0}"> -->
  <axis xyz="0 0 -1"/>
  <!-- </xacro:if> -->
  <dynamics damping="0" friction="0"/>
</joint>
<link name="leg_4_1">
  <visual>
    <origin rpy="1.57079632679 0 0" xyz="0 -0.0225 0"/>
    <geometry>
      <box size="0.055 0.02 0.045"/>
      <!-- <cylinder length="0.04" radius="0.02"/> -->
    </geometry>
    <material name="Red"/>
  </visual>
  <collision>
    <origin rpy="1.57079632679 0 0" xyz="0 -0.0225 0"/>
    <geometry>
      <box size="0.055 0.02 0.045"/>
    </geometry>
  </collision>
  <inertial>
    <!-- CENTER OF MASS -->
    <origin rpy="1.57079632679 0 0" xyz="0 -0.0225 0"/>
    <mass value="0.01"/>
    <!-- box inertia: 1/12*m(y^2+z^2), ... -->
    <inertia ixx="2.021e-6" ixy="0" ixz="0" iyy="4.208e-6" iyz="0" izz="2.854e-6"/>
  </inertial>
</link>
<joint name="leg_4_1_2" type="revolute">
  <parent link="leg_4_1"/>
  <child link="leg_4_2"/>
  <limit effort="1.961" lower="-2.0944" upper="2.0944" velocity="5.236"/>
  <origin rpy="0 0 0" xyz="0 -0.045 0"/>
  <axis xyz="-1 0 0"/>
```

```
    <dynamics damping="0" friction="0"/>
  </joint>
  <link name="leg_4_2">
    <visual>
      <origin rpy="1.57079632679 0 0" xyz="0 -0.045 0"/>
      <geometry>
        <box size="0.044 0.02 0.09"/>
        <!-- <cylinder length="0.09" radius="0.02"/> -->
      </geometry>
      <material name="Blue"/>
    </visual>
    <collision>
      <origin rpy="1.57079632679 0 0" xyz="0 -0.045 0"/>
      <geometry>
        <box size="0.044 0.02 0.09"/>
      </geometry>
    </collision>
    <inertial>
      <!-- CENTER OF MASS -->
      <origin rpy="1.57079632679 0 0" xyz="0 -0.045 0"/>
      <mass value="0.124"/>
      <!-- box inertia: 1/12*m(y^2+z^2), ... -->
      <inertia ixx="0.00008783" ixy="0" ixz="0" iyy="1.037e-4" iyz="0" izz="2.414e-5"/>
    </inertial>
  </link>
  <joint name="leg_4_2_3" type="revolute">
    <parent link="leg_4_2"/>
    <child link="leg_4_3"/>
    <limit effort="1.961" lower="-2.0944" upper="2.0944" velocity="5.236"/>
    <origin rpy="0 0 0" xyz="0 -0.09 0"/>
    <axis xyz="1 0 0"/>
    <dynamics damping="0" friction="0"/>
  </joint>
  <link name="leg_4_3">
    <visual>
      <origin rpy="-0.785 0 0" xyz="0 -0.0441941738 -0.0441941738"/>
      <geometry>
        <box size="0.055 0.04 0.125"/>
        <!-- <cylinder length="0.125" radius="0.025"/> -->
      </geometry>
      <material name="Red"/>
    </visual>
```

```
    <collision>
      <origin rpy="-0.785 0 0" xyz="0 -0.0441941738 -0.0441941738"/>
      <geometry>
        <box size="0.055 0.04 0.125"/>
      </geometry>
    </collision>

    <inertial>
      <!-- CENTER OF MASS -->
      <origin rpy="-0.785 0 0" xyz="0 -0.0441941738 -0.0441941738"/>
      <mass value="0.01"/>
      <!-- box inertia: 1/12*m(y^2+z^2), ... -->
      <inertia ixx="1.435e-5" ixy="0" ixz="0" iyy="1.554e-5" iyz="0" izz="0.00000385"/>
    </inertial>
  </link>

  <joint name="joint_4_eef" type="fixed">
    <parent link="leg_4_3"/>
    <child link="dummy_eef_4"/>
    <origin rpy="-0.785 0 0" xyz="0 -0.0883883476 -0.0883883476"/>
  </joint>

  <link name="dummy_eef_4">
    <visual>
      <geometry>
        <sphere radius="0.005"/>
      </geometry>
      <material name="Green"/>
    </visual>
    <inertial>
      <!-- CENTER OF MASS -->
      <origin rpy="0 0 0" xyz="0 0 0"/>
      <mass value="0"/>
      <inertia ixx="0" ixy="0" ixz="0" iyy="0" iyz="0" izz="0"/>
    </inertial>
  </link>
  <!-- END OF LEG LINKS/JOINTS -->
</robot>
```