

UC Berkeley

UC Berkeley Electronic Theses and Dissertations

Title

Rapid Adaptation for Robot Control

Permalink

<https://escholarship.org/uc/item/12j856f2>

Author

Kumar, Ashish

Publication Date

2023

Supplemental Material

<https://escholarship.org/uc/item/12j856f2#supplemental>

Peer reviewed|Thesis/dissertation

Rapid Adaptation for Robot Control

By

Ashish Kumar

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Jitendra Malik, Chair

Professor Pieter Abbeel

Professor Koushil Sreenath

Summer 2023

Rapid Adaptation for Robot Control

Copyright 2023
by
Ashish Kumar

Abstract

Rapid Adaptation for Robot Control

by

Ashish Kumar

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor Jitendra Malik, Chair

Once robots are deployed in the real world, they will inevitably encounter scenarios that they have never seen before. Consequently, to develop robots that can help us in routine tasks, we need to first make progress towards the problem of generalization in robotics, which asks the question: how do we get robots to handle scenarios beyond what they have seen before? This thesis will present an approach to achieve such generalization through the use of rapid adaptation. Concretely, the proposed algorithm - Rapid Motor Adaptation (RMA) - allows robots to generalize by adapting to new and unseen scenarios in fractions of a second. The proposed algorithm is developed in the context of blind quadruped walking in complex terrain in the real world. Subsequently, it is extended to quadruped walking with egocentric vision, enabling robots to cross very challenging terrains, such as stepping stones, which are beyond the reach of blind systems. The same algorithm is then applied to the task of in-hand rotation to develop a single controller capable of rotating a diverse set of objects in the real world, as well as to bipedal robots.

To my family:

*my parents, Rajeev Kumar and Jayshree Kumari,
and my sister, Amrita Kumari*

for your unconditional love and support

Contents

Contents	ii
1 Introduction	1
2 RMA: Rapid Motor Adaptation for Legged Robots	3
2.1 Rapid Motor Adaptation	7
2.2 Experimental Setup	12
2.3 Results and Analysis	13
2.4 Related Work	16
2.5 Summary	17
3 Emergent walking gaits via Energy Minimization	18
3.1 Method	20
3.2 Experimental Setup and Training Details	23
3.3 Results and Analysis	24
3.4 Related Works	28
3.5 Summary	29
4 Legged Locomotion in Challenging Terrains using Egocentric Vision	30
4.1 Method: Legged Locomotion from Egocentric Vision	32
4.2 Experimental Setup	36
4.3 Results and Analysis	37
4.4 Related Work	39
4.5 Summary	40
5 In-hand Rotation via Rapid Adaptation	41
5.1 Related Work	43
5.2 Rapid Motor Adaptation for In-Hand Object Rotation	44
5.3 Experimental Setup and Implementation Details	46
5.4 Results and Analysis	47
5.5 Summary	52
6 RMA for Bipedal Locomotion	53

6.1	Related Work	55
6.2	General Walking Controller	57
6.3	Walking Controller with Rapid Adaptation	58
6.4	Experimental Setup	59
6.5	Results and Analysis	61
6.6	Summary	64
7	Conclusion	67
	Bibliography	68
A	Chapter 2 Supplementary Material	82
A.1	Metrics	82
A.2	Additional Training and Deployment Details	82
A.3	Additional Real-World Adaptation Analysis	84
A.4	Additional Simulation Testings	84
B	Chapter 3 Supplementary Material	88
B.1	Experimental Setup and Training Details	88
B.2	Details of the MPC baseline	89
B.3	Additional Experiment Results	91
C	Chapter 4 Supplementary Material	92
C.1	Proof of Theorem 3.1	92
C.2	Rewards	94
C.3	Experimental Setup and Implementation Details	95
D	Chapter 5 Supplementary Material	98
D.1	Additional Results and Analysis	98
D.2	Ablation Experiments	99
D.3	Implementation Details	100

Acknowledgments

This thesis would not have been possible without all the amazing people who supported me during my Ph.D. journey. The acknowledgments that follow are far from enough to thank them.

I would like to thank my advisor, Jitendra Malik, whose guidance and support was instrumental in this journey and enabled me to think of high level research directions and goals. I learned a lot through our conversations, and in particular, greatly benefitted from his knowledge of the literature in multiple fields ranging all the way back to 20th century. I am grateful for his time and effort towards mentoring me during my Ph.D. journey.

I would also like to thank Alexei (Alyosha) Efros for being an amazing mentor to me during my time at Berkeley. I have really enjoyed all our late night research and personal conversations. Alyosha is a master of forming provocative opinions that are hard to argue with, which have often forced me to think out of the box.

I have also had the privilege of learning from my amazing collaborators, who added significantly new perspectives to my thinking. The research presented here wouldn't be possible without their efforts and constant motivation. I would like to thank Saurabh Gupta, David Fouhey and Deepak Pathak for helping me grow into a mature researcher during my time at Berkeley, and taking a risk on me when I was starting off from a blank slate. I would also like to thank Prof. Koushil Sreenath, Prof. Claire Tomlin, Prof. Pieter Abbeel, Prof. Mark Mueller, Prof. Ken Goldberg, Roberto Calandra, and Prof. Yi Ma for wonderful collaborations and their guidance during my Ph.D. My deepest thanks go to each of them for encouraging me to strive for better.

I am very grateful for my wonderful colleagues, collaborators, and friends who really enriched my day to day experience through random discussions about research and non-research things. I'd like to thank Angjoo, Pulkit, Shubham Tulsiani, Shubham Goel, Haozhi, Anastasios, Andrea, Antonio, Assaf, Bill, Boyi, Dave, Ethan, Hang, Kartik, Shiry, Tete, Tim, Toru, Utkarsh, Vickie, Vongani, Yossi, Zhe, Sasha, Daisy, Raven, Ananye, Zipeng. Thanks to the Admin team - Angie, Ami, Roxana, Lena, Shirley and Jean for shielding me from bureaucracy and logistics, often without me even knowing it.

I have also been blessed with a wonderful set of friends who helped me along in my personal journey during my Ph.D. I would like to thank Yeshwanth, Armin, Ilija, Evonne, Jasmine, Allan, Yu, Zihao, Ashvin, Nilesh, Melih, Kiran, Divya, Chandan. Thank you all for being by my side through thick and thin, and transforming Berkeley into a home away from home. I would like to thank Ilija and Divya for helping me through some of my difficult times during my Ph.D., and Divya again for her immense help with the making of my robot videos.

I would also like to thank people from my past life (before I started my journey in Berkeley). I'm indebted to my friends, teachers, mentors, and professors (from IIT Jodhpur) for their constant support. I would like to thank Manik for giving me an opportunity to work with him at Microsoft Research before I started my Ph.D. . I learned a lot from him during that time and I attribute a lot of my subsequent successes to him. I would also like to thank Yash, Himanshu, Saurabh, Ankit Anand, Diksha, Ankit Singh, Dhiraj, Tavish, Harshit, Kishan, Maninder, Shivam, Abhishek Pilania, Deep, Debashish, Atul, Abhishek Bassan, Akhil, Arpit and Siddharth for being wonderful friends. I

would also like to thank my cousins for the amazing time we spent together over the years every time I visited back home.

Finally, I would like to thank my family for their unwavering faith and support in me. I would like to thank my father and mother, Rajeev and Jayshree, who have been absolutely instrumental in teaching me almost everything I know. They have been a constant source of wisdom over the years. My mother taught me how to be a kind person and how to accept constructive criticism well. She always believed in me and that gave me a certain confidence which has stayed with me and has been an important part of my successful journey. My father has been immensely helpful in helping me make difficult decisions in my life, and has always supported me in my goals, be it financially or otherwise. This has allowed me to stay focussed professionally over the years. Talking to him has always kept me grounded and made me feel safe. To my parents, words fall short to express my gratitude. Your sacrifices, love, and relentless belief in me have been the bedrock of my journey. I would also like to thank my sister, who has always added color and joy in my life, and has supported me through thick and thin. She made sure to always be just a phone call away, always available to cheer me up and motivate me through difficult times. I'm truly blessed to have such an amazing sister and wonderful parents.

Chapter 1

Introduction

How can we build a robot capable of solving a general set of tasks – a single robot system that can cook, clean, help in construction, factory assembly, search-and-rescue, elderly care, etc., and do so in a very diverse set of environments? If we look at the state of robots today, we have indeed made quite a bit of progress toward this goal. We have robots that can pick and place objects in factories, drones that can do inspections, robot arms that pour water into a glass, and robots that can even solve Rubik’s cube (in less than a second)! However, each of these requires a separate specialized solution that fails to work on other tasks, or for that matter, even beyond the environment they were designed for. Humans, on the other hand, are an existence proof that a single system capable of solving a very general set of tasks in a very diverse set of environments can exist. Hence, we humans are a lower bound to the level of generality and proficiency that can potentially be achieved by an artificial system. However, what makes this really challenging is the sheer complexity of the real world that will inevitably expose such a robot to a new and unseen scenario once it is deployed in the real world. This makes it nearly impossible to anticipate all scenarios that the robot is likely going to encounter once deployed. Consequently, to make significant advances toward truly general robots, we need generalization – the ability to work in scenarios that are different than what was encountered before.

This thesis presents a line of work that approaches generalization via rapid adaptation. The key idea is to develop controllers that can adapt to new, unseen scenarios in fractions of a second and hence can generalize to a new environment by Rapid Adaptation. I will first develop it in the context of a blind quadruped for the task of walking on challenging terrains, and then show how the same algorithm can enable a) an anthropomorphic hand to rotate a diverse set of objects in-hand b) a biped robot to walk on challenging terrains and pull varying payloads. I will then scale this idea of rapid adaptation to quadrupeds with visual sensing to achieve walking controllers capable of crossing stepping stones and other challenging terrains which were beyond the reach of the blind robot. The underlying algorithm for all these applications is RMA (Rapid Motor Adaptation), and its design is driven by the following three important principles:

- Using large-scale data-driven search, instead of heuristics, to learn a controller in a principled way. For example, most works employ simplified dynamics models and gait heuristics to

develop a walking controller, whereas, we will use end-to-end reinforcement learning without any heuristic priors to learn a walking policy in a principled data-driven way.

- Using energy minimization as our primary reward shaping to allow the robot to achieve natural-looking behaviors, instead of a complex and heavily shaped reward.
- Developing an adaptive system that is aware of the environment it is dealing with and can adapt to it in fractions of a second.

With these principles in mind, in chapter 2, we will describe the RMA algorithm in the context of a blind quadruped robot that only has access to its own joint angles (proprioception). But how far can we go with such a limited system? A blind system that can only feel its limbs (i.e. a proprioception-only system) and has no knowledge of the terrain must walk conservatively to avoid a fall, and can only traverse simple terrains. Instead of using a single conservative gait for all scenarios, what we really want is an adaptive system that continuously estimates the local environment to adjust its gait. But how can we estimate this for systems with only proprioception? While walking on a slippery surface, our feet slip and move further than they are commanded to, and this discrepancy between what was commanded and what actually happened leaks information about the environment. Chapter 2 operationalizes this idea in the context of quadruped walking on complex terrains and presents the details of the Rapid Motor Adaptation (RMA) algorithm [video]. In chapter 3, we will analyze the reward function that guides reinforcement learning in RMA. We find that a reward function that simply penalizes the energy consumed by the robot while rewarding the robot to track target velocities can lead to emergent walking gaits such as walking, trotting, and galloping without the need for any pre-programming or gait heuristics [video]. In chapter 4, we will extend the same design principles to a high dimensional input – egocentric onboard vision. We will develop an end-to-end system that can jointly process visual and proprioceptive inputs to walk on very complex terrains such as stairs almost as high as the robot, slippery slopes, debris, and stepping stones (set up in my living room!) [video]. Our approach is in contrast to most learning-based and classical approaches which use vision through intermediate terrain maps. Constructing these terrain maps on the fly is a very challenging problem, and as we will demonstrate, also not necessary to solve the downstream task of walking. On the contrary, we find that not relying on intermediate terrain maps improves the overall performance of the system (chapter 4).

Since RMA algorithm makes no assumptions about the task of walking, or four-legged systems, we can also apply it to other tasks in robotics. Concretely, we will apply it to the task of dexterous manipulation to enable a four-fingered hand to rotate a diverse set of objects in the real world (chapter 5) [video], to the task of drone flight and learn a single universal low-level controller capable of flying drones with very diverse morphologies [200] and to bipeds (chapter 6) [video].

Chapter 2

RMA: Rapid Motor Adaptation for Legged Robots

In this chapter, we will formalize the idea of rapid adaptation and apply to the challenging task of quadruped locomotion. Great progress has been made in legged robotics over the last forty years through the modeling of physical dynamics and the tools of control theory [119, 146, 155, 55, 192, 203, 166, 80, 85, 10, 73]. These methods require considerable expertise on the part of the human designer, and in recent years there has been much interest in replicating this success using reinforcement learning and imitation learning techniques [72, 62, 136, 190, 102] which could lower this burden, and perhaps also improve performance. The standard paradigm is to train an RL-based controller in a physics simulation environment and then transfer to the real world using various sim-to-real techniques [171, 137, 72]. This transfer has proven quite challenging, because the sim-to-real gap itself is the result of multiple factors: (a) the physical robot and its model in the simulator differ significantly; (b) real-world terrains vary considerably (Figure 2.1) from our models of these in the simulator; (c) the physics simulator fails to accurately capture the physics of the real world – we are dealing here with contact forces, deformable surfaces and the like – a considerably harder problem than modeling rigid bodies moving in free space.

We will study this problem on a relatively cheap A1 robot from Unitree as our experimental platform. Figure 2.1 shows some sample examples with in-action results in the video. Before outlining our approach (Figure 2.2), we begin by noting that human walking in the real world entails rapid adaptation as we move on different soils, uphill or downhill, carrying loads, with rested or tired muscles, and coping with sprained ankles and the like. Let us focus on this as a central problem for legged robots as well, and call it **Rapid Motor Adaptation (RMA)**. We will posit that RMA has to occur online, at a time scale of fractions of a second, which implies that we have no time to carry out multiple experiments in the physical world, rolling out multiple trajectories and optimizing to estimate various system parameters. It may be worse than that. If we introduce the quadruped onto a rocky surface with no prior experience, the robot policy would fail often, causing serious

This chapter is based on joint work with Zipeng Fu, Deepak Pathak and Jitendra Malik [96]

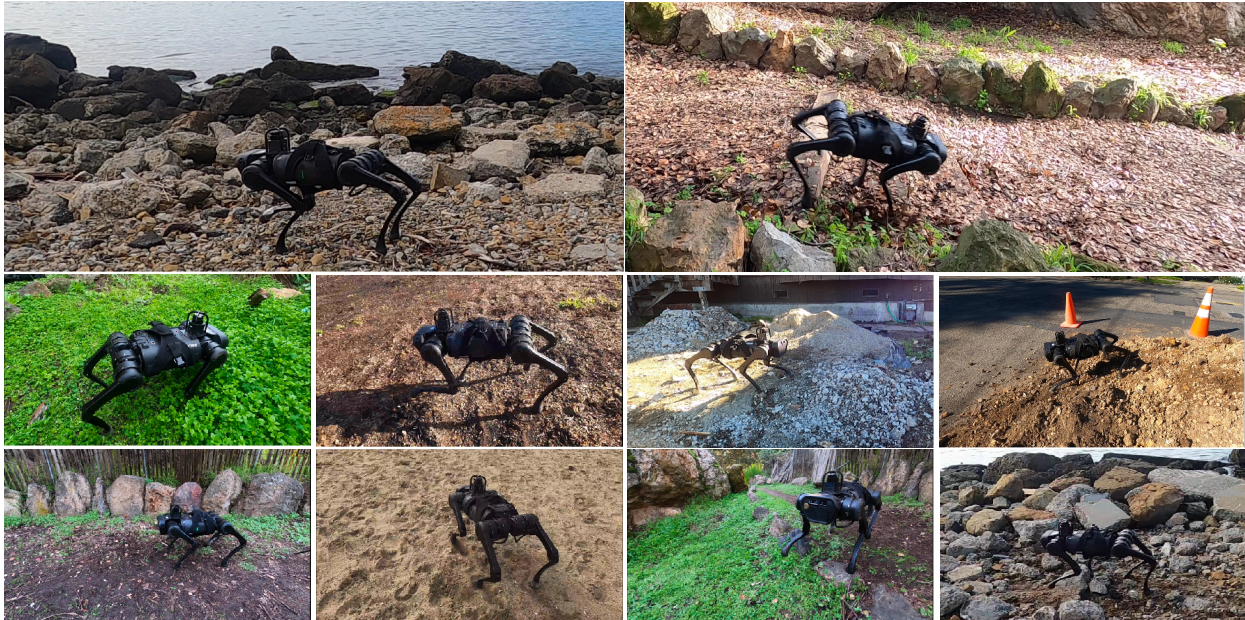


Figure 2.1: We demonstrate the performance of RMA on several challenging environments. The robot is successfully able to walk on sand, mud, hiking trails, tall grass and dirt pile without a single failure in all our trials. The robot was successful in 70% of the trials when walking down stairs along a hiking trail, and succeeded in 80% of the trials when walking across a cement pile and a pile of pebbles. The robot achieves this high success rate despite never having seen unstable or sinking ground, obstructive vegetation or stairs during training. All deployment results are with the same policy without any simulation calibration, or real-world fine-tuning. Videos at <https://ashish-kmr.github.io/rma-legged-robots/>.

damage to the robot. Collecting even 3-5 mins of walking data in order to adapt the walking policy may be practically infeasible. Our strategy therefore entails that not just the basic walking policy, but also RMA must be trained in simulation, and directly deployed in the real world. But, how?

Figure 2.2 shows that RMA consists of two subsystems: the base policy π and the adaptation module ϕ , which work together to enable online real time adaptation on a very diverse set of environment configurations. The base policy is trained via reinforcement learning in simulation using privileged information about the environment configuration e_t such as friction, payload, etc. Knowledge of the vector e_t allows the base policy to appropriately adapt to the given environment. The environment configuration vector e_t is first encoded into a latent feature space z_t using an encoder network μ . This latent vector z_t , which we call the *extrinsics*, is then fed into the base policy along with the current state x_t and the previous action a_{t-1} . The base policy then predicts the desired joint positions of the robot a_t . The policy π and the environmental factor encoder μ are

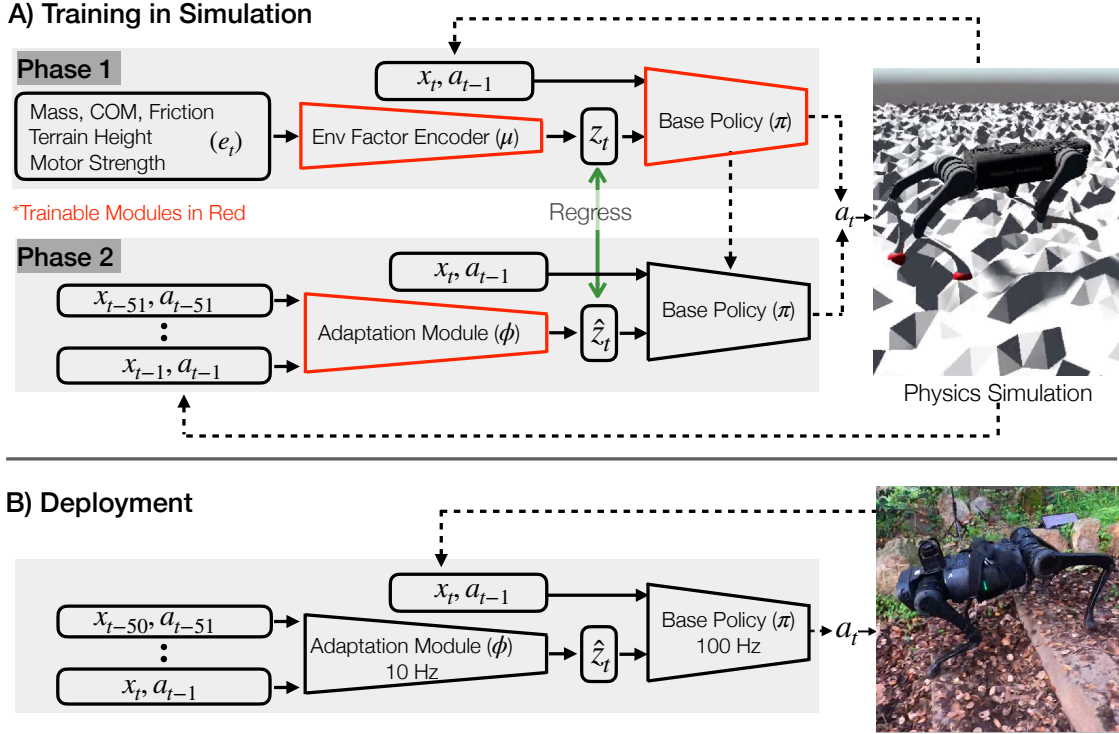


Figure 2.2: RMA consists of two subsystems - the base policy π and the adaptation module ϕ . **Top:** RMA is trained in two phases. In the first phase, the base policy π takes as input the current state x_t , previous action a_{t-1} and the privileged environmental factors e_t which is encoded into the latent extrinsics vector z_t using the environmental factor encoder μ . The base policy is trained in simulation using model-free RL. In the second phase, the adaptation module ϕ is trained to predict the extrinsics \hat{z}_t from the history of state and actions via supervised learning with on-policy data. **Bottom:** At deployment, the adaptation module ϕ generates the extrinsics \hat{z}_t at 10Hz, and the base policy generates the desired joint positions at 100Hz which are converted to torques using A1’s PD controller. Since the adaptation module runs at a lower frequency, the base policy consumes the most recent extrinsics vector \hat{z}_t predicted by the adaptation module to predict a_t . This asynchronous design was critical for seamless deployment on low-cost robots like A1 with limited on-board compute.

jointly trained via RL in simulation.

Unfortunately, this policy cannot be directly deployed because we don't have access to e_t in the real world. What we need to do is to estimate the extrinsics at run time, which is the role of the adaptation module ϕ . The key insight is that when we command a certain movement of the robot joints, the actual movement differs from that in a way that depends on the extrinsics. So instead of using privileged information, we might hope to use the recent history of the agent's state to estimate this extrinsics vector, analogously to the operation of a Kalman filter for state estimation from history of observables. Specifically, the goal of ϕ is to estimate the extrinsics vector z_t from the robot's recent state and action history, without assuming any access to e_t . That is at runtime, but at training time, life is easier. **Since both the state history and the extrinsics vector z_t can be computed in simulation, we can train this module via supervised learning.** At deployment, both these modules work together to perform robust and adaptive locomotion. In our experimental setup with its limited on-board computing, the base policy π runs at 100 Hz, while the adaptation module ϕ is slower and runs at 10Hz. The two run asynchronously in parallel with no central clock to align them. The base policy just uploads the most recent prediction of the extrinsics vector z_t from the adaptation module to predict action a_t .

Our approach is in contrast to previous learning-based work in locomotion that adapt learned policies via inferring the key parameters about the environment from a small dataset collected in every new situation to which the robot is introduced. These could either be physical parameters like friction, etc. [19] or their latent encoding [136]. Unfortunately, as mentioned earlier, collecting such a dataset, when the robot hasn't yet acquired a good policy for walking, could result in falls and damage to the robot. Our approach avoids this because RMA, through the rapid estimation of z_t permits the walking policy to adapt quickly¹ and avoid falls.

Training of a base policy using RL with an extra argument for the environmental parameters has also been pursued in [196, 136]. Our novel aspects are the use of a varied terrain generator and "natural" reward functions motivated by bioenergetics which allows us to learn walking policies without using any reference demonstrations [136]. But the truly novel contribution of this work is the adaptation module, trained in simulation, which makes RMA possible. This, at deployment time, has the flavor of system identification, but it is an on-line version of system identification, based just on the single trajectory that the robot has seen in the past fraction of a second. One might reasonably ask why it should work at all, but we can offer a few speculations:

- System identification is traditionally thought of as an optimization problem. But in many settings researchers have found that given sample (input, output) pairs of optimization problems with their solutions, we could use a neural network to approximate the function mapping the problem to its solution [6, 59]. Effectively that is what ϕ is learning to do.
- We don't need perfect system identification for the approach to work. The vector of extrinsics z_t is a lower-dimensional nonlinear projection of the environmental parameters. This takes care of some identifiability issues where some parameters could covary with identical effects on observables. Secondly, we don't need this vector of extrinsics to be correct in some

¹RMA takes less than 1s, whereas [136] need to collect 4 – 8mins (50 episodes of 5 – 10s) of data.

“ground truth” sense. What matters is that it leads to the “right” action, and the end-to-end training optimizes for that.

- The range of situations seen in training should encompass what the robot will encounter in the real world. We use a fractal terrain generator which accompanied by the randomization of parameters such as mass, friction etc. creates a wide variety of physical contexts in which the walking robot has to react.

The most comparable work in terms of robust performance of RL policies for legged locomotion in the real-world is that of [102] which, unlike our work, relies on hand-coded domain knowledge of predefined trajectory generator [75] and motor models [72]. We evaluated RMA across a wide variety of terrains in the real world (Figure 2.1). The proposed adaptive controller is able to walk on slippery surfaces, uneven ground, deformable surfaces (such as foam, mattress, etc) and on rough terrain in natural environments such as grass, long vegetation, concrete, pebbles, rocky surfaces, sand, etc.

2.1 Rapid Motor Adaptation

We now describe each component of the RMA algorithm introduced earlier and summarized in Figure 2.2. Following sections discuss the base policy, the adaptation module and the deployment on the real-robot in order. We will use the same notation as introduced earlier.

Base Policy

We learn a base policy π which takes as input the current state $x_t \in \mathbb{R}^{30}$, previous action $a_{t-1} \in \mathbb{R}^{12}$ and the extrinsics vector $z_t \in \mathbb{R}^8$ to predict the next action a_t . The predicted action a_t is the desired joint position for the 12 robot joints which is converted to torque using a PD controller. The extrinsics vector z_t is a low dimensional encoding of the environment vector $e_t \in \mathbb{R}^{17}$ generated by μ .

$$z_t = \mu(e_t) \quad (2.1)$$

$$a_t = \pi(x_t, a_{t-1}, z_t) \quad (2.2)$$

We implement μ and π as MLPs (details in Section 2.2). We jointly train the base policy π and the environmental factor encoder μ end to end using model-free reinforcement learning. At time step t , π takes the current state x_t , previous action a_{t-1} and the extrinsics $z_t = \mu(e_t)$, to predict an action a_t . RL maximizes the following expected return of the policy π :

$$J(\pi) = \mathbb{E}_{\tau \sim p(\tau|\pi)} \left[\sum_{t=0}^{T-1} \gamma^t r_t \right],$$

where $\tau = \{(x_0, a_0, r_0), (x_1, a_1, r_1) \dots\}$ is the trajectory of the agent when executing policy π , and $p(\tau|\pi)$ represents the likelihood of the trajectory under π .

Stable Gait through Natural Constraints: Instead of adding artificial simulation noise, we train our agent under the following natural constraints. First, the reward function is motivated from bioenergetic constraints of minimizing work and ground impact [139]. We found these reward functions to be critical for learning realistic gaits in simulation. Second, we train our policies on uneven terrain (Figure 2.2) as a substitute for additional rewards used by [72] for foot clearance and robustness to external push. A walking policy trained under these natural constraints transfers to simple setups in the real world (like concrete or wooden floor) without any modifications. This is in contrast to other sim-to-real work which either calibrates the simulation with the real world [169, 72], or fine-tunes the policy in the real world [136]. The adaptation module then enables it to scale from simple setups to very challenging terrains as shown in Figure 2.1.

RL Rewards: The reward function encourages the agent to move forward with a maximum speed of 0.35 m/s, and penalizes it for jerky and inefficient motions. Let’s denote the linear velocity as v , the orientation as θ and the angular velocity as ω , all in the robot’s base frame. We additionally define the joint angles as q , joint velocities as \dot{q} , joint torques as τ , ground reaction forces at the feet as f , velocity of the feet as v_f and the binary foot contact indicator vector as g . The reward at time t is defined as the sum of the following quantities:

1. Forward: $\min(v_x^t, 0.35)$
2. Lateral Movement and Rotation: $-\|v_y^t\|^2 - \|\omega_{yaw}^t\|^2$
3. Work: $-\|\tau^T \cdot (q^t - q^{t-1})\|$
4. Ground Impact: $-\|f^t - f^{t-1}\|^2$
5. Smoothness: $-\|\tau^t - \tau^{t-1}\|^2$
6. Action Magnitude: $-\|a^t\|^2$
7. Joint Speed: $-\|\dot{q}^t\|^2$
8. Orientation: $-\|\theta_{roll, pitch}^t\|^2$
9. Z Acceleration: $-\|v_z^t\|^2$
10. Foot Slip: $-\|\text{diag}(g^t) \cdot v_f^t\|^2$

The scaling factor of each reward term is 20, 21, 0.002, 0.02, 0.001, 0.07, 0.002, 1.5, 2.0, 0.8 respectively.

Training Curriculum: If we naively train our agent with the above reward function, it learns to stay in place because of the penalty terms on the movement of the joints. To prevent this collapse, we follow the strategy described in [72]. We start the training with very small penalty coefficients, and then gradually increase the strength of these coefficients using a fixed curriculum. We also linearly increase the difficulty of other perturbations such as mass, friction and motor strength as the training progresses. We don’t have any curriculum on the terrains and start the training with randomly sampling the terrain profiles from the same fixed difficulty.

Adaptation Module

The knowledge of privileged environment configuration e_t and its encoded extrinsics vector z_t are not accessible during deployment in the real-world. Hence, we propose to estimate the extrinsics online using the adaptation module ϕ . Instead of e_t , the adaptation module uses the recent history of robot’s states $x_{t-k:t-1}$ and actions $a_{t-k:t-1}$ to generate \hat{z}_t which is an estimate of the true extrinsics vector z_t . In our experiments, we use $k = 50$ which corresponds to 0.5s.

$$\hat{z}_t = \phi(x_{t-k:t-1}, a_{t-k:t-1})$$

Note that instead of predicting e_t , which is the case in typical system identification, we directly estimate the extrinsics z_t that only encodes how the behavior should change to correct for the given environment vector e_t .

To train the adaptation module, we just need the state-action history and the target value of z_t (given by the environmental factor encoder μ). Both of these are available in simulation, and hence, ϕ can be trained via supervised learning to minimize: $\text{MSE}(\hat{z}_t, z_t) = \|\hat{z}_t - z_t\|^2$, where $z_t = \mu(e_t)$. We model ϕ as a 1-D CNN to capture temporal correlations (Section 2.2).

One way to collect the state-action history is to unroll the trained base policy π with the ground truth z_t . However, such a dataset will contain examples of only good trajectories where the robot walks seamlessly. Adaptation module ϕ trained on this data would not be robust to deviations from the expert trajectory, which will happen often during deployment.

We resolve this problem by training ϕ with on-policy data (similar to [151]). We unroll the base policy π with the \hat{z}_t predicted by the randomly initialized policy ϕ . We then use this state action history, paired with the *ground truth* z_t to train ϕ . We iteratively repeat this until convergence. This training procedure ensures that RMA sees enough exploration trajectories during training due to *a*) randomly initialized ϕ , and *b*) imperfect prediction of \hat{z}_t . This adds robustness to the performance of RMA during deployment.

Asynchronous Deployment

We train RMA completely in simulation and then deploy it in the real world without any modification or fine-tuning. The two subsystems of RMA run asynchronously and at substantially different frequencies, and hence, can easily run using little on-board compute. The adaptation policy is slow because it operates on the state-action history of 50 time steps, roughly updating the extrinsic vector \hat{z}_t once every 0.1s (10 Hz). The base policy runs at 100 Hz and uses the most recent \hat{z}_t generated by the adaptation module, along with the current state and the previous action, to predict a_t . This asynchronous execution doesn’t hurt performance in practice because \hat{z}_t changes relatively infrequently in the real world.

Alternately, we could have trained a base policy which directly takes the state and action history as input without decoupling them into the two modules. We found that this (a) leads to unnatural gaits and poor performance in simulation, (b) can only run at 10Hz on the on-board compute, and (c) lacks the asynchronous design which is critical for a seamless deployment of RMA on the real robot without the need for any synchronization or calibration of the two subsystems. This asynchronous

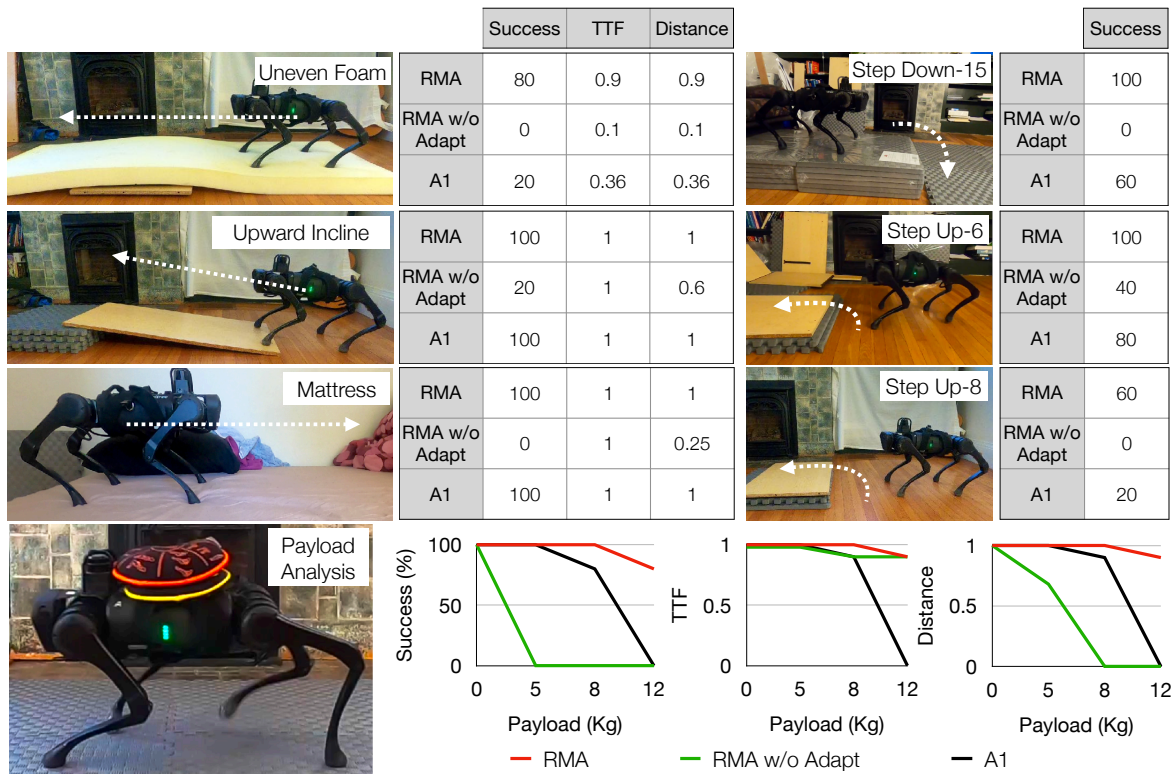


Figure 2.3: We evaluate RMA in several out-of-distribution setups in the real world. We compare RMA to A1’s controller and RMA without the adaptation module. We find that RMA steps down a height of 15cm with 80% success rate and walks over unseen deformable surfaces, such as a memory foam mattress and a slightly uneven foam with 100% success rate. It is also able to successfully climb inclines and steps. A1’s controller fails to walk over uneven foam. At the bottom, we also analyze the payload carrying limits of the three methods. We see that the A1 controller’s performance starts degrading at 8Kg payload capacity. RMA w/o adaptation fails to move for payloads more than 8Kg, but rarely falls. For reference, A1 robot weights 12Kg. Overall, the proposed method consistently dominates the baseline methods. The numbers reported are averaged over 5 trials.

design is fundamentally enabled by the decoupling of the relatively infrequently changing extrinsics vector with the quickly changing robot state.

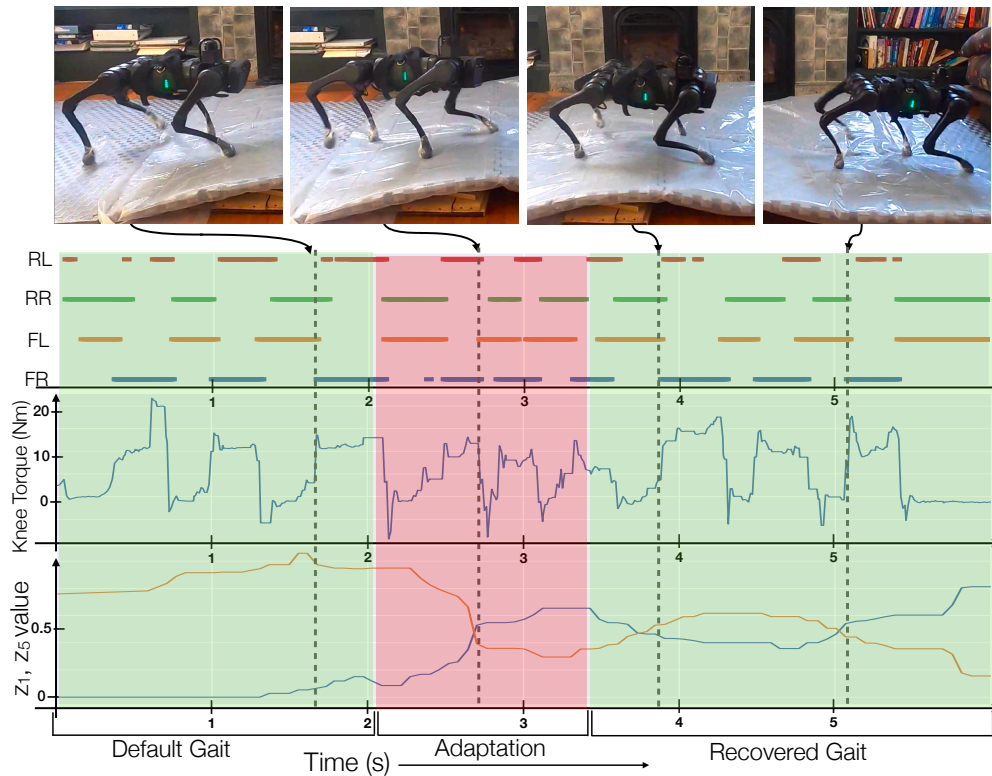


Figure 2.4: We analyze RMA as the robot walks over an oily plastic sheet with additional plastic covering on its feet. We plot the torque of the knee and the gait pattern which indicates the contact of the four feet (F/R denotes Front/Rear and R/L denotes Right/Left). The bottom plot shows median filtered 1st and 5th components of the extrinsics vector \hat{z} predicted by the adaptation module. When the robot enters the slippery patch we see a change in the two components of the extrinsics vector \hat{z} , indicating that the slip event has been detected by the adaptation module. Note that post adaptation, the recovered gait time period is similar to the original, the torque magnitudes have increased and \hat{z} continues to capture the fact that the surface is still slippery. RMA was successful in 90% of the runs over oily patch.

Parameters	Training Range	Testing Range
Friction	[0.05, 4.5]	[0.04, 6.0]
K_p	[50, 60]	[45, 65]
K_d	[0.4, 0.8]	[0.3, 0.9]
Payload (Kg)	[0, 6]	[0, 7]
Center of Mass (cm)	[-0.15, 0.15]	[-0.18, 0.18]
Motor Strength	[0.90, 1.10]	[0.88, 1.22]
Re-sample Probability	0.004	0.01

Table 2.1: Ranges of the environmental parameters.

2.2 Experimental Setup

Environment Details

Hardware Details: We use A1 robot from Unitree for all our real-world experiments. A1 is a relatively low cost medium sized robotic quadruped dog. It has 18 degrees of freedom out of which 12 are actuated (3 motors on each leg) and weighs about 12 kg. To measure the current state of the robot, we use the joint position and velocity from the motor encoders, roll and pitch from the IMU sensor and the binarized foot contact indicators from the foot sensors. The deployed policy uses position control for the joints of the robots. The predicted desired joint positions are converted to torque using a PD controller with fixed gains ($K_p = 55$ and $K_d = 0.8$).

Simulation Setup: We use the RaiSim simulator [71] for rigid-body and contact dynamics simulation. We import the A1 URDF file from Unitree [177] and use the inbuilt fractal terrain generator to generate uneven terrain (fractal octaves = 2, fractal lacunarity = 2.0, fractal gain = 0.25, z-scale = 0.27). Each RL episode lasts for a maximum of 1000 steps, with early termination if the height of the robots drops below 0.28m, magnitude of the body roll exceeds 0.4 radians or the pitch exceeds 0.2 radians. The control frequency of the policy is 100 Hz, and the simulation time step is 0.025s.

State-Action Space: The state is 30 dimensional containing the joint positions (12 values), joint velocities (12 values), roll and pitch of the torso and binary foot contact indicators (4 values). For actions, we use position control for the 12 robot joints. RMA predicts the desired joint angles $a = \hat{\mathbf{q}} \in \mathbb{R}^{12}$, which is converted to torques $\boldsymbol{\tau}$ using a PD controller: $\boldsymbol{\tau} = K_p (\hat{\mathbf{q}} - \mathbf{q}) + K_d (\hat{\dot{\mathbf{q}}} - \dot{\mathbf{q}})$. K_p and K_d are manually-specified gains, and the target joint velocities $\hat{\dot{\mathbf{q}}}$ are set to 0.

Environmental Variations: All environmental variations with their ranges are listed in Table 2.1. Of these, e_t includes mass and its position on the robot (3 dims), motor strength (12 dims), friction (scalar) and local terrain height (scalar), making it a 17-dim vector. Note that although the difficulty of the terrain profile is fixed, the local terrain height changes as the agent moves. We discretize the terrain height under each foot to the first decimal place and then take the maximum among the four feet to get a scalar. This ensures that the controller does not critically depend on a fast and accurate

sensing of the local terrain, and allows the base policy to use it asynchronously at a much lower update frequency during deployment.

Training Details

Base Policy and Environment Factor Encoder Architecture: The base policy is a 3-layer multi-layer perceptron (MLP) which takes in the current state $x_t \in \mathbb{R}^{30}$, previous action $a_{t-1} \in \mathbb{R}^{12}$ and the extrinsics vector $z_t \in \mathbb{R}^8$, and outputs 12-dim target joint angles. The dimension of hidden layers is 128. The environment factor encoder is a 3-layer MLP (256, 128 hidden layer sizes) and encodes $e_t \in \mathbb{R}^{17}$ into $z_t \in \mathbb{R}^8$.

Adaptation Module Architecture: The adaptation module first embeds the recent states and actions into 32-dim representations using a 2-layer MLP. Then, a 3-layer 1-D CNN convolves the representations across the time dimension to capture temporal correlations in the input. The input channel number, output channel number, kernel size, and stride of each layer are [32, 32, 8, 4], [32, 32, 5, 1], [32, 32, 5, 1]. The flattened CNN output is linearly projected to estimate \hat{z}_t .

Learning Base Policy and Environmental Factor Encoder Network: We jointly train the base policy and the environment encoder network using PPO [158] for 15,000 iterations each of which uses batch size of 80,000 split into 4 mini-batches. The learning rate is set to $5e-4$. The coefficient of the reward terms are provided in Section 2.1. Training takes roughly 24 hours on an ordinary desktop machine, with 1 GPU for policy training. In this duration, it simulates 1.2 billion steps.

Learning Adaptation Module: We train the adaptation module using supervised learning with on-policy data. We use Adam optimizer [87] to minimize MSE loss. We run the optimization process for 1000 iterations with a learning rate of $5e-4$ each of which uses a batch size of 80,000 split up into 4 mini-batches. It takes 3 hours to train this on an ordinary desktop machine, with 1 GPU for training the policy. In this duration, it simulates 80 million steps.

2.3 Results and Analysis

We compare the performance of RMA with several baselines in simulation (Table 2.2). We additionally compare to the manufacturer’s controller, which ships with A1, in the real world indoor setups (Figure 2.3) and run RMA in the wild in a very diverse set of terrains (Figure 2.1). Videos at <https://ashish-kmr.github.io/rma-legged-robots/>

Baselines: We compare to the following baselines:

1. A1 Controller: The default robot manufacturer’s controller which uses a force-based control scheme with MPC.
2. Robustness through Domain Randomization (Robust): The base policy is trained without z_t to be robust to the variations in the training range [171, 137].

	Success (%)	TTF	Reward	Distance (m)	Samples	Torque	Smoothness	Ground Impact
Robust [171]	62.4	0.80	4.62	1.13	0	527.59	122.50	4.20
SysID [196]	56.5	0.74	4.82	1.17	0	565.85	149.75	4.03
AWR [136]	41.7	0.65	4.17	0.95	40k	599.71	162.60	4.02
RMA w/o Adapt	52.1	0.75	4.72	1.15	0	524.18	106.25	4.55
RMA	73.5	0.85	5.22	1.34	0	500.00	92.85	4.27
Expert	76.2	0.86	5.23	1.35	0	485.07	85.56	3.90

Table 2.2: **Simulation Testing Results:** We compare the performance of our method to baseline methods in simulation. Our train and test settings are listed in Table 2.1. We resample the environment parameters within an episode with a re-sampling probability of 0.01 per step during testing. Baselines and metrics are defined in Section 2.3. The numbers reported are averaged over 3 randomly initialized policies and 1000 episodes per random initialization. RMA beats the performance of all the baselines, with only a slight degradation in performance compared to the Expert.

3. Expert Adaptation Policy (Expert): In simulation, we can use the true value of the extrinsics vector z_t . This is an upper bound to the performance of RMA.
4. RMA w/o Adaptation: We can also evaluate the performance of the base policy without the adaptation module to ablate the importance of the adaptation module.
5. System Identification [196]: Instead of predicting \hat{z}_t , we directly predict the system parameters \hat{e}_t .
6. Advantage Weighted Regression for Domain Adaptation (AWR) [136]: Optimize \hat{z}_t offline using AWR by using real-world rollouts of the policy in the testing environment.

Learning baselines were trained with the same architecture, reward function and other hyper-parameters.

Metrics: We compare the performance of RMA against baselines using the following metrics: (1) time-to-fall divided by maximum episode length to get a normalized value between 0 – 1 (TTF); (2) average forward reward, (3) success rate, (4) distance covered, (5) exploration samples needed for adaptation, (6) torque applied, (7) smoothness which is derivative of torque and (7) ground impact (details in the supplementary).

Indoor Experiments

In the real world, we compare RMA with A1’s controller and with RMA without the adaptation module (Figure 2.3). We limit comparison to these two baselines to avoid damage to the robot

hardware. We run 5 trials for each method and report the success rate, time to fall (TTF), and distance covered. Note that if a method drastically failed at a task, we only run two trials and then report a failure. This is done to minimize damage to the robot hardware. We have the following indoor setups:

- **n-kg Payload:** Walk 300cm with n-kg payload on top.
- **StepUp-n:** Step up on an n-cm high step.
- **Uneven Foam:** Walk 180cm on a center elevated foam.
- **Mattress:** Walk 60cm on a memory foam mattress.
- **StepDown-n:** Step down an n-cm high step.
- **Incline:** Walk up on a 6-degrees incline.
- **Oily Surface:** Cross through an an oily patch.

Each trial of **StepUp-n** and **StepDown-n** is terminated after a success or a failure. Thus, we only report the success rate for these tasks because other metrics are meaningless.

We observe that RMA achieves a high success rate in all these setups, beating the performance of A1's controller by a large margin in some cases. We find that turning off the adaptation module substantially degrades performance, implying that the adaptation module is critical to solve these tasks. A1's controller struggled with uneven foam and with a large step down and step up. The controller was destabilized by unstable footholds in most of its failures. In the payload analysis, the A1's controller was able to handle higher than the advertised payload (5Kg), but starts sagging, and eventually falls as the payload increases. In contrast, RMA maintains the height and is able to carry up to 12Kg (100% of body weight) with a high success rate. RMA w/o adaptation mostly doesn't fall, but also doesn't move forward. We also evaluated RMA in a more challenging task of crossing an oily path with plastic wrapped feet. The robot successfully walks across the oily patch. Interestingly, RMA w/o adaptation was able to walk successfully on wooden floor without any fine-tuning or simulation calibration. This is in contrast to existing methods which calibrate the simulation [169, 72] or fine-tune their policy at test time [136] even for flat and static environments.

Outdoor Experiments

We demonstrate the performance of RMA on several challenging outdoor environments as shown in Figure 2.1. The robot is successfully able to walk on sand, mud and dirt without a single failure in all our trials. These terrains make locomotion difficult due to sinking and sticking feet, which requires the robot to change the footholds dynamically to ensure stability. RMA had a 100% success rate for walking on tall vegetation or crossing a bush. Such terrains obstruct the feet of the robot, making it periodically unstable as it walks. To successfully walk in these setups, the robot has to stabilize against foot entanglements, and power through some of these obstructions aggressively. We also evaluate our robot on walking down some stairs found on a hiking trail. The robot was

successful in 70% of the trials, which is still remarkable given that the robot never sees a staircase during training. And lastly, we test the robot over construction debris, where it was successful 100% of the times when walking downhill over a mud pile and 80% of the times when walking across a cement pile and a pile of pebbles. The cement pile and pebbles were itself on a ground which was steeply sloping sideways, making it very challenging for the robot to go across the pile.

Simulation Results

We compare the performance of our method to baseline methods in simulation (Table 2.2). We sample our training and testing parameters according to Table 2.1, and resample them within an episode with a resampling probability of 0.004 and 0.01 per step respectively for training and testing. The numbers reported are averaged over 3 randomly initialized policies and 1000 episodes per random initialization. RMA performs the best with only a slight degradation compared to Expert’s performance. The constantly changing environment leads to poor performance of AWR which is very slow to adapt. Since the Robust baseline is agnostic to extrinsics, it learns a very conservative policy which loses on performance. Note that the low performance of SysID implies that explicitly estimating e_t is difficult and unnecessary to achieve superior performance. We also compare to RMA w/o adaptation, which shows a significant performance drop without the adaption module.

Adaptation Analysis

We analyze the gait patterns, torque profiles and the estimated extrinsics vector \hat{z}_t for adaptation over slippery surface (Figure 2.4). We pour oil on the plastic surface on the ground and additionally cover the feet of the robot in plastic. The robot then tries to cross the slippery patch and is able to successfully adapt to it. We found that RMA was successful in 90% of the runs over oily patch. For one such trial, we plot the torque profile of the knee, the gait pattern, and median filtered 1st and 5th components of the extrinsics vector \hat{z}_t in Figure 2.4. When the robot first starts slipping somewhere around 2s, the slip disturbs the regular motion of the robot, after which it enters the adaptation phase. This is noticeable in the plotted components of the extrinsics vector which change in response to the slip. This detected slip enables the robot to recover and continue walking over the slippery patch. Note that although post adaptation, the torque stabilizes to a slightly higher magnitude and the gait time period is roughly recovered, the extrinsics vector does not recover and continues to capture the fact that the surface is slippery. See supplementary more such analysis.

2.4 Related Work

Conventionally, legged locomotion has been approached by using control-based methods [119, 146, 55, 192, 166, 80, 85, 10, 73, 13]. MIT Cheetah 3 [17] can achieve high speed and jump over obstacles by using regularized model predictive control (MPC) and simplified dynamics [38]. The ANYmal robot [69] locomotes by optimizing a parameterized controller and planning based on an inverted pendulum model [54]. However, these methods require accurate modeling of the

real-world dynamics, in-depth prior knowledge of the robots, and manual tuning of gaits and behaviors. Optimizing controllers, combined with MPC, can mitigate some of the problems [89, 20, 27], however they still require significant task-specific feature engineering [37, 54, 11].

Learning for Legged Locomotion Some of the earliest attempts to incorporate learning into locomotion can be dated back to DARPA Learning Locomotion Program [203, 204, 149, 202, 81]. More recently, deep reinforcement learning (RL) offered an alternative to alleviate the reliance on human expertise and has shown good results in simulation [158, 106, 120, 51]. However, such policies are difficult to transfer to the real world [92, 127, 18]. One approach is to directly train in the real world [62, 190]. However, such policies are limited to very simple setups, and scaling to complex setups requires unsafe exploration and a large number of samples.

Sim-to-Real Reinforcement Learning To achieve complex walking behaviours in the real world using RL, several methods try to bridge the Sim-to-Real gap. Domain randomization is a class of methods in which the policy is trained with a wide range of environment parameters and sensor noises to learn behaviours which are robust in this range [169, 171, 137, 183, 125]. However, domain randomization trades optimality for robustness leading to an over conservative policy [108].

Alternately, the Sim-to-Real gap can also be reduced by making the simulation more accurate [72, 169, 65]. [169] improve the motor models by fitting a piece-wise linear function to data from the actual motors [169]. [72], instead, use a neural network to parameterize the actuator model [72, 102]. However, these approaches require initial data collection from the robot to fit the motor model, and would require this to be done for every new setup.

System Identification and Adaptation Instead of being agnostic to physics parameters, the policy can condition on these parameters via online system identification. During deployment in the real world, physics parameters can either be inferred through a module that is trained in simulation [196], or be directly optimized for high returns by using evolutionary algorithms [193]. Predicting the exact system parameters is often unnecessary and difficult, leading to poor performance in practice. Instead, a low dimensional latent embedding can be used [136, 201]. At test time, this latent can be optimized using real-world rollouts by using policy gradient methods [136], Bayesian optimization [197], or random search [195]. Another approach is to use meta learning to learn an initialization of policy network for fast online adaptation [47]. Although they have been demonstrated on real robots [165, 28], they still require multiple real-world rollouts to adapt.

2.5 Summary

We presented the RMA algorithm for real-time adaptation of a legged robot walking in a variety of terrains. Two notable aspects of the proposed design is that a) we do not use any predefined gaits or reference motions to train, and b) the system is completely blind. In the next chapter, we will delve deeper into the first aspect of letting gaits emerge from energy minimization instead of using predefined gaits, and in the subsequent chapter, extend the RMA algorithm to enable visual walking.

Chapter 3

Emergent walking gaits via Energy Minimization

In our approach to legged walking introduced in the previous chapter, we did not rely on any predefined gaits, or heuristic reference motions. This is in contrast to most common approaches to understanding mammalian locomotion, which study it in terms of a set of discrete gaits. Seminal works include [124]’s motion video of a galloping horse and [66]’s study of different gaits observed at different speeds. This has inspired research on replicating these gaits in robots by pre-programming them to achieve effective locomotion. This approach of pre-programming fixed gait patterns has lead to immense progress in robotic locomotion [147, 113, 39, 38].

However, pre-programmed gaits limit the ability of legged systems to perform general locomotion in diverse terrains and at different speeds. Studies of motor control in both animals [] and human infants [1] in the last few decades have shown that general locomotion in natural settings is irregular and does not fit predefined fixed gaits. It is better defined in terms of bouts of steps than periodic cycles [30, 93]. That being said, we do see consistent regular patterns in ideal conditions when an animal is moving in a straight line over flat terrain at different speeds, e.g., walk, trot, canter, gallop, etc. [66, 77, 2, 67] and these patterns are shared across species of diverse animals [8]. How do we reconcile these seemingly contradictory arguments about unstructured locomotion patterns on complex terrains with that of regular patterns seen in flat terrains under the same rubric?

One way to make sense of these differences is to see them from the lens of minimizing energy consumption. Locomotion consumes a significant fraction of an animal’s metabolic energy budget which suggests that there would be evolutionary selection pressure for locomotion strategies that minimize energy consumption. Indeed, this has been a focus of several studies in biomechanics and energetics. [67] used energy minimization to explain why animals switch from one gait to another as their speed increases. Similar studies explain gaits across different animals including humans [15, 140]. However, most of the biomechanics studies are limited to the straight-line motion of animals in ideal flat terrains. Furthermore, these studies take gaits as given and offer an explanation for them

This chapter is based on joint work with Zipeng Fu, Jitendra Malik and Deepak Pathak [50]

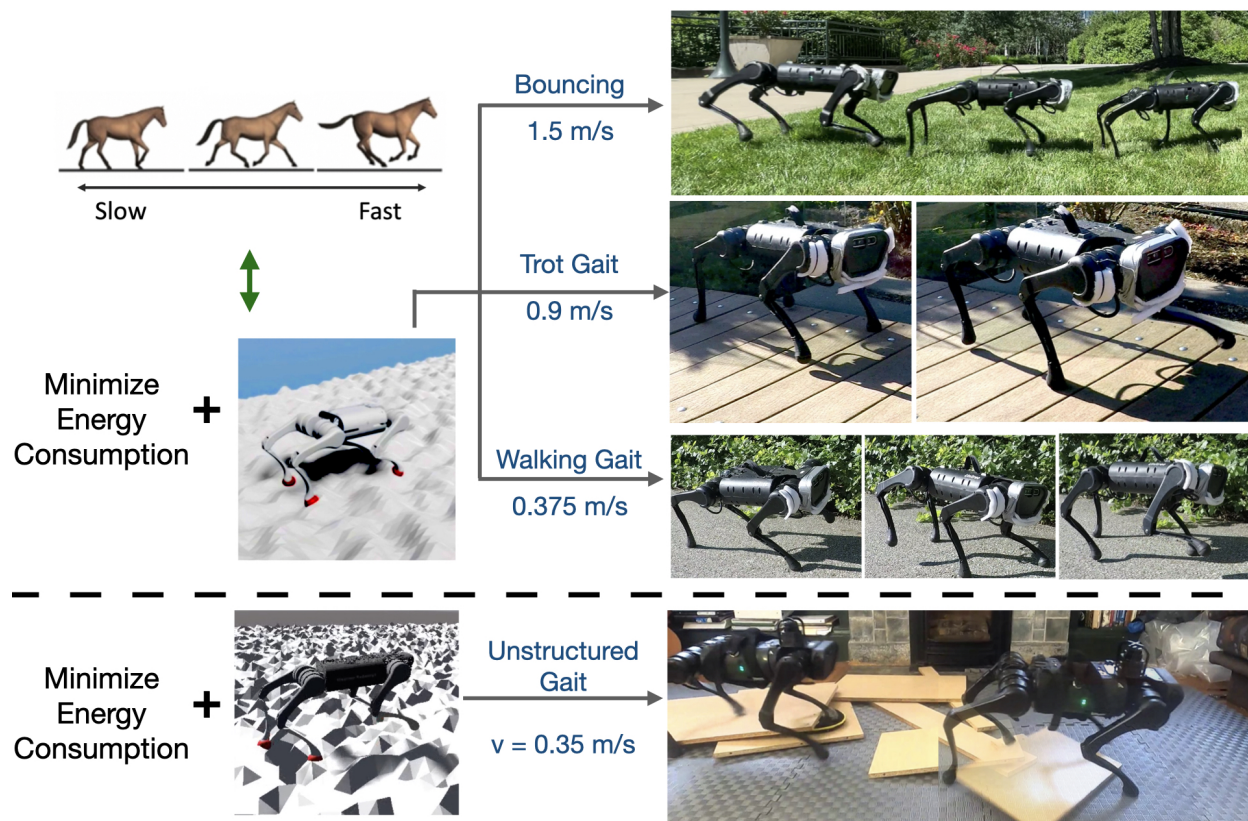


Figure 3.1: We demonstrate via analysis-by-synthesis approach that learning to move forward by minimizing energy consumption plays a key role in the emergence of natural locomotion patterns in quadruped robots. We do not pre-program any primitives for leg motions and the policy directly output desired joint angles. **Top:** Bio-energetics driven learning on flat terrain leads to different gaits namely walk, trot and bounce (similar to gallop) as the speed increases. Our high-speed bounce gait displays an emergent *flight* phase despite low energy usage. This corresponds to the nearest animals (sheep/horse) with similar Froude numbers. **Bottom:** The same pipeline on diverse uneven terrains leads to unstructured gait as is true in general animal locomotion. [website](#).

from an energy perspective without any statement about their synthesis.

In this work, we take an analysis-by-synthesis approach to show how energy minimization leads to the emergence of structured locomotion gait patterns in flat terrains as well as unstructured gaits in complex terrains. We use energetics to design an end-to-end learning framework and display the resulting gait patterns in a real quadruped robot. We employ model-free reinforcement learning as the optimizer to learn controllers to make the quadruped move forward while simultaneously minimizing the bio-energetics constraints of minimizing work [139]. We train our policies on simple fractal terrains with varying frequency of terrain heights instead of perfectly flat terrain. This mimics the real world more closely and achieves efficient and robust gaits without the need for artificial rewards for foot clearance or periodic external push during training. At low frequency, it resembles the simple flat terrain settings and at high frequency, it simulates complex terrains. In

addition, we present a learning pipeline that enables smooth gait transition given changing target speeds by combining RL and knowledge distillation from trained expert gait policies. We then transfer the policies onto the real robot by performing online adaptation [96].

We empirically demonstrate that minimizing energy consumption plays a key role in the emergence of natural gait patterns in a quadruped legged robot. Our quadruped robot is closest to sheep and horse in terms of physicality as determined by the Froude number [175] which is a scalar metric characterizing gaits of quadrupeds (discussed more in Section 3.3). Therefore, energetics-based training leads to similar three gaits in our robot as found in horses and sheep, namely, walk, trot and gallop as the speed is gradually increased in flat terrains Figure 3.1. Interestingly, we show that the gait selection is inherently linked with the speed of the robot. For instance, trotting is only energy efficient around the medium speed of 0.9m/s. If we increase or decrease the speed of the robot, it takes more energy for trot than to gallop (similar to bouncing gait) or walk as shown in Figure 3.2. Finally, when the same pipeline is run in uneven complex terrains, it leads to unstructured gaits as consistent with animal locomotion in usual diverse conditions [93, 30].

Prior works have used energy penalty to obtain energy-efficient gaits [169, 165] but have not shown the correspondence to different gaits at different speeds. Furthermore, out of these works, the ones which deploy these gaits on a real robot either use a predefined hand-coded gait library, use reference motions [169], or only learn high-level contact sequences for a predefined swing leg motion [191]. These approaches have some fundamental limitations in contrast to what we propose. Firstly, the use of predefined foot motions never optimize for efficient swing leg motions, and using reference trajectories gives gaits that look realistic, but are not necessarily energy-efficient for the specific robot we have. Second, learning controllers end to end without the use of any priors (reference motions or predefined motions) allows for the possibility of learning behaviors that don't follow any predefined gaits. We show the robustness of this complex terrain policy by successfully deploying it in complex terrains in the real world.

3.1 Method

Claim: Energy Minimization leads to Emergence of Natural Gaits

We would like to validate that indeed certain gaits will emerge at certain speeds if we minimize the energy consumption. To do so, on the right, we plot the energy consumption vs speed of the A1 robot in simulation for walk, trot and bounce (modification of gallop/canter) gaits. We use [38] to achieve the different gaits by hard-coding the contact sequences for walk, trot and bounce. This plot is qualitatively similar to the plot in [67], which shows the energy consumption of different horse gaits across different speeds. We additionally plot the energy consumption of the three policies we get at different speeds (0.375 m/s, 0.9 m/s, 1.5 m/s). This plot shows two things. First, it is observed that certain gaits are only efficient at certain speeds and using them for other speeds is suboptimal. For instance, walk is optimal at low speeds with transition to trot and bounce. Secondly, we observe that our learning framework converges to the optimal gait at each of the speeds, and beats the efficiency of MPC gaits from [38], which does not optimize the swing foot trajectory for energy. Details of MPC method are in the supplementary.

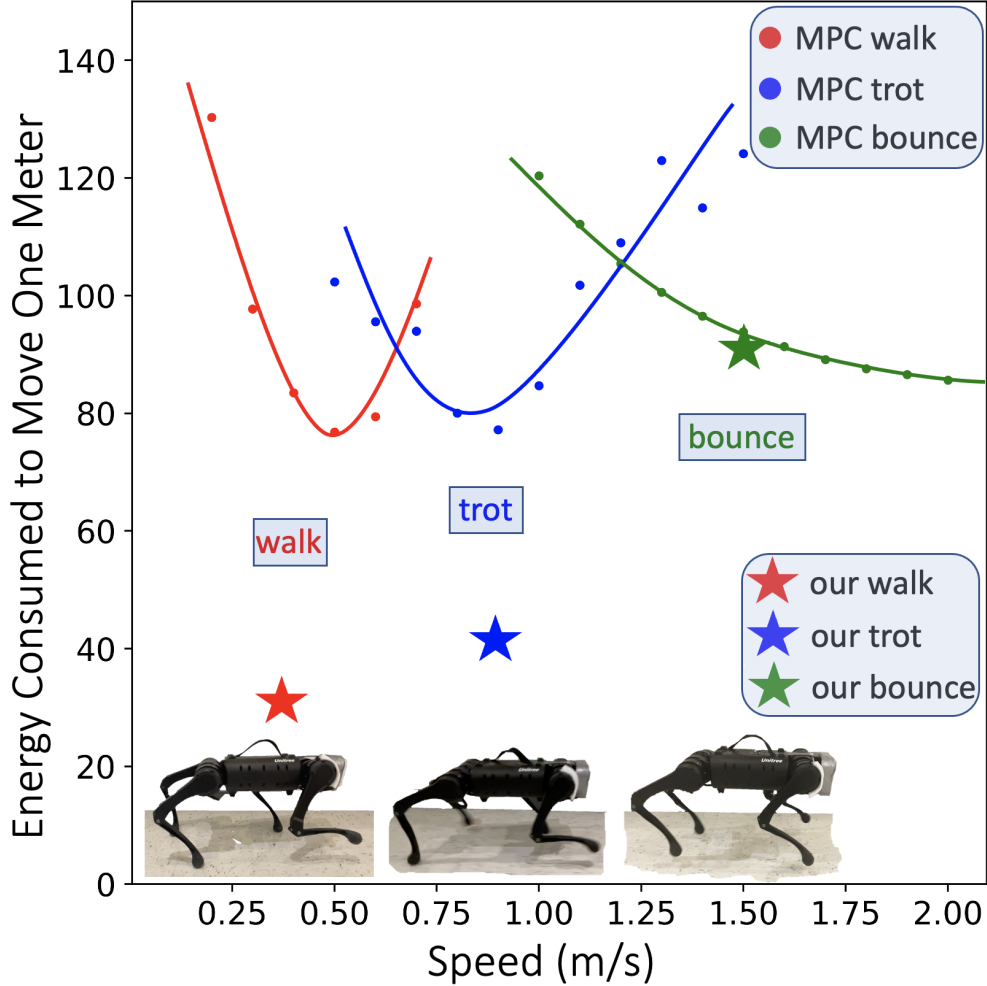


Figure 3.2: Energy consumed in moving 1m distance using our method and MPC shows why certain gaits are stable at certain speeds. Videos on the website.

Learning Locomotion Policy

We learn the locomotion policy π which takes as input the current state $x_t \in \mathbb{R}^{30}$, previous action $a_{t-1} \in \mathbb{R}^{12}$ to predict the next action a_t (Equation 3.1). The predicted action a_t is the desired joint position for the 12 robot joints which is converted to torque using a PD controller.

$$a_t = \pi(x_t, a_{t-1}) \quad (3.1)$$

We implement π as MLPs (details in Section B.1) and train the base policy π end to end using model-free reinforcement learning. At time step t , π takes the current state x_t , previous action a_{t-1} to predict an action a_t . RL maximizes the following expected return of the policy π :

$$J(\pi) = \mathbb{E}_{\tau \sim p(\tau|\pi)} \left[\sum_{t=0}^{T-1} \gamma^t r_t \right], \quad (3.2)$$

where $\tau = \{(x_0, a_0, r_0), (x_1, a_1, r_1) \dots\}$ is the trajectory of the agent when executing policy π , and $p(\tau|\pi)$ represents the likelihood of the trajectory under π .

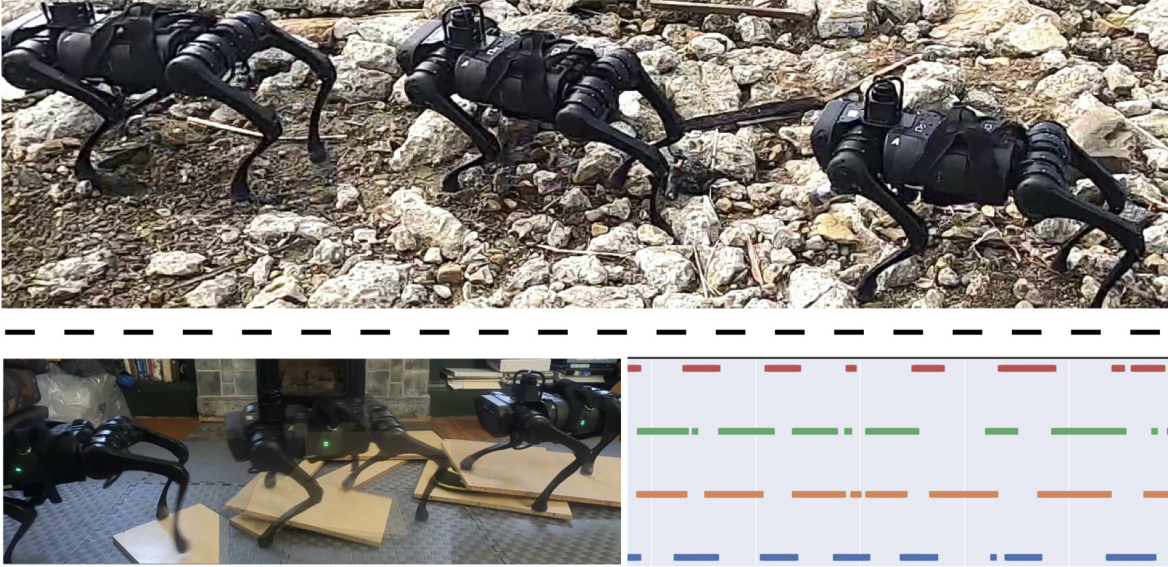


Figure 3.3: Complex real-world behaviors at the low speed in 2 settings. TOP: key frames on rocky terrain. BOTTOM: Foot contact plots and key frames on unstable moving planks. Videos on the website.

Stable Gait through Natural Constraints

Instead of adding artificial simulation noise, we train our agent under the following natural constraints. First, the reward function is motivated from bio-energetic constraints of minimizing work [139]. We found these reward functions to be critical for learning realistic gaits in simulation. Second, we train our policies on uneven terrain (Figure 3.1) as a substitute for additional rewards used by [72] for foot clearance and robustness to external push. We see that these constraints are enough to achieve all the gaits and results we demonstrate in this chapter.

Energy Consumption-Based Reward

Let’s denote the linear velocity as v and the angular velocity as ω , both in the robot’s base frame. We additionally define joint torques as τ and joint velocities as \dot{q} . We define our reward as sum of the following three terms:

$$r = r_{\text{forward}} + \alpha_1 * r_{\text{energy}} + r_{\text{alive}} \quad (3.3)$$

where,

$$r_{\text{forward}} = -\alpha_2 * |v_x - v_x^{\text{target}}| - |v_y|^2 - |\omega_{\text{yaw}}|^2 \quad (3.4)$$

$$r_{\text{energy}} = -\tau^T \dot{q}, \quad (3.5)$$

r_{forward} rewards the agent for walking straight at the specified speed, r_{energy} penalizes energy consumption and r_{alive} is the survival bonus.

We use simple rules to set the hyper-parameters. At a given target linear speed v_x^{target} , the survival bonus c is set to $20 * v_x^{\text{target}}$. We set $\alpha_1 = 0.04$, and $\alpha_2 = 20$ across all settings.

Notice that the actual energy consumption on the robot depending on the low-level hardware design is not directly measurable. We estimate the unit energy consumption per time step by

summing the instantaneous power of the 12 motors by multiplying the torque and the joint velocity at each motor. Also notice that this reward is much more concise than those used in prior works. We discuss the role of minimizing energy consumption in Section 3.3.

Sim to Real Transfer

For simulation to real transfer, we use RMA [96] which consists of the an adaptation module on top of the base policy. During deployment, the adaptation module uses the state history to estimate the vector of extrinsics (which contains environment information) online. This adaptation module is trained in simulation using supervised learning after the base policy is learned using RL with the true extrinsics parameters. The environment perturbations we train on is listed in Table 3.1.

3.2 Experimental Setup and Training Details

Hardware and Simulation: We use Unitree’s A1 robot as our hardware platform [177] and use its A1 URDF [177] to simulate the A1 robot in RaiSim simulator [71]. The full details of terrain and environment setup are provided in supplementary due to space constraints.

State-Action Space: The state is 30 dimensional containing the joint positions (12 values), joint velocities (12 values), roll and pitch of the torso and binary foot contact indicators (4 values). The action space is 12 dimensional corresponding to the target joint position for the 12 robot joints. The predicted target joint angles $a = \hat{q} \in \mathbb{R}^{12}$ is converted to torques τ using a PD controller with target joint velocities set to 0.

Environmental Variations: All environmental variations with their ranges are listed in Table 3.1. Policies with simple terrain and mild environment variations achieve walking, trotting and bouncing gaits. To deploy our robot in the complex terrain, we use aggressive environment randomization with sharper fractal terrains which gives us the unstructured walking terrain.

Parameters	Normal Perturbation	Aggressive Perturbation
Friction Coeff.	[0.6, 1.2]	[0.05, 4.5]
K_p	[50, 60]	[50, 60]
K_d	[0.4, 0.8]	[0.4, 0.8]
Payload (kg)	[0.0, 0.5]	[0.0, 6.0]
Center of Mass (m)	[-0.15, 0.15]	[-0.15, 0.15]
Motor Strength	[0.95, 1.05]	[0.90, 1.10]
Re-sample Prob.	0.02	0.02

Policy Learning: The policy is a multi-layer perceptron with 3 layers which takes in the current state $x_t \in \mathbb{R}^{30}$, previous action $a_{t-1} \in \mathbb{R}^{12}$ and outputs 12-dim target joint angles. The hidden layers have 128 units. We

train the policy and the environment encoder network using PPO [158]. The training runs for 15,000 iterations with a batch size of 100,000 split into 4 mini-batches and learning rate = $5e-4$.

Table 3.1: Ranges of the environment parameters in simulation. Normal perturbation is used for regular gait emergence. Aggressive perturbation is used for unstructured gait emergence.

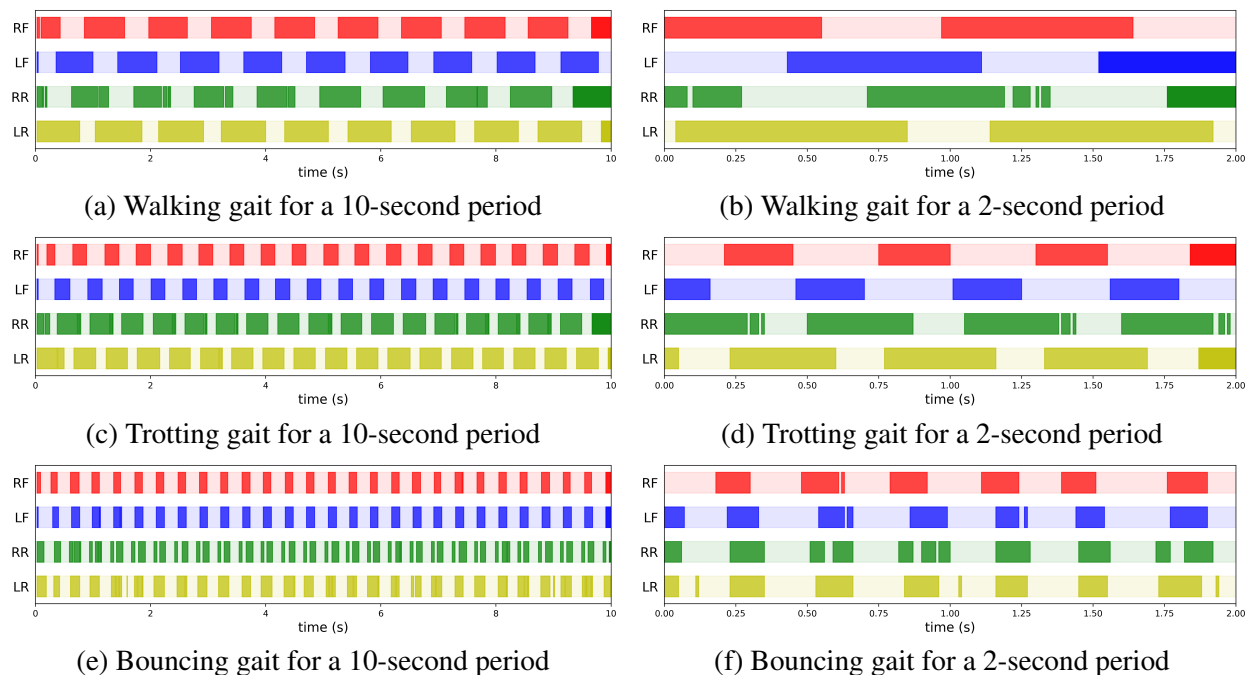


Figure 3.4: Foot contact plots for walking, trotting and bouncing gait of A1 robot in the real world. Bold color means the corresponding foot is in contact with the ground and the light color means the foot is in the air. (RF: Right-Front foot, LF: Left-Front foot, RR: Right-Rear foot, LR: Left-Rear foot)

We train for a total of 1.5 billion samples which takes roughly 24 hours on a desktop machine with 1 GPU.

3.3 Results and Analysis

Emergence of Locomotion Patterns in Complex Terrain

We analyze the performance of the walking policy on complex terrain with aggressive perturbations in the real world (videos on website). We show two uneven terrain deployments of our method in Figure 3.3. During the deployment of the robot in the rocky terrain, we see that the robot roughly follows the walking gait which repeatedly gets obstructed by the rocks and has to step over rocks of different heights, prematurely terminating the swing motion of the leg. Consequently, the gait we observe is an unstructured walking gait. We further analyse this in another deployment on unstable planks which move as the robot tries to cross it (see video). The foot contact plot for this setup is not periodic and changes depending on the complexity of the terrain the policy needs to react to. In the video, this can be seen when the front right foot of the robot steps on an unstable plank which moves as a result, but the robot maintains stability and successfully crosses it.

Method	Walk (0.375 m/s)		Trot (0.9 m/s)		Bounce (1.5 m/s)	
	Speed (m/s)	Energy	Speed (m/s)	Energy	Speed (m/s)	Energy
Ours	0.353	30.7	0.970	41.2	1.439	90.7
MPC [38]	0.360	87.0	0.953	77.2	1.482	93.9

Table 3.2: We show the actual speeds and energy consumed to move one meter of the our method and convex MPC, which does not directly optimize for energy efficiency, for walking gait at target speed 0.375 m/s, trotting gait at target speed 0.9 m/s, and bouncing gait at target speed 1.5 m/s in simulation. Our method shows the advantages in energy consumption in all gaits.

Emergence of Walking, Trotting and Bouncing

Simulation Gaits: We train 3 separate policies for 3 target speeds - 0.375 m/s (low), 0.9 m/s (median) and 1.5 m/s (high) in simulation. We use the same hyper-parameters across all target speeds. We observe that the walking gait emerges at target speed 0.375 m/s, trotting emerges at 0.9 m/s and hopping emerges at 1.5 m/s in simulation. Note that we use RMA [96] (Section 3.1) to transfer our policies to the real world without any fine-tuning. We observe the same foot contact sequence in the real world as in simulation. In Table 3.2, we compare the performance of our policies with MPC [38] and observe that our walking gait can achieve accurate speed tracking while being 50% more energy efficient than the MPC baseline.

Real-World Behaviors: At the low speed (0.375 m/s), the robot demonstrates a *quarter-off* walking gait with the following foot contact sequence: Right-Front (RF), Left-Rear (LR), Left-Front (LF) and Right-Rear (RR). At the median speed (0.9 m/s), two beat trot gait emerges where diagonal legs (RF & LR or LF & RR) are synchronized and hit the ground at the same time. At the high speed (1.5 m/s), the policy converges to bouncing with approximately half the gait cycle as flight phase. Key frames of the 3 gaits are shown in Supplementary and the foot contact plots in Figure 3.4. We use the foot contact sensor reading from the A1 robot to generate these plots.

We also measured the actual speed of the robot in the real world by taking the average of 3 trials per target speed. In each trial, we log the total distance divided by total time taken to traverse. The average real-world speeds are 0.396 m/s at target speed 0.375 m/s, 0.914 m/s at target speed 0.9 m/s, and 1.714 m/s at target speed 1.5 m/s. The average real-world speeds closely match the target speeds in all settings. The standard deviation of speeds across 3 trials is negligible.

We test the robustness of the walking policies by adding a 1 Kg payload on the robot by strapping two bottles with 500 ml water. In Figure 3.5, we show 4 key frames of trotting gait. Note that the 1 kg payload is outside the range of training perturbations (Table 3.1), but our policies with the Sim-to-Real Adaptation in Section 3.1 learn to compensate the increase in weight by implicitly predicting changes in dynamics via latent extrinsics. We didn't test bouncing with payload, because the extra 1 kg prevents the human operator from swiftly preempting the robot at high speed by lifting.

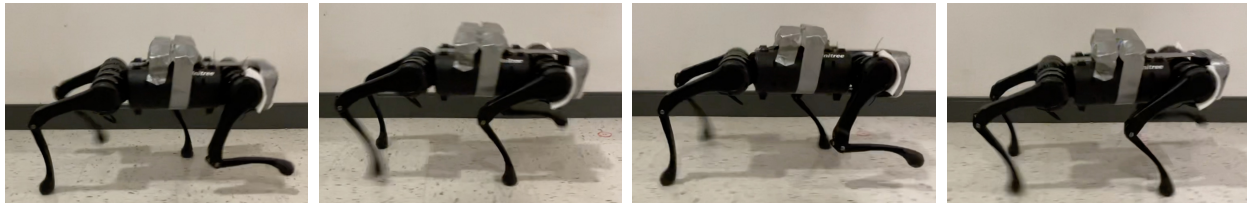


Figure 3.5: Trotting with 1 kg payload (two bottles of 500 ml water are strapped onto the robot). The 1 kg payload is out of the normal perturbation of environment parameters used in simulation training shown in Table 3.1. We show qualitative and gait patterns for other two gaits in the appendix. We find that the gait patterns are robust and remain the same even in the presence of disturbances.

Froude Number Analysis: In Biomechanics, Froude number F is a scalar metric characterizing gaits of quadrupeds and bipeds [175]. Concretely, $F = v^2/gh$, where v is the linear speed, g is the gravitational acceleration, and h is the height of the hip joint from the ground. Existing research shows that animals with a similar morphology but with different sizes tend to use the same gait when walking with equal Froude numbers [7, 77]. We calculate the Froude numbers of our robot in the real world for the low, median and high speeds, and compare them with the Froude numbers of quadrupeds in nature – dogs, horses and sheep [67, 77]. We compare the Froude number of animals at the gait transition boundaries and observe that our robot is closest to the sheep in Table 3.3.

Gaits (Froude #)	A1	Sheep [77]	Horses [67]	Dogs [77]
Walk	0.059	0.05	0.07	0.10
Trot	0.316	0.30	0.61	0.50
Bounce	1.110	1.10	1.70	1.73

Table 3.3: We compare the Froude numbers of our emergent gaits with other quadruped animals at their gait transition boundaries. Our policies are closest to sheep in all gaits and to horse in walking gait. Bounce includes canter and gallop.

Gait Transition via a Velocity-Conditioned Policy

We have shown walking, trotting and bouncing emerged at three different target speeds. To demonstrate foot patterns at a continuous range of velocities, we propose a learning scheme (Figure 3.6) for a velocity-conditioned policy that enables smooth gait transition when the command velocity changes.

Naive Multi-Task Training Fails: We first tried the naive approach to train the velocity-conditioned policy in a multi-task fashion by randomly sampling desired velocities and using the corresponding velocity-conditioned reward (Section 3.1). However, this did not work and the resulting emerged gaits collapse to only two modes: walking and trotting, but trotting at high speeds is not energy-efficient (Figure 3.7). We believe the reason for failure is difficulty in optimization as the robot is now tasked not only to learn to move forward but also do it by learning different gaits which causes it to collapse.

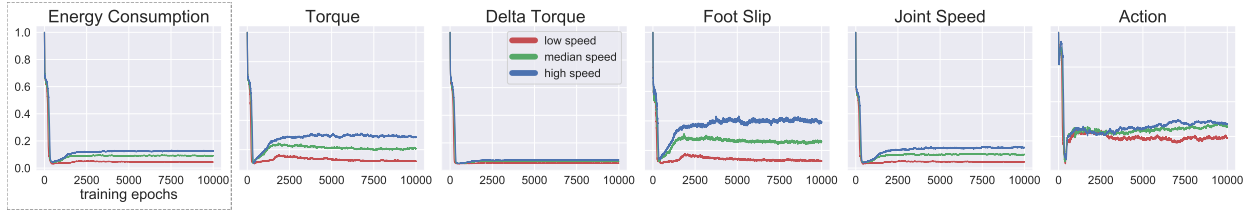


Figure 3.7: Energy consumption plot during training. Even though our reward function does not include the hand-designed locomotion penalties that are commonly used in prior works (torque, delta torque, foot slip, joint speed, and action), we show that all these penalties are minimized in our training framework as a byproduct of minimizing energy consumption.

Learning via Distillation and RL: We sidestep this issue via stage-wise distillation approach. We first train our fixed-velocity policies as described in this chapter which lead to walking, trotting and bouncing (galloping) gaits. We then treat these policies as experts and collect demonstration data from them to self-supervise and bootstrap the initial training phase of the velocity-conditioned policy. These three policies serve as experts at three velocities at low (0.375 m/s), median (0.9 m/s) and high (1.5 m/s), represented as three-dimensional one-hot vectors, which are fed into the velocity-conditioned policy as input. Expert supervisions guide the velocity-conditioned policy to follow the energy-efficient gaits at the three expert velocities. To learn motor skills and smooth gait transition at intermediate velocities in the continuous range of 0.375 m/s to 1.5 m/s, the learning relies on the velocity-conditioned RL rewards only without expert supervisions. To represent a randomly sampled target velocity to feed into the policy, we use an interpolated vector based on the closest two experts’ one-hot velocity representations (e.g. 1.2 m/s is represented as [0, 0.5, 0.5] and 0.5 m/s as [0.238, 0.762, 0]). We linearly anneal the L2 supervision loss from the expert supervision (for the three velocity modes) as the training progresses. Towards the end, the learning relies only on the velocity-conditioned RL loss for the entire velocity range. Details are in the supplementary.

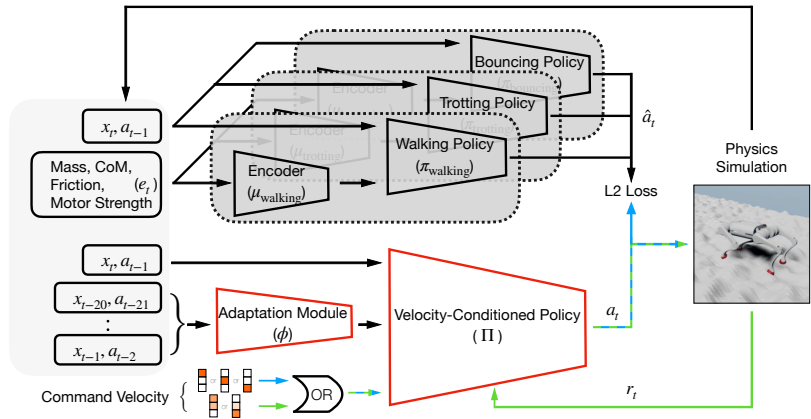


Figure 3.6: Learning scheme for smooth gait transition. Trainable modules are in red. The blue path indicates forward and backward passes for the L2 loss on action, whereas the green path is for training by RL.

Real-World Smooth Gait Transition: We test our velocity-conditioned policy in the wild and want to highlight the smooth gait transition happening at different speeds. Video is in the supplementary.

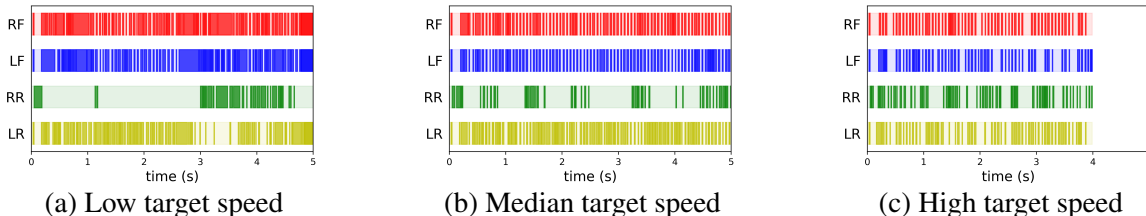


Figure 3.8: Resulting gait plots of policies without minimizing energy consumption. All policies exhibit a high-frequency tapping behavior that is not energy-efficient and not transferable to robots in the real world. In (c), the robot in simulation falls off the terrain after 4 seconds.

Ablation Studies

The two key components we use during policy training to get reliable walking gaits is an energy based reward function, and a fractal terrain. We posit that these ingredients are minimal, and we demonstrate this by ablating each of them below.

Minimizing Energy Consumption Also Minimizes Artificial Penalties: To demonstrate the sufficiency of minimizing energy consumption, we plot the decreasing trend of the artificial penalties during training in Figure 3.7, even though these artificial penalties are not included in our reward function. These include Torque penalty, delta torque penalty, foot slip penalty, joint speed penalty, and the action regularization [191, 197, 72, 53, 196]. A high positive correlation between energy consumption and artificial penalties can be observed. All these artificial penalties are minimized in our training framework as a byproduct of minimizing energy consumption. During training, the robot first learns to stand still with a minimal energy consumption (around epoch 500), and then starts moving forward with a slightly higher energy consumption.

Minimizing Energy Consumption: In Figure 3.8, we show the the resulting gaits of policies trained without the unit energy consumption term in the reward function. All policies show high-frequency tapping behaviors that are not natural and inefficient. The high-frequency joint movements can burn the joint motors in the robot and are not transferable to the real hardware.

Fractal Terrain during Training: We train the policies with energy minimization (Section 3.1) but on a flat terrain without fractal perturbations (Section 3.1). All training trials converge to unnatural and unstable gaits. We found that adding fractal terrain also facilitates a larger foot clearance and robustness which improves real-world hardware deployment. We show the key frames of 2 example unnatural gaits in the supplementary.

3.4 Related Works

Model-Based Optimization for Gaits: Model-based controllers to generate gaits [38] optimize for the stance legs for a given foot contact sequence. They use predefined swing leg motion and show stable gait generation on a real robot system. One class of method is contact-implicit optimization [111, 141, 122], optimizing contact forces along with sequences. However, these gaits do not emerge as an energy-minimizing solution to moving at different speeds. [167] analyzes biped

gaits by optimizing energy consumption, but uses a minimal mechanical model assuming point-mass body and massless legs. In contrast, our swing leg motion, stance leg force and contact sequence, all emerge from minimizing energy consumption of moving of a real robot. We take a functional approach to gaits where we achieve them not as predefined movements, but as a consequence of energy minimization.

Learning for Legged Locomotion and Gaits: Data-driven learning for legged locomotion has shown robust controllers for quadrupedal robots [72, 102, 33, 63, 136]. Of these, [102, 72] focus on a controller on complex terrains but do not synthesize and analyze leg patterns of the robot across diverse gaits at different target speeds. [33] demonstrates a learning method to select gait controllers, but the low-level gait controllers are predefined primitives. Concurrent work [191] shows the learned gaits and gait transition, but their method relies on many human priors (e.g. predefined cyclic motion priors), shows trotting at high speed with short periods of flight phase, and requires powerful on-board computation power (Apple M1 chip) to solve online MPC optimization. In graphics, [194] shows low energy also plays a role in generating natural locomotion animations.

3.5 Summary

This work demonstrates that energy consumption plays a key role in the emergence of natural locomotion patterns in animals by following an analysis-by-synthesis approach of showing the result in real quadruped robots. This allows generation of the straight line lab gaits, as well as unstructured complex terrain gaits in animals. We perform thorough analysis of gait patterns, show their robustness under disturbances, and propose a learning pipeline for smooth gait transition. Interestingly, the gaits obtained by our robot are most similar to those in horses and sheep – the animals which are close to it in terms of the Froude number metric space. In the next chapter, we will apply this framework of emergent gaits along with rapid motor adaptation developed in the previous chapter to the task of visual walking and show how a small robot dog can climb tall stairs and walk on stepping stones from just onboard egocentric vision.

Chapter 4

Legged Locomotion in Challenging Terrains using Egocentric Vision

Of what use is vision during locomotion? Clearly, there is a role of vision in navigation – using maps or landmarks to find a trajectory in the 2D plane to a distant goal while avoiding obstacles. But given a local direction in which to move, it turns out that both humans [107] and robots [102, 96] can do remarkably well at blind walking. Where vision becomes necessary is for locomotion in challenging terrains. In an urban environment, staircases are the most obvious example. In the outdoors, we can deal with rugged terrain such as scrambling over rocks, or stepping from stone to stone to cross a stream of water. There is a fair amount of scientific work studying this human capability and showing tight coupling of motor control with vision [116, 133, 121]. In this chapter, we will develop this capability for a quadrupedal walking robot equipped with egocentric depth vision. We use a reinforcement learning approach trained in simulation, which we are directly able to transfer to the real world. Figure 4.1 and the accompanying videos shows some examples of our robot walking guided by vision.

Humans receive an egocentric stream of vision which is used to control feet placement, typically without conscious planning. As children we acquire it through trial and error [1] but for adults it is an automatized skill. Its unconscious execution should not take away from its remarkable sophistication. The footsteps being placed now are based on information collected some time ago. Typically, we don't look at the ground underneath our feet, rather at the upcoming piece of ground in front of us a few steps away [107, 116, 133, 121]. A short term memory is being created which persists long enough to guide foot placement when we are actually over that piece of ground. Finally, note that we learn to walk through bouts of steps, not by executing pre-programmed gaits [1].

We take these observations about human walking as design principles for the visually-based walking controller for an AI robot. The walking policy is trained by reinforcement learning with a recurrent neural network being used as a short term memory of recent egocentric views, proprioceptive states, and action history. Such a policy can maintain memory of recent visual

This chapter is based on joint work with Ananye Agarwal, Jitendra Malik and Deepak Pathak [3]

information to retrieve characteristics of the terrain under the robot or below the rear feet, which might no longer be directly visible in the egocentric view.

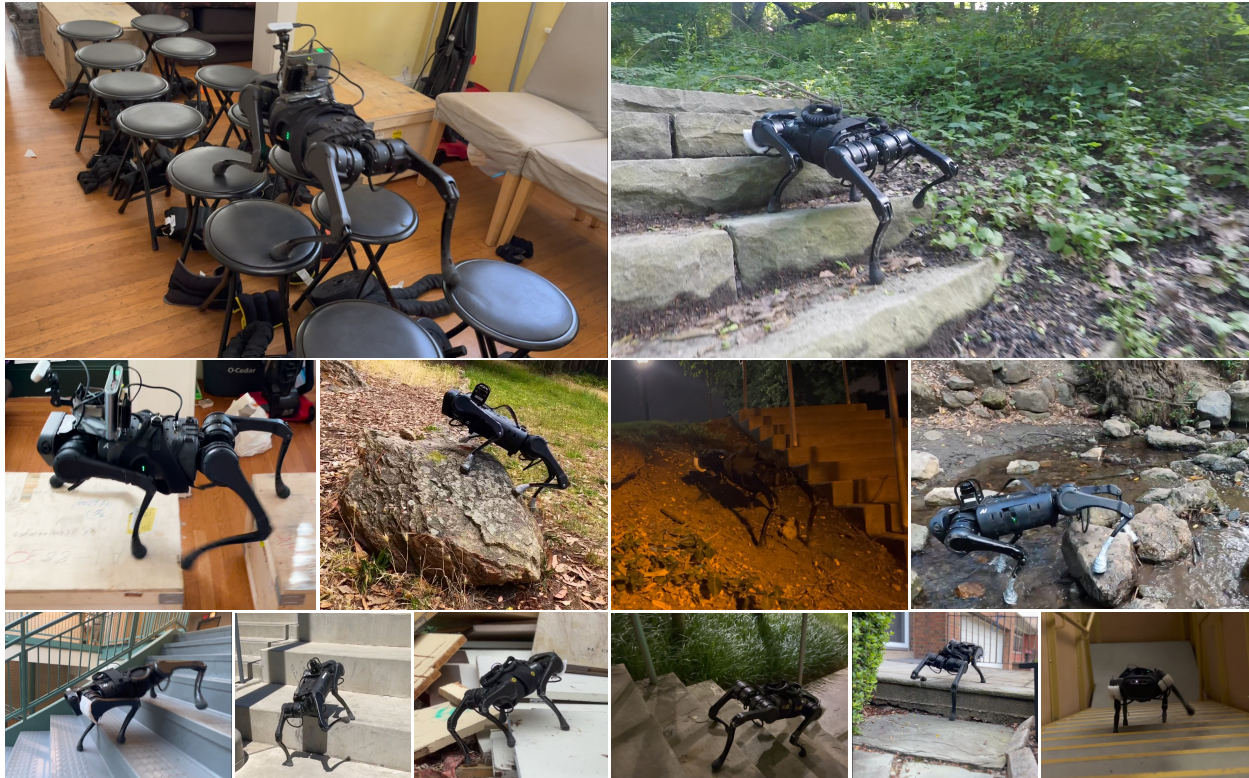


Figure 4.1: Our robot can traverse a variety of challenging terrain in indoor and outdoor environments, urban and natural settings during day and night using a single front-facing depth camera. The robot can traverse curbs, stairs and moderately rocky terrain. Despite being much smaller than other commonly used legged robots, it is able to climb stairs and curbs of a similar height. Videos at <https://vision-locomotion.github.io>

In contrast, prior locomotion techniques rely on the metric elevation map of the terrain around and under the robot [118, 86, 78] to plan foot steps and joint angles. The elevation map is constructed by fusing information from multiple depth images (collected over time). This fusion of depth images into a single elevation map requires the relative pose between cameras at different times. Hence, tracking is required in the real world to obtain this relative pose using visual or inertial odometry. This is challenging because of noise introduced in sensing and odometry, and hence, previous methods add different kinds of structured noise at training time to account for the noise due to pose estimation drift [43, 117, 131]. The large amount of noise hinders the ability of such systems to perform reliably on gaps and stepping stones. We use vision as a first class citizen and show all the uneven terrain capabilities along with a high success rate on crossing gaps and stepping stones.

The design principle of not having pre-programmed gait priors turns out to be quite advantageous

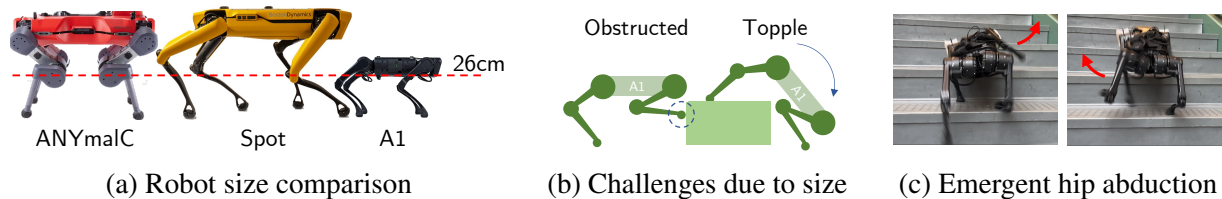


Figure 4.2: A smaller robot (a) faces challenges in climbing stairs and curbs due to the stair obstructing its feet while going up and a tendency to topple over when coming down (b). Our robot deals with this by climbing using a large hip abduction that automatically emerges during training (c).

for our relatively small robot¹ (fig. 4.2). Predefined gait priors or reference motions fail to generalize to obstacles of even a reasonable height because of the relatively small size of the quadruped. The emergent behaviors for traversing complex terrains without any priors enable our robot with a hip joint height of 28cm to traverse the stairs of height upto 25cm, 89% relative to its height, which is significantly higher than any existing methods which typically rely on gait priors.

Since our robot is small and inexpensive, it has limited onboard compute and sensing. It uses a single front-facing D435 camera for exteroception. In contrast, AnymalC has four such cameras in addition to two dome lidars. Similarly, Spot has 5 depth cameras around its body. Our policy computes actions with a single feedforward pass and requires no tracking. This frees us from running optimization for MPC or localization which requires expensive hardware to run in real-time.

Overall, this use of learning “all the way” and the tight coupling of egocentric vision with motor control are the distinguishing aspects of our approach.

4.1 Method: Legged Locomotion from Egocentric Vision

Our goal is to learn a walking policy that maps proprioception and depth input to target joint angles at 50Hz. Since depth rendering slows down the simulation by an order of magnitude, directly training this system using reinforcement learning (RL) would require billions of samples to converge making this intractable with current simulations. We therefore employ a two-phase training scheme. In phase 1, we use low resolution scandots located under the robot as a proxy for depth images. Scandots refer to a set of (x, y) coordinates in the robot’s frame of reference at which the height of the terrain is queried and passed as observation at each time step (fig. 4.3). These capture terrain geometry and are cheap to compute. In phase 2, we use depth and proprioception as input to an RNN to implicitly track the terrain under the robot and directly predict the target joint angles at 50Hz. This is supervised with actions from the phase 1 policy. Since supervised learning is orders of magnitude more sample efficient than RL, our proposed pipeline enables training the whole system on a single GPU in a few days. Once trained, our deployment policy does not construct metric elevation maps, which typically rely on metric localization, and instead directly predicts joint angles from depth and proprioception.

¹A1 standing height is 40cm as measured by us. Spot, ANYmalC both are 70cm tall reported here and here.

One potential failure mode of this two-phase training is that the scandots might contain more information than what depth can infer. To get around this, we choose scandots and camera field-of-view such that phase 2 loss is low. We formally show that this guarantees that the phase 2 policy will have close to optimal performance in Thm 4.1.1 below.

Theorem 4.1.1. $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, R, \gamma)$ be an MDP with state space \mathcal{S} , action space \mathcal{A} , transition function $P : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$, reward function $R : \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$ and discount factor γ . Let $V^1(s)$ be the value function of the phase 1 policy that is trained to be close to optimal value function $V^*(s)$, i.e., $|V^*(s) - V^1(s)| < \epsilon \forall s \in \mathcal{S}$, and $\pi^1(s)$ be the greedy phase 1 policy obtained from $V^1(s)$. Suppose the phase 2 policy operates in a different state space \mathcal{S}' given by a mapping $f : \mathcal{S} \rightarrow \mathcal{S}'$. If the phase 2 policy is close to phase 1 $|\pi^1(s) - \pi^2(f(s))| < \eta \forall s$ and R, P are Lipschitz continuous, then the return of phase 2 policy is close to optimal everywhere, i.e., $\forall s$, $\left| V^*(s) - V^{\pi^2}(f(s)) \right| < \frac{2\epsilon\gamma + \eta c}{1-\gamma}$ where $c \propto \sum_{s \in \mathcal{S}} V^*(s)$ is a large but bounded constant. (proof in sec. C.1)

We instantiate our training scheme using two different architectures. The monolithic architecture is an RNN that maps from raw proprioception and vision data directly to joint angles. The RMA architecture follows [96, 143], and contains an MLP base policy that takes γ_t (which encodes the local terrain geometry) along with the extrinsics vector z_t (which encodes environment parameters [96]), and proprioception x_t to predict the target joint angles. An estimate of γ_t is generated by an RNN that takes proprioception and vision as inputs. While the monolithic architecture is conceptually simpler, it implicitly tracks γ_t and z_t in its weights and is hard to disentangle. In contrast, the RMA architecture allows direct access to each input (γ_t or z_t) through latent vectors. This allows the possibility of swapping sensors (like replacing depth by RGB) or using one stream to supervise the other while keeping the base motor policy fixed.

Phase 1: Reinforcement Learning from Scandots

Given the scandots m_t , proprioception x_t , commanded linear and angular velocity $u_t^{\text{cmd}} = (v_x^{\text{cmd}}, \omega_z^{\text{cmd}})$ we learn a policy using PPO without gait priors and with reward functions that minimize energetics to walk on a variety of terrains. Proprioception consists of joint angles, joint velocities, angular velocity, roll and pitch measured by onboard sensors in addition to the last policy actions a_{t-1} . Let $o_t = (m_t, x_t, u_t^{\text{cmd}})$ denote the observations. The RMA policy also takes privileged information e_t as input which includes center-of-mass of robot, ground friction, and motor strength.

Monolithic The scandots m_t are first compressed to γ_t and then passed with the rest of the observations to a GRU that predicts the joint angles.

$$\gamma_t = \text{MLP}(m_t) \quad (4.1)$$

$$a_t = \text{GRU}_t(x_t, \gamma_t, u_t^{\text{cmd}}) \quad (4.2)$$

the subscript t on the GRU indicates that it is stateful.

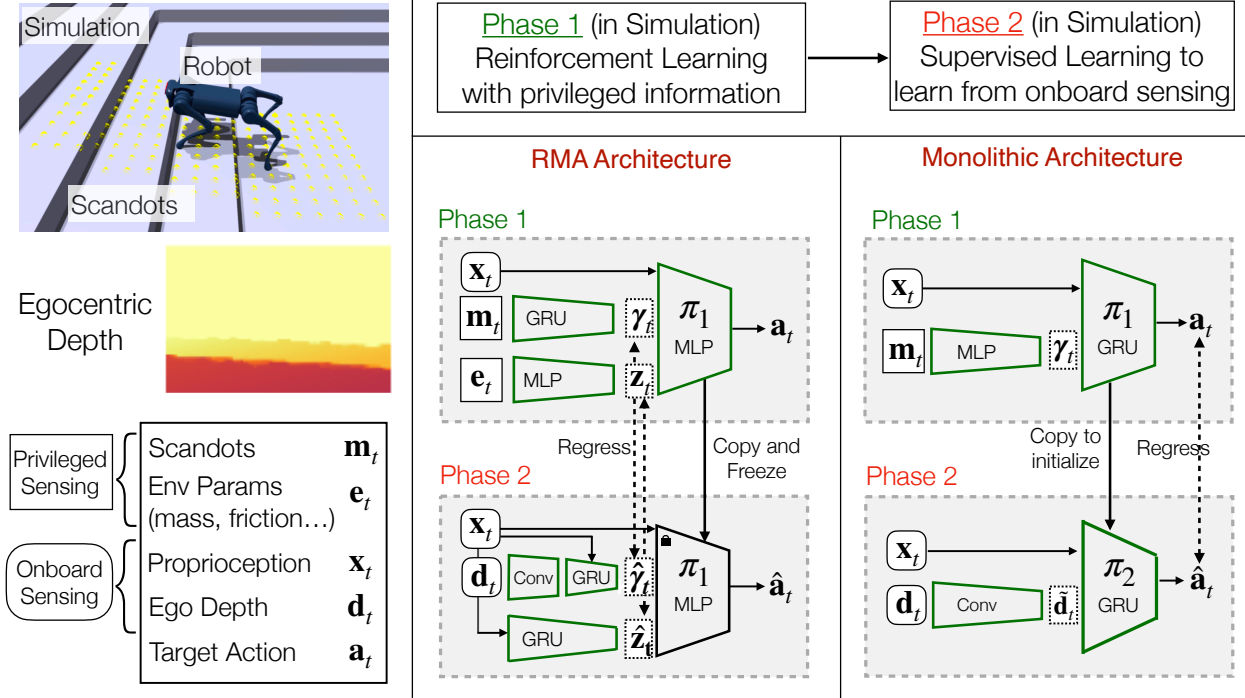


Figure 4.3: We train our locomotion policy in two phases to avoid rendering depth for too many samples. In phase 1, we use RL to train a policy π^1 that has access to scandots that are cheap to compute. In phase 2, we use π^1 to provide ground truth actions which another policy π^2 is trained to imitate. This student has access to depth map from the front camera. We consider two architectures (1) a monolithic one which is a GRU trained to output joint angles with raw observations as input (2) a decoupled architecture trained using RMA [96] that is trained to estimate vision and proprioception latents that condition a base feedforward walking policy.

RMA Instead of using a monolithic memory based architecture for the controller, we use an MLP as the controller, pushing the burden of maintaining memory and state on the various inputs to the MLP. Concretely, we process the environment parameters (e_t) with an MLP and the scandots (m_t) with a GRU to get z_t and γ_t respectively which are given as input to the base feedforward policy.

$$\gamma_t = \text{GRU}_t(m_t) \quad (4.3)$$

$$z_t = \text{MLP}(e_t) \quad (4.4)$$

$$a_t = \text{MLP}(x_t, \gamma_t, z_t, u_t^{\text{cmd}}) \quad (4.5)$$

Both the phase 1 architectures are trained using PPO [158] with backpropagation through time [179] truncated at 24 timesteps.

Rewards We extend the reward functions proposed in [96, 48] to simply penalizing the energy consumption along with additional penalties to prevent damage to hardware on complex terrain

(sec. C.2). Importantly, we do not impose any gait priors or predefined foot trajectories and let optimal gaits that are stable and natural to emerge for the task.

- *Absolute work penalty* $-|\boldsymbol{\tau} \cdot \mathbf{q}|$ where $\boldsymbol{\tau}$ are the joint torques. We use the absolute value so that the policy does not learn to get positive reward by exploiting inaccuracies in contact simulation.
- *Command tracking* $v_x^{\text{cmd}} - |v_x^{\text{cmd}} - v_x| - |\omega_z^{\text{cmd}} - \omega_z|$ where v_x is velocity of robot in forward direction and ω_z is yaw angular velocity (x, z are coordinate axes fixed to the robot).
- *Foot jerk penalty* $\sum_{i \in \mathcal{F}} \|\mathbf{f}_t^i - \mathbf{f}_{t-1}^i\|$ where \mathbf{f}_t^i is the force at time t on the i^{th} rigid body and \mathcal{F} is the set of feet indices. This prevents large motor backlash.
- *Feet drag penalty* $\sum_{i \in \mathcal{F}} \mathbb{I}[f_z^i \geq 1\text{N}] \cdot (|v_x^i| + |v_y^i|)$ where \mathbb{I} is the indicator function, and v_x^i, v_y^i is velocity of i^{th} rigid body. This penalizes velocity of feet in the horizontal plane if in contact with the ground preventing feet dragging on the ground which can damage them.
- *Collision penalty* $\sum_{i \in \mathcal{C} \cup \mathcal{T}} \mathbb{I}[\mathbf{f}^i \geq 0.1\text{N}]$ where \mathcal{C}, \mathcal{T} are the set of calf and thigh indices. This penalizes contacts at the thighs and calves of the robot which would otherwise graze against edges of stairs and discrete obstacles.
- *Survival bonus* constant value 1 at each time step to prioritize survival over following commands in challenging situations.

Training environment Similar to [152] we generate different sets of terrain (fig. C.1) of varying difficulty level. Following [96], we generate fractal variations over each of the terrains to get robust walking behaviour. At training time, the environments are arranged in a 6×10 matrix with each row having terrain of the same type and difficulty increasing from left to right. We train with a curriculum over terrain [152] where robots are first initialized on easy terrain and promoted to harder terrain if they traverse more than half its length. They are demoted to easier terrain if they fail to travel at least half the commanded distance $v_x^{\text{cmd}} T$ where T is maximum episode length. We randomize parameters of the simulation (tab. C.2) and add small i.i.d. gaussian noise to observations for robustness (tab. C.1).

Phase 2: Supervised Learning

In phase 2, we use supervised learning to distil the phase 1 policy into an architecture that only has access to sensing available onboard: proprioception (\mathbf{x}_t) and depth \mathbf{d}_t .

Monolithic We create a copy of the recurrent base policy 4.2. We preprocess the depth map through a convnet before passing it to the base policy.

$$\tilde{\mathbf{d}}_t = \text{ConvNet}(\mathbf{d}_t) \quad (4.6)$$

$$\hat{\mathbf{a}}_t = \text{GRU}_t(\mathbf{x}_t, \tilde{\mathbf{d}}_t, \mathbf{a}_t^{\text{cmd}}) \quad (4.7)$$

We train with DAgger [151] with truncated backpropagation through time (BPTT) to minimize mean squared error between predicted and ground truth actions $\|\hat{\mathbf{a}}_t - \mathbf{a}_t\|^2$. In particular, we unroll the student inside the simulator for $N = 24$ timesteps and then label each of the states encountered with the ground truth action \mathbf{a}_t from phase 1.

RMA Instead of retraining the whole controller, we only train estimators of γ_t and z_t , and use the same base policy trained in phase 1 (eqn. 4.5). The latent $\hat{\gamma}$, which encodes terrain geometry, is estimated from history of depth and proprioception using a GRU. Since the camera looks in front of the robot, proprioception combined with depth enables the GRU to implicitly track and estimate the terrain under the robot. Similar to [96], history of proprioception is used to estimate extrinsics \hat{z} .

$$\tilde{\mathbf{d}}_t = \text{ConvNet}(\mathbf{d}_t) \quad (4.8)$$

$$\hat{\gamma}_t = \text{GRU}_t(\mathbf{x}_t, \mathbf{u}_t^{\text{cmd}}, \tilde{\mathbf{d}}_t) \quad (4.9)$$

$$\hat{z}_t = \text{GRU}_t(\mathbf{x}_t, \mathbf{u}_t^{\text{cmd}}) \quad (4.10)$$

$$\mathbf{a}_t = \text{MLP}(\mathbf{x}_t, \mathbf{u}_t^{\text{cmd}}, \hat{\gamma}_t, \hat{z}_t) \quad (4.11)$$

As before, this is trained using DAgger with BPTT. The vision GRU 4.9 and convnet 4.8 are jointly trained to minimize $\|\hat{\gamma}_t - \gamma_t\|^2$ while the proprioception GRU 4.10 minimizes $\|\hat{z}_t - z_t\|^2$.

Deployment The student can be deployed as-is on the hardware using only the available onboard compute. It is able to handle camera failures and the asynchronous nature of depth due to the randomizations we apply during phase 1. It is robust to pushes, slippery surfaces and large rocky surfaces and can climb stairs, curbs, and cross gaps and stepping stones.

4.2 Experimental Setup

We use the Unitree A1 robot pictured in Fig. 4.2. The robot has 12 actuated joints. The robot has a front-facing Intel RealSense depth camera in its head. The onboard compute consists of the UPboard and a Jetson NX. The policy operates at 50Hz and sends joint position commands which are converted to torques by a low-level PD controller running at 400Hz. Depth map is obtained from a Intel RealSense camera inside the head of the robot. The camera captures images every $100\text{ms} \pm 20\text{ms}$ at a resolution of 480×848 . We preprocess the image by cropping 200 white pixels from the left, applying nearest neighbor hole-filling and downsampling to 58×87 . This is passed through a backbone to obtain the compressed $\tilde{\mathbf{d}}_t$ 4.8, 4.6 which is sent over a UDP socket to the base policy. This has a latency of $10 \pm 10\text{ms}$ which we account for during phase 2.

We use the IsaacGym (IG) simulator with the legged_gym library [152] to train our walking policies. We construct a large terrain map with 100 sub-terrains arranged in a 20×10 grid. Each row has the same type of terrain arranged in increasing difficulty while different rows have different terrain.

Baselines We compare against two baselines, each of which uses the same number of learning samples for both RL phase and supervised learning phase.

- **Blind policy** trained with the scandots observations \mathbf{m}_t masked with zeros. This baseline must rely on proprioception to traverse terrain and helps quantify the benefit of vision for walking.

Terrain	Average x -Displacement (\uparrow)				Mean Time to Fall (s)			
	RMA	MLith	Noisy	Blind	RMA	MLith	Noisy	Blind
Slopes	43.98	44.09	36.14	34.72	88.99	85.68	70.25	67.07
Stepping Stones	18.83	20.72	1.09	1.02	34.3	41.32	2.51	2.49
Stairs	31.24	42.4	6.74	16.64	69.99	90.48	15.77	39.17
Discrete Obstacles	40.13	28.64	29.08	32.41	85.17	57.53	59.3	66.33
Total	134.18	135.85	73.05	84.79	278.45	275.01	147.83	175.06

Table 4.1: We measure the average displacement along the forward axis and mean time to fall for all methods on different terrains in simulation. For each method, we train a single policy for all terrains and use that for evaluation. We see that the monolithic (MLith) and RMA architectures of our method outperform the noisy and blind baselines by 60-90% in terms of total mean time to fall and average displacement. Vision is not strictly necessary for traversing slopes and the baselines make significant progress on this terrain, however, MLith and RMA travel upto 25% farther. The difference is more stark on stepping stones where blind and noisy baselines barely make any progress due to not being able to locate positions of the stones, while MLith and RMA travel for around 20m. Noisy and blind make some progress on stairs and discrete obstacles, but our methods travel upto 6.3 times farther.

- **Noisy Methods** which rely on elevation maps need to fuse multiple depth images captured over time to obtain a complete picture of terrain under and around the robot. This requires camera pose relative to the first depth input, which is typically estimated using vision or inertial odometry [43, 131, 117]. However, these pose estimates are typically noisy resulting in noisy elevation maps [118]. To handle this, downstream controllers trained on this typically add a large noise in the elevation maps during training. Similar to [118], we train a teacher with ground truth, noiseless elevation maps in phase 1 and distill it to a student with large noise, with noise model from [118], added to the elevation map. We simulate a latency of 40ms in both the phases of training to match the hardware. This baseline helps in understanding the effect on performance when relying on pose estimates which introduce additional noise in the pipeline.

4.3 Results and Analysis

Simulation Results We report mean time to fall and mean distance travelled before crashing for different terrain and baselines in Table 4.1. For each method, we train a single policy for all terrains and use that for evaluation. Although the blind policy makes non trivial progress on stairs, discrete obstacles and slopes, it is significantly less efficient at traversing these terrains. On slopes our methods travel upto 27% farther implying that the blind baseline crashes early. Similarly, on stairs and discrete obstacles the distance travelled by our methods is much greater (upto 90%). On slopes and stepping stones the noisy and blind baselines get similar average distances and mean time to fall and both are worse than our policy. This trend is even more significant on the stepping stones

terrain where all baselines barely make any progress while our methods travel upto 20m. The blind policy has no way of estimating the position of the stone and crashes as soon as it steps into the gap. For the noisy policy, the large amount of added noise makes it impossible for the student to reliably ascertain the location of the stones since it cannot rely on proprioception any more. We note that the blind baseline is better than the noisy one on stairs. This is because the blind baseline has learnt to use proprioception to figure out location of stairs. On the other hand, the noisy policy cannot learn to use proprioception since it is trained via supervised learning. However, the blind baseline bumps into stairs often is not very practical to run on the real robot. The noisy baseline works well in [118] possibly because of predefined foot motions which make the phase 2 learning easier. However, as discussed in the beginning of this chapter predefined motions will not work for our small robot.

Real World Comparisons We compare the performance of our methods to the blind baseline in the real world. In particular we have 4 testing setups as shows in fig. 4.4: Upstairs, Downstairs, Gaps and Stepping stones. While we train a single phase 1 policy for all terrain, for running baselines, we obtain different phase 2 policies for stairs vs. stepping stones and gaps. Different phase 2 policies are obtained by changing the location of the camera. We use the in-built camera inside the robot for stairs and a mounted external camera for stepping stones and gaps. The in-built camera is less prone to damage but the stepping stones are gaps are not clearly visible since it is horizontal. This is done for convenience, but we also have a policy that traverses all terrain using the same mounted camera.

We see that the blind baseline is incapable of walking upstairs beyond a few steps and fails to complete the staircase even once. Although existing methods have shown stairs for blind robots, we note that our robot is relatively smaller making it a more challenging task for a blind robot. On downstairs, we observe that the blind baseline achieves 100% success, although it learns to fall on every step and stabilize leading to a very high impact gait which led to the detaching of the rear right hip of the robot during our experiments. We additionally show results in stepping stones and gaps, where the blind robot fails completely establishing the hardness of these setups and the necessity of vision to solve them. We show a 100% success on all tasks except for stepping stone on which we achieve 94% success, which is very high given the challenging setup.

Urban Environments We experiment on stairs, ramps and curbs (fig. 4.1). The robot was successfully able to go upstairs as well as downstairs for stairs of height upto 24cm in height and 28cm as the lowest width. Since the robot has to remember terrain under its body from visual history, it sometimes misses a step, but shows impressive recovery behaviour and continues climbing or descending. The robot is able to climb curbs and obstacles as high as 26cm which is almost as high as the robot 4.2. This requires an emergent hip abduction movement because the small size of the robot doesn't leave any space between the body and stair for the leg to step up. This behavior emerges because of our tabula rasa approach to learning gaits without reliance on priors or datasets of natural motion.

Gaps and Stepping Stones We construct an obstacle course consisting of gaps and stepping stones out of tables and stools (fig. 4.4). For this set of experiments we use a policy trained on

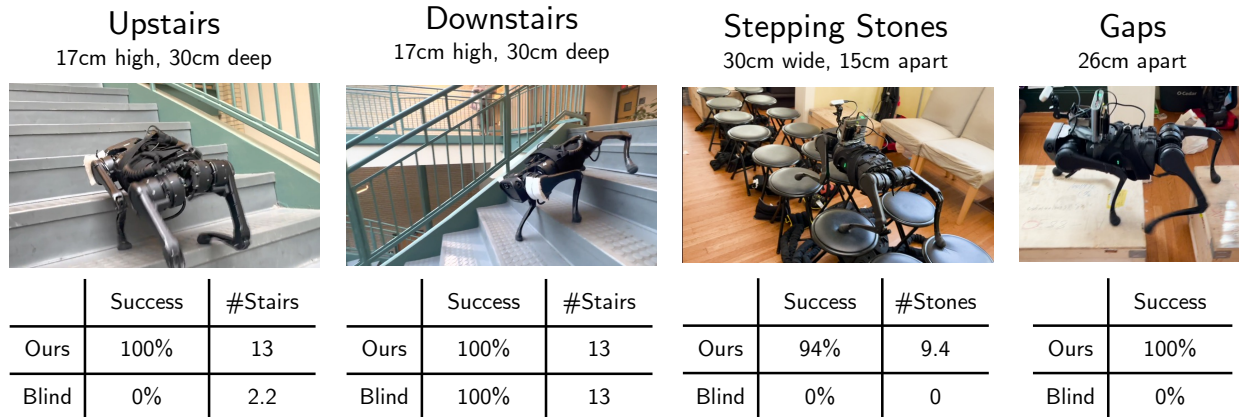


Figure 4.4: We show success rates and time-to-failure (TTF) for our method and the blind baseline on curbs, stairs, stepping stones and gaps. We use a separate policy for stairs which is distilled to front camera, and use a separate policy trained on stepping stones distilled to the top camera which we use for gaps and stepping stones. We observe that our method solves all the tasks perfectly except for the stepping stone task in which the robot achieves 94% success. The blind baseline fails completely on gaps and stepping stones. For upstairs, it makes some progress, but fails to complete the entire staircase even once, which is expected given the small size of the robot. The blind policy completes the downstairs task 100% success, although it learns a very high impact falling gait to solve the task. In our experiments, the robot dislocates its real right leg during the blind downstairs trials.

stepping stones on gaps, and distilled onto the top camera instead of the front camera. The robot achieves a 100% success rate on gaps of upto 26cm from egocentric depth and 94% on difficult stepping stones. The stepping stones experiment shows that our visual policy can learn safe foothold placement behavior even without an explicit elevation map or foothold optimization objectives. The blind baseline achieves zero success rate on both tasks and falls as soon as any gap is encountered.

Natural Environments We also deploy our policy on outdoor hikes and rocky terrains next to river beds (fig. 4.1). We see that the robot is able to successfully traverse rugged stairs covered with dirt, small pebbles and some large rocks. It also avoids stumbling over large tree roots on the hiking trail. On the beach, we see that the robot is able to successfully navigate the terrain despite several slips and unstable footholds given the nature of the terrain. We see that the robot sometimes gets stuck in the crevices and in some cases shows impressive recovery behavior as well.

4.4 Related Work

Legged locomotion Legged locomotion an important problem which has been studied for decades. Several classical works use model based techniques, or define heuristic reactive controllers to achieve the task of walking [119, 146, 55, 192, 166, 80, 85, 10, 73, 13, 17, 69, 74]. This method has led to several promising results in the real world, although they still lack the generality needed to deploy

them in the real world. This has motivated work in using RL for learning to walk in simulation [158, 106, 120, 51], and then successfully deploy them in a diverse set of real world scenarios [169, 171, 137, 183, 125, 72, 169, 65]. Alternatively, a policy learned in simulation can be adapted at test-time to work well in real environments [196, 193, 136, 201, 197, 195, 165, 28, 96, 50, 164, 189]. However, most of these methods are blind, and only use proprioceptive signal to walk.

Locomotion from Elevation Maps To achieve visual control of walking, classical methods decouple the perception and control aspects, assuming a perfect output from perception, such as an elevation map, and then using it for planning and control [42, 132, 101, 100, 88]. The control part can be further decoupled into searching for feasible footholds on the elevation map and then execute it with a low-level policy [25]. The foothold feasibility scores can either be estimated heuristically [180, 26, 78, 114, 44, 86, 5] or learned [91, 82, 178, 115, 109]. Other methods forgo explicit foothold optimization and learn traversability maps instead [187, 23, 61, 53]. Recent methods skip foothold planning and directly train a deep RL policy that takes the elevation map as input and outputs either low-level motor primitives [118, 173] or raw joint angles [152, 134, 135, 182]. Elevation maps can be noisy or incorrect and dealing with imperfect maps is a major challenge to building robust locomotion systems. Solutions to this include incorporating uncertainty in the elevation map [42, 14, 43] and simulating errors at training time to make the walking policy robust to them [118].

Locomotion from Egocentric Depth Closest to ours is the line of work that doesn't construct explicit elevation maps and predicts actions directly from depth. [189] learn a policy for obstacle avoidance from depth on flat terrain, [76] train a hierarchical policy which uses depth to traverse curved cliffs and mazes in simulation, [40] use lidar scans to show zero-shot generalization to difficult terrains. [198] train a policy to step over gaps by predicting high-level actions using depth from the head and below the torso. Relatedly, [112] train a high-level policy to jump over gaps from egocentric depth using a whole body impulse controller. In contrast, we directly predict target joint angles from egocentric depth without constructing metric elevation maps.

4.5 Summary

In this work, we show an end-to-end approach to walking with egocentric depth. The proposed approach does not rely on explicit mapping, predefined gaits, or any other heuristics. The entire system is learned end to end in simulation and then deployed zero-shot in the real world. Once deployed in the real world, it is capable of walking on a large variety of terrains including stairs, gaps and stepping stones.

Chapter 5

In-hand Rotation via Rapid Adaptation

In this and the following chapters, we will develop RMA in the context of other challenging tasks in robotics, starting with dexterous manipulation. Humans are remarkably good at manipulating objects in-hand – they can even adapt to new objects of different shapes, sizes, mass and materials with no apparent effort. While several works have shown in-hand object rotation with real-world multi-fingered hands for a single or a few objects [129, 130, 162, 123], truly generalizable in-hand manipulation remains an unsolved challenge of robotics.

In this chapter, we demonstrate that it is possible to train an adaptive controller capable of rotating diverse objects over the z -axis with the fingertips of a multi-fingered robot hand (Figure 5.1). This task is a simplification of the general in-hand reorientation task, yet still quite challenging for robots since at all times the fingers need to maintain a dynamic or static force closure on the object to prevent it from falling (as it can not make use of any other supporting surface such as the palm).

Our approach is inspired by the recent advances in legged locomotion [102, 97] using reinforcement learning. The core of these works is to learn a compressed representation of different terrain properties (called *extrinsics*) for walking, which is jointly trained with the control policy. During deployment, the *extrinsics* is estimated online and the controller can perform rapid adaptation to it. Our key insight is that, despite the diversity of real-world objects, for the task of in-hand object rotation, the important physical properties such as local shape, mass, and size as perceived by the *fingertips* can be compressed to a compact representation. Once such a compressed representation (*extrinsics*) of different objects is learned, the controller can estimate it online from proprioception history and use it to adaptively manipulate a diverse set of objects.

Specifically, we encode the object’s intrinsic properties (such as mass and size) to an *extrinsics* vector, and train an adaptive policy with it as an input. The learned policy can robustly and efficiently rotate different objects in simulation environments. However, we do not have access to the *extrinsics* when we deploy the policy in the real world. To tackle this problem, we use the rapid motor adaptation [97] to learn an adaptation module which estimate the *extrinsic* vector, using the discrepancy between observed proprioception history and the commanded actions. This adaptation

This chapter is based on joint work with Haozhi Qi, Roberto Calandra, Yi Ma, Jitendra Malik [143]

module can also be trained solely in simulation via supervised learning. The concept of estimating physical properties using proprioceptive history has been widely used in locomotion [102, 97, 49] but has not yet been explored for in-hand manipulation.

Train in Simulation

Directly Deploy in the Real World



Figure 5.1: **Left:** Our controller is trained only in simulation on simple cylindrical objects of different sizes and weights. **Right:** Without any real world fine-tuning, the controller can be deployed to a real robot on a diverse set of objects with different shapes, sizes and weights (object mass and the shortest/longest diameter axis length along the fingertips are shown in the figure) using only proprioceptive information. **Website:** Emergence of natural stable finger gaits can be observed in the learned control policy.

Experimental results on a multi-finger Allegro Hand [181] show that our method can successfully rotate over 30 objects of diverse shapes, sizes (from 4.5 cm to 7.5 cm), mass (from 5 g to 200 g), and other physical properties (e.g., deformable or soft objects) in the real world. We also observe that an adaptive and smooth finger gait emerges from the learning process. Our approach shows the surprising effectiveness of using only proprioception sensory signals for adaptation to different objects, even without the usage of vision and tactile sensing. To further understand the underlying mechanisms of our approach, we studied the estimated extrinsics when manipulating different objects. We find interpretable extrinsic values that correlate to mass and scale changes, and that a low-dimensional structure of the embedding does exist, both of which are critical to our generalization ability.

5.1 Related Work

Classic Control for In-Hand Manipulation. Dexterous in-hand manipulation has been an active research area for decades [128]. Classic control methods usually need an analytical model of the object and robot geometry to perform motion planning for object manipulation. For example, [64, 156] rely on such a model to plan finger movement to rotate objects. [153] assumes objects are piece-wise smooth and use finger-tracking to rotate objects. [12, 122] demonstrate reorientation of different objects in simulation by generating trajectories using optimization. There have been also attempts to deploy systems in the real-world. For example, [45] calculates precise contact locations to plan a sequence of contact locations for twirling objects. [138] plans over a set of predefined grasp strategies to achieve object reorientation using two multi-fingered hands. [52, 34] use throwing or external forces to perturb the object in the air and re-grasp it. Recently, works such as [170, 168] do in-grasp manipulation without breaking the contact. [123] demonstrates complex object reorientation skills using a non-anthropomorphic hand by leveraging the compliance and an accurate pose tracker. The diversity of the objects they can manipulate is still limited due to the intrinsic complexity of the physical world. In contrast to traditional control approaches which may use heuristics or simplified models to solve this task, we instead use model-free reinforcement learning to train an adaptive policy, and use adaptation to achieve generalization.

Reinforcement Learning for In-Hand Manipulation. To get around the need of an accurate object model and physical property measures, in the last few years, there has been a growing interest in using reinforcement learning directly in the real-world for dexterous in-hand manipulation. [174] learns simple in-grasp rolling for cylindrical objects. [98, 126] learns a dynamics model and plan over it for rotating objects on the palm. [60, 99] use human demonstration to accelerate the learning process. However, since reinforcement learning is very sample inefficient, the learned skills are rather simple or have limited object diversity. Although complex skills such as re-orientating a diverse set of objects [68, 24, 84] and tool use [148, 145] can be obtained in simulation, transferring the results to real-world remains challenging. Instead of directly training a policy in the real world, our approach learns the policy entirely in the simulator and aims to directly transfer to the real world.

Sim-to-Real Transfer via Domain Randomization. Several works aim to train reinforcement learning policies using a simulator and directly deploy it in a real-world system. Domain randomization [171] varies the simulation parameters during training to expose the policy to diverse simulation environments so that they can be robustly deployed in the real-world. Representative examples are [129] and [130]. They leverage massive computational resources and large-scale reinforcement learning methods to learn an agile object reorientation skills and solve Rubik’s Cube with a single robot hand. However, they still focus only on manipulating a limited number of objects. [162] learns a finger-gaiting behavior efficiently and transfers to a real robot when hand facing downwards, but they do not only use fingertips and the objects considered are all cubes. Our approach focuses on generalization to a diverse set of objects and can be trained within a few hours.

Sim-to-Real via Adaptation. Instead of relying on domain randomization which is agnostic to current environment parameters, [83] performs system identification via initial calibration, or online adaptive control to estimate the system parameters for Sim-to-Real Transfer. However, learning the

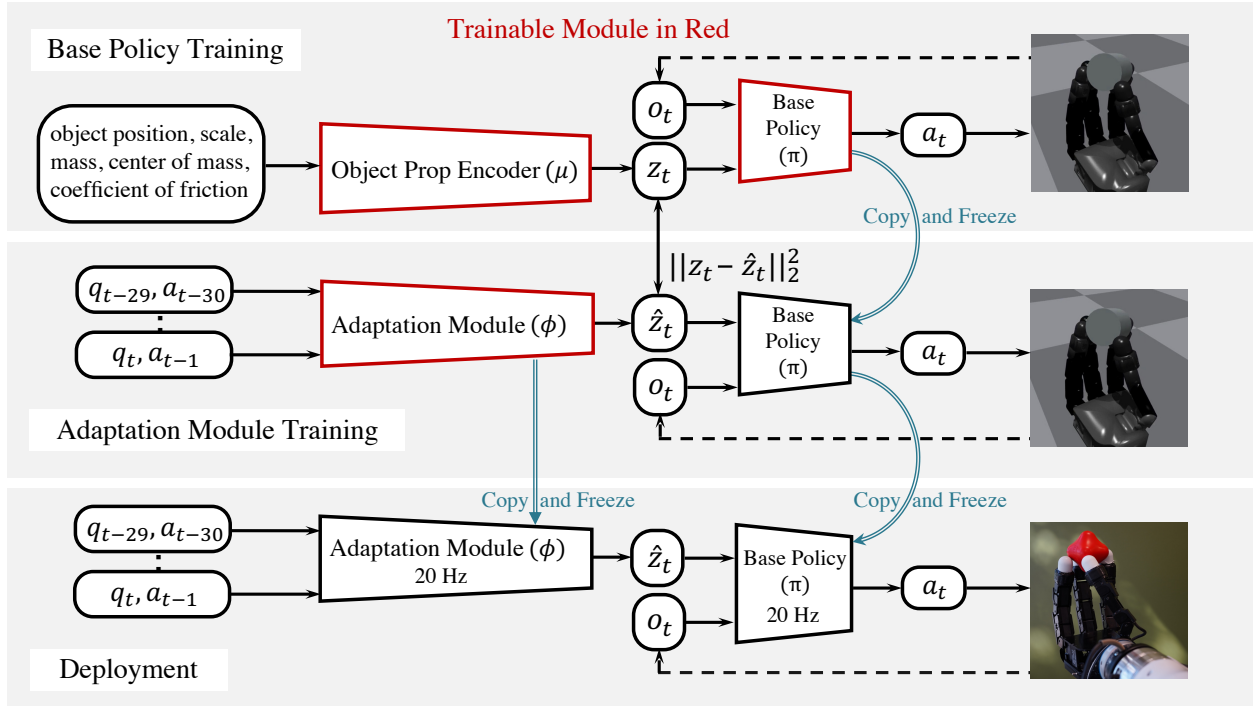


Figure 5.2: An overview of our approach at different training and deployment stages. In *Base Policy Learning*, we jointly optimize μ and π using PPO [158]. The observation o_t only contains three past joint positions and commanded actions. Next, in *Adaptation Module Learning*, we freeze the policy π and use supervised learning to train ϕ which uses proprioception and action history to estimate the extrinsics vector z_t . During *Deployment*, the base policy π uses the extrinsics \hat{z}_t estimated and updated online by ϕ .

exact physical values and alignment between simulation and real-world may be sub-optimal because of the intrinsic inaccuracy of physics simulations. An alternative way is to learn a low dimensional embedding which encodes the environment parameters [102, 97] which is then used by the control policy to act. This paradigm has enabled robust and adaptive locomotion policies. However, it is not straightforward to apply it on the in-hand manipulation task. Our approach demonstrates how to design the reward and training environment to enable a natural and stable controller that can transfer to the real world.

5.2 Rapid Motor Adaptation for In-Hand Object Rotation

An overview of our approach is shown in Figure 5.2. During deployment (Figure 5.2, bottom), our policy infers a low-dimensional embedding of object’s properties such as size and mass from proprioception and action history, which is then used by our base policy to rotate the object. We first describe how we train a *base policy* with object property provided by a simulator, then we discuss

how to train an *adaptation module* that is capable of inferring these properties.

Base Policy Training

Privileged Information. In this chapter, privileged information refers to the object’s properties such as position, size, mass, coefficient of friction, center of mass of the object. This information, denoted by a 9-dim vector $\mathbf{e}_t \in \mathbb{R}^9$ at timestep t , can be accurately measured in simulation. We provide this as an input to the policy, but instead of using \mathbf{e}_t directly, we use an 8-dim embedding (called extrinsics in [97]) $\mathbf{z}_t = \mu(\mathbf{e}_t)$ which gives better generalization behavior as we show in Section 5.4.

Base Policy. Our control policy π takes the current robot joint positions $\mathbf{q}_t \in \mathbb{R}^{16}$, the predicted action $\mathbf{a}_{t-1} \in \mathbb{R}^{16}$ at the last timestep, together with the extrinsics vector $\mathbf{z}_t \in \mathbb{R}^8$ as input, and outputs the target of the PD Controller (denoted as \mathbf{a}_t). We also augment the observation to include two additional timesteps to have the velocity and acceleration information. Formally, the base policy output $\mathbf{a}_t = \pi(\mathbf{o}_t, \mathbf{z}_t)$ where $\mathbf{o}_t = (\mathbf{q}_{t-2:t}, \mathbf{a}_{t-3:t-1}) \in \mathbb{R}^{96}$.

Reward Function. We jointly optimize the policy π and the embedding μ using PPO [158]. The reward function depends on several quantities: $\boldsymbol{\omega}$ is the object’s angular velocity. $\hat{\mathbf{k}}$ is the desired rotation axis (we use the z -axis in the hand coordinate system). \mathbf{q}_{init} is the starting robot configuration. $\boldsymbol{\tau}$ is the commanded torques at each timestep. \mathbf{v} is the object’s linear velocity. Our reward function r (subscript t omitted for simplicity) to maximize is then

$$r \doteq r_{rot} + \lambda_{pose} r_{pose} + \lambda_{linvel} r_{linvel} + \lambda_{work} r_{work} + \lambda_{torque} r_{torque} \quad (5.1)$$

where $r_{rot} \doteq \max(\min(\boldsymbol{\omega} \cdot \hat{\mathbf{k}}, r_{max}), r_{min})$ is the rotation reward, $r_{pose} \doteq -\|\mathbf{q} - \mathbf{q}_{init}\|_2^2$ is the hand pose deviation penalty, $r_{torque} \doteq -\|\boldsymbol{\tau}\|_2^2$ is the torque penalty, $r_{work} \doteq -\boldsymbol{\tau}^T \dot{\mathbf{q}}$ is the energy consumption penalty, and $r_{linvel} \doteq -\|\mathbf{v}\|_2^2$ is the object linear velocity penalty. Note that in contrast to [84] which explicitly encourages at least three fingertips to always be in contact with the object, we do not enforce any heuristic finger gaiting behaviour. Instead, a stable finger gaiting behaviour emerges from the energy constraints and the penalty on deviation from the initial pose.

Object Initialization and Dynamics Randomization. A good training environment has to provide enough variety in simulation to enable generalization in the real world. In this work we find that using cylinders with different aspect ratios and masses provides such variety. We uniformly sample different diameters and side lengths of the cylinder.

We initialize the object and the fingers in a stable precision grasp. Instead of constructing the fingertip positions as in [84], we simply randomly sample the object position, pose, and robot joint position around a canonical grasp until a stable grasp is achieved. We also randomize the mass, center of mass, and friction of these objects (see appendix for the details).

Adaptation Module Training

We cannot directly deploy the learned policy π to the real world because we do not directly observe the vector e_t and hence we cannot compute the extrinsics z_t . Instead, we estimate the extrinsics vector \hat{z}_t from the discrepancy between the proprioception history and the commanded action history via an adaptation module ϕ . This idea is inspired by recent work in locomotion [102, 97] where the proprioception history is used to estimate the terrain properties. We show that this information can also be used to estimate the object properties.

To train this network, we first collect trajectories and privileged information by executing the policy $\mathbf{a}_t = \pi(\mathbf{o}_t, \hat{\mathbf{z}}_t)$ with the predicted extrinsic vectors $\hat{\mathbf{z}}_t = \phi(\mathbf{q}_{t-k:t}, \mathbf{a}_{t-k-1:t-1})$. Meanwhile we also store the ground-truth extrinsic vector \mathbf{z}_t and construct a training set

$$\mathcal{B} = \{(\mathbf{q}_{t-k:t}^{(i)}, \mathbf{a}_{t-k-1:t-1}^{(i)}, \mathbf{z}_t^{(i)}, \hat{\mathbf{z}}_t^{(i)})\}_{i=1}^N.$$

Then we optimize ϕ by minimizing the ℓ_2 distance between \mathbf{z}_t and $\hat{\mathbf{z}}_t$ using Adam [87]. The process is iterative until the loss converges. We apply the same object initialization and dynamics randomization setting as the above section.

5.3 Experimental Setup and Implementation Details

Hardware Setup. We use an Allegro Hand from Wonik Robotics [181]. The Allegro hand is a dexterous anthropomorphic robot hand with four fingers, with each finger having four degrees of freedom. These 16 joints are controlled using position control at 20 Hz. The target position commands are converted to torque using a PD Controller ($K_p = 3.0$, $K_d = 0.1$) at 300 Hz.

Simulation Setup. We use the IsaacGym [110] simulator. During training, we use 16384 parallel environments to collection samples for training the agent. Each environment contains a simulated Allegro Hand and an cylindrical object with different shape and physical properties (the exact parameters are in the supplementary material). The simulation frequency is 120 Hz and the control frequency is 20 Hz. Each episode lasts for 400 control steps (equivalent to 20 s).

Baselines. We compare our method to the baselines listed below. We also compare with the policy with access to privileged information (*Expert*), as the upper bound of our method (Figure 5.2, top row).

1. *A Robust Policy trained with Domain Randomization (DR)*: This baseline is trained with the same reward function but without privileged information. This gives a policy which is robust, instead of adaptive, to all the shape and physical property variations [129, 162, 130, 9].
2. *Online Explicit System Identification (SysID)*: This baseline predicts the exact system parameters e_t instead of the extrinsic vector z_t during training the adaptation module.
3. *No Online Adaptation (NoAdapt)*: During deployment, the extrinsics vector \hat{z}_t is estimated at the first time step and stays frozen during the rest of the run. This is to study the importance of online adaptation enabled by the adaptation module ϕ .
4. *Action Replay (Periodic)*: We record a reference trajectory from the expert policy with privileged information and run it blindly. This is to show our policy adapts to different objects and disturbances instead of periodically executing the same action sequence.

Method	Within Training Distribution				Out-of-Distribution			
	RotR (\uparrow)	TTF (\uparrow)	ObjVel (\downarrow)	Torque (\downarrow)	RotR (\uparrow)	TTF (\uparrow)	ObjVel (\downarrow)	Torque (\downarrow)
Expert	233.71 \pm 25.24	0.85 \pm 0.01	0.28 \pm 0.05	1.24 \pm 0.19	165.07 \pm 15.65	0.71 \pm 0.04	0.42 \pm 0.06	1.24 \pm 0.16
Periodic	43.62 \pm 2.52	0.44 \pm 0.12	0.72 \pm 0.21	1.77 \pm 0.49	22.45 \pm 0.59	0.34 \pm 0.08	1.11 \pm 0.19	1.41 \pm 0.54
NoAdapt	90.89 \pm 4.85	0.65 \pm 0.07	0.44 \pm 0.11	1.34 \pm 0.12	54.50 \pm 3.91	0.51 \pm 0.06	0.63 \pm 0.13	1.34 \pm 0.11
DR	176.12 \pm 26.47	0.81 \pm 0.02	0.34 \pm 0.05	1.42 \pm 0.06	140.80 \pm 17.51	0.63 \pm 0.02	0.64 \pm 0.06	1.48 \pm 0.20
SysID	174.42 \pm 23.31	0.81 \pm 0.02	0.32 \pm 0.03	1.29 \pm 0.72	132.56 \pm 17.42	0.62 \pm 0.09	0.50 \pm 0.09	1.26 \pm 0.17
Ours	222.27 \pm 21.20	0.82 \pm 0.02	0.29 \pm 0.05	1.20 \pm 0.19	160.60 \pm 10.22	0.68 \pm 0.07	0.47 \pm 0.07	1.20 \pm 0.17

Table 5.1: We compare our method to several baselines in simulation in two settings: 1) *Within Training Distribution*; 2) *Out-of-Distribution*. Our method with online continuously adaptation achieves the best performance compared to all the baselines, closely emulating the performance of the Expert that has the privileged information as the input.

Metrics. We use the following metrics to compare the performance of our method to baselines.

1. *Time-to-Fall (TTF)*. The average length of the episode before the object falls out of the hand. This value is normalized by the maximum episode length (20s in simulation experiments and 30s in the real world experiments).
2. *Rotation Reward (RotR)*. This is the average rotation reward ($\omega \cdot \hat{k}$) of an episode in simulation. Note that we do not train with this reward. Instead, we use a clipped version of this reward during training.
3. *Radians Rotated within Episode (Rotations)*. Since angular velocity of the object is hard to accurately measure in the real world, we instead measure the net rotation of the object (in radians) achieved by the policy with respect to the world’s z-axis. This metric is only used in the real world experiments.
4. *Object’s Linear Velocity (ObjVel)*. We measure the magnitude of the linear velocity to measure the stability of the object. This is only measured in simulation. The value is scaled by 100.
5. *Torque Penalty (Torque)*. We measure the average ℓ_1 norm of commanded torque per timestep during execution to measure the energy efficiency.

5.4 Results and Analysis

In this section, we compare the performance of our method to several baselines both in simulation and in real-world deployment. We also analyze what the adaptation module learns and how it changes during policy execution and as the objects change. Finally, we focus on training a policy for rotating an object along the negative z-axis with respect to the world coordinate. We also explore the possibility of training a multi-axis policy (\pm z-axis) in the appendix. More qualitative results of

our method and several ablations of our method are in our Project Website and the supplementary.

Generalization via Adaptation

Comparison in Simulation. We first compare our method with the baselines mentioned in Section 5.3 in simulation. We evaluate all methods under two settings: 1) In the *Within Training Distribution* setting, we use the same object set and randomization setting as in RL training; 2) In the *Out-of-Distribution* setting, we use objects with a larger range of physical randomization range. We also change 20% of the objects to be spheres and cubes. We compute the average performance over 500K episodes with different parameter randomizations and initialization conditions. We report the average and standard deviation over five models trained with different seeds.

The results in Table 5.1 show that our method with online adaptation achieves the best performance compared to all the baselines. We see that adaptation to shape and dynamics of the object not only enables a better performance in training, but also gives a much better generalization to out-of-distribution object parameters compared to all the baseline methods. The *Periodic* baseline (i.e., simply playback of the expert policy) does not give a reasonable performance. Although it can rotate the exact same object with the same initial grasp and dynamic parameters, it fails to generalize beyond this very narrow setup. This baseline helps us understand the difficulty of the problem. The *NoAdapt* also performs poorly compared to our method which uses continuous online adaptation. The weaker performance of this baseline is explained by the fact that it does not update the extrinsics during the episode. This shows the importance of continuous online adaptation. The *DR* baseline, although roughly matches or performs better than the other baselines in terms of *RotR* and *TTF*, it is worse in other metrics related to object stability and energy efficiency. This is because the *DR* baseline is unaware of the underlying object properties and needs to learn a single gait, instead of an adaptive one, for all possible objects. The *SysID* baseline also performs worse than our method in both of the evaluation distributions. This is because it is harder as well as unnecessary to learn the exact values of the shape and dynamics parameters to adapt. This comparison shows the benefit of learning a low-dimensional compact representation which has a coarse relative activation for different physical properties instead of the exact value (see Figure 5.5 and Figure 5.6).

Real-World Comparisons. Next, we show real world comparisons of our policy to the baselines on two set of challenging objects shown in Figure 5.3 and Figure 5.4. We exclude the *Periodic* baseline because it does not work well even in Simulation and *Expert* because we cannot access privileged information in the real world. We evaluate each method using 20 different initial grasping positions and 6 different objects in each set. The maximum episode length is 30 s.

We first study the behavior of our policy and different baselines in a set of heavy objects (more than 100 g) including a baseball, different fruits, vegetables, and a cup (Figure 5.3). Our method performs significantly better in all metrics. Our method can achieves consistent rotation for almost all trials without falling down with an average Rotations of 23.96 (with a equivalent rotation velocity 0.8 rad s^{-1}). We find the *DR* baseline is very slow and conservative in all the trials since it is unaware of object properties and needs to learn a single gaiting behaviour for all objects. As a result, it has the lowest Rotations, although the TTF is higher than the *SysID* baseline. We also find the *DR* could perform reasonably well on objects of larger sizes because they are easier to rotate and forms

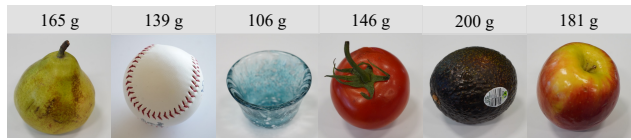
	Method	Rotations (\uparrow)	TTF (\uparrow)	Torque (\downarrow)
	DR	9.67 \pm 4.33	0.72 \pm 0.34	2.03 \pm 0.36
	SysID	10.36 \pm 2.32	0.61 \pm 0.33	1.88 \pm 0.38
	NoAdapt	N.A.	0.35 \pm 0.20	N.A.
	Ours	23.96\pm3.16	0.98\pm0.08	1.84\pm0.24

Figure 5.3: Quantitative Evaluation on a diverse set of Heavy Objects (Left). Our method which uses adaptation performs the best in terms of total rotated angle (in radians), Time To Fall (TTF), and Energy efficiency (Torque). The *DR* baseline has a conservative policy which results in a slower angular velocity. *SysID* has a more dynamic and agile policy but very unstable as can be seen in a lower TTF compared to *DR* and ours. The *NoAdapt* baseline fails on the task showing the importance of continuous online adaptation.

a reasonable chunk of training distribution. The *SysID* baseline learns a more dynamic and agile behavior, having a better Rotations metric, but a lower TTF and Rotations showing that precise estimation of system parameters is both unnecessary and difficult. We find that it is particularly hard for it to generalize to the small cup and the slightly soft tomato. Lastly, we observe a similar behavior as in [97] that the *NoAdapt* baseline does not transfer successfully to the real world, showing the importance of continuous online adaptation for successful real-world deployment.

We perform the same comparisons on a collection of irregular objects (Figure 5.4). It contains a container with moving COM, objects with concavity, a cylindrical kiwi fruit, a shuttlecock, a toy with holes, and a cube toy. Rotating the cube is particularly difficult because we only rely on usage of fingertips. Although these variations are challenging for this task and are beyond what is seen during training, our method still performs reasonably well, outperforming all the baselines. We see that the *DR* baseline can perform stable but slow in-hand rotation for the container and the shuttlecock, but largely fails for the other objects, indicating its difficulty in shape generalization. For the *SysID* baseline, its stability is significantly lower than the *DR* baseline as well as our method, despite having a higher angular velocity. The *NoAdapt* baseline performs similarly to what we observed in Figure 5.3.

Understanding and Analysis

To understand how our method generalizes to a diverse set of objects, we design a few experiments and visualization to help develop some insights.

Extrinsics over Time. We run one continuous evaluation episode in the real world in which we replace the object in the hand every 30s for 6 objects. Note that during training, we never randomize the objects within an episode. During the entire run, we record the estimated extrinsics and plot 2 out of the 8-dim extrinsics vector in Figure 5.5. The top plot shows the extrinsic value $z_{t,0}$ which responds to changes in object diameter. It has a lower value for smaller diameters and higher value for larger diameters. The bottom plot shows that the extrinsic value $z_{t,2}$ responds to variations in


	Method	Rotations (\uparrow)	TTF (\uparrow)	Torque (\downarrow)
	DR	6.59 ± 3.71	0.66 ± 0.41	1.85 ± 0.37
	SysID	8.16 ± 3.39	0.46 ± 0.36	1.70 ± 0.40
	NoAdapt	N.A.	0.12 ± 0.05	N.A.
	Ours	19.22 ± 4.08	0.78 ± 0.27	1.48 ± 0.30

Figure 5.4: Quantitative Evaluation on a diverse set of Irregular Objects (Left). Our method can successfully generalize to rotating a diverse set of objects including objects with holes, soft and deformable objects (none of these were included in the training). Our method outperforms the baselines on all the metrics. The *DR* baseline has the second highest TTF but low Rotations because it outputs very conservative and slow trajectories. *SysID* achieves slightly faster but a very unstable policy. The superior performance of our method over baselines shows the importance of adaptation via a low dimensional extrinsics estimation for generalization in this task.

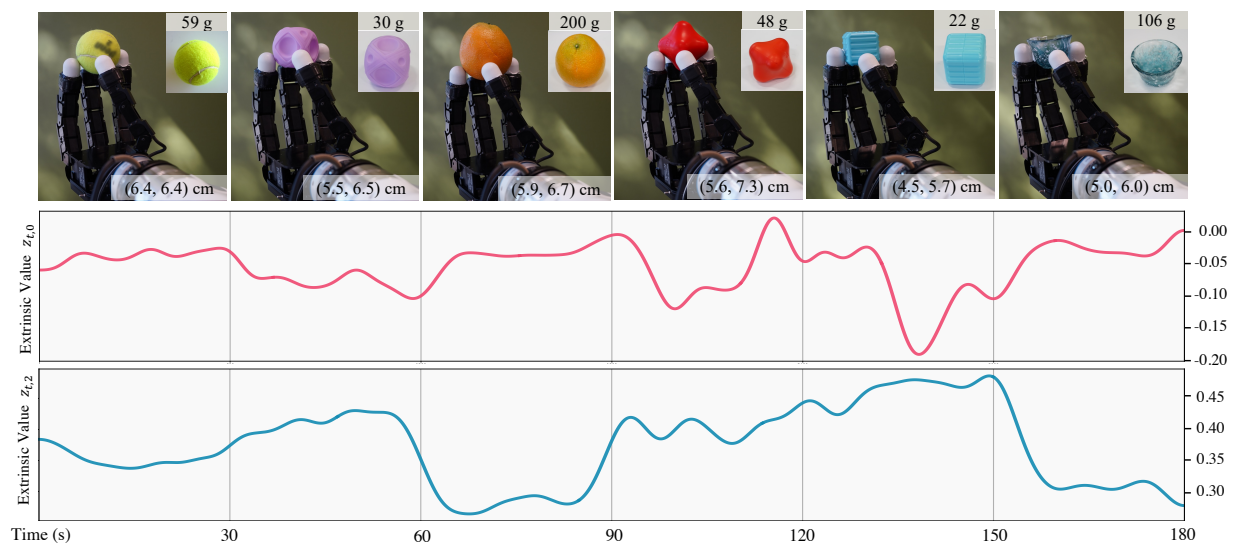


Figure 5.5: Two of the 8-dim estimated extrinsics vector during one continuous run in which we change the object in the hand every 30s for 6 objects. The top plot shows the extrinsic value $z_{t,0}$ which responds to changes in object diameter with smaller diameters leading to lower values. Since the perceived diameter of the object changes during rotating an irregular shaped object, we see variations in this extrinsic value even within the same object. The bottom plot shows the correlation between $z_{t,2}$ and object mass: it attains higher values for lighter objects and lower for heavier objects. See this link for the video.

object mass.

Extrinsic Clustering. Another way to understand the estimated extrinsics \hat{z} is by clustering the extrinsics vector estimated while rotating different objects. In Figure 5.6, we visualize the estimated

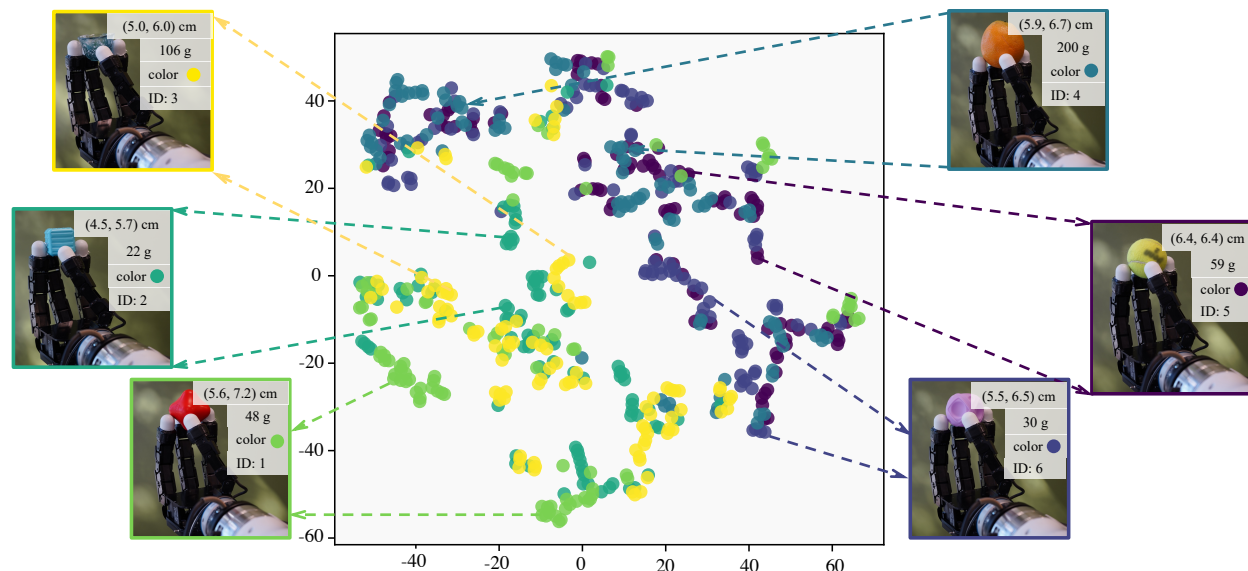


Figure 5.6: We visualize the estimated extrinsics vector \hat{z}_t using t-SNE for rotating 6 different objects for 5 seconds each. We found that objects of different sizes and different weights tend to occupy separate regions. For example, \hat{z} corresponding to smaller objects tend to cluster on the bottom left. The object with ID 1 (shown in the bottom left in the above figure) produces scattered \hat{z} because of its irregular shape, which results in an equivalently varying scale of the object. This locality leads to the scattering of the extrinsics across the t-SNE plot, and this locality is also what gives us generalization as evaluated in Table 5.1, Figure 5.3 and Figure 5.4.

extrinsics vector using t-SNE for rotating 6 different objects. We find that objects of different sizes and different weights tend to occupy separate regions as explained in Figure 5.6.

Emergent Finger Gaits. We find it important to use cylindrical objects for the emergence of a stable and high-clearance gait. On our website, we compare the learned finger gaits when training with cylindrical objects and with pure spherical objects. The latter training scheme leads to a policy with a dynamic gait which works well on balls but fail to generalize to more complex objects.

Real World Qualitative Results

On our Project Website, we qualitatively compare our method with the baselines on 5 different objects and further evaluate our method on 30 objects of different shapes and physical properties, including porous objects and non-rigid objects. The diameter ranges from 4.5 cm to 7.5 cm and the mass ranges from 5 g to 200 g. Note that we only use cylindrical objects of different sizes and aspect ratios during training and show shape generalization in the real world with the use of rapid adaptation. The key insight which allows this generalization is that the shape of the objects as perceived by the fingertips of the hand can be compressed to a low dimensional space. We estimate this low dimensional space from proprioceptive history, which allows us to generalize to objects

which are seemingly different in the real world but might look similar in the extrinsics space.

5.5 Summary

There are different levels of difficulty for general dexterous in-hand manipulation. The task considered in this chapter (in-hand object rotation over the z -axis) is a simplification of the general $SO(3)$ reorientation problem. However, it is not a limiting simplification. With three policies for rotation along three principle axes, we can achieve rotating the object to any target pose.

We show the feasibility of purely proprioceptive in-hand object rotation for a diverse set of objects via rapid adaptation. We find most of our failure cases in the real-world experiments are due to incorrect contact points which leads to unstable force closure. Since our method only relies on the proprioceptive sensing, it is unaware of the precise contact positions between the objects and the fingertips. Another failure case is when the object to be rotated is tiny (with diameter less than 4.0 cm). In this case the fingers will frequently collide with each other, making the robot not able to maintain the grasp balance of the object. More extreme and sophisticated shapes are also harder to manipulate. For those more challenging tasks, it might be necessary to incorporate tactile or visual feedback.

Chapter 6

RMA for Bipedal Locomotion

In this chapter, we study bipedal robots which are more sensitive to mismatches in simulation and real. We find that the RMA algorithm, when applied as is, achieves a suboptimal performance due to errors in learning the adaptation module. This makes it challenging to deploy it in the real world. In contrast, several existing control theoretic methods have shown quite promising results in the real world [176, 94, 142, 186, 46, 35, 58, 144, 32, 56, 57, 103, 150, 36]. However, designing these controllers often requires expert parameter tuning for different terrain. Learning based methods can get around this with the use of data and simulation.

In this chapter, our goal is to see how well a learning-based approach works for bipedal robots. Unlike their quadruped or hexapod counterparts, bipeds are more dynamic and offer coverage of more terrains at the cost of being more prone to instability. Although there have been several investigations of applying learning methods to bipedal robots [104, 160], until now, the robustness performance has not matched their quadruped counterparts in terms of robustness to scenarios not seen during training (Figure 6.1).

We train our robot in simulation and start with Rapid Motor Adaptation (RMA) [96] as a baseline for sim2real transfer to the Cassie biped. In contrast to the popular idea of training a terrain-invariant policy [171], RMA trains an adaptive policy conditioned on latent *extrinsics* vector that encodes terrain-specific information in simulation. After training this base policy, the extrinsic vector is then estimated online by an adaptation module using the history of observed proprioceptive states. However, we found that scaling RMA as-is to complex bipeds is faced with several challenges.

The first challenge is training the base policy from scratch using reinforcement learning. To address this issue and learn naturally stable high-performance gaits, we bootstrap the base policy from a set of reference motions generated using a gait library. The second challenge, however, is a more fundamental one. For complex robots like bipeds, it is usually very challenging to precisely estimate the privileged extrinsics at deployment just from the observable states. This creates a large domain gap for the base policy at deployment which was trained with accurate extrinsics. The higher the magnitude of extrinsics estimation error, the worse the base policy performs. We bridge

This chapter is based on joint work with Zhongyu Li, Jun Zeng, Deepak Pathak, Koushil Sreenath, Jitendra Malik [95]

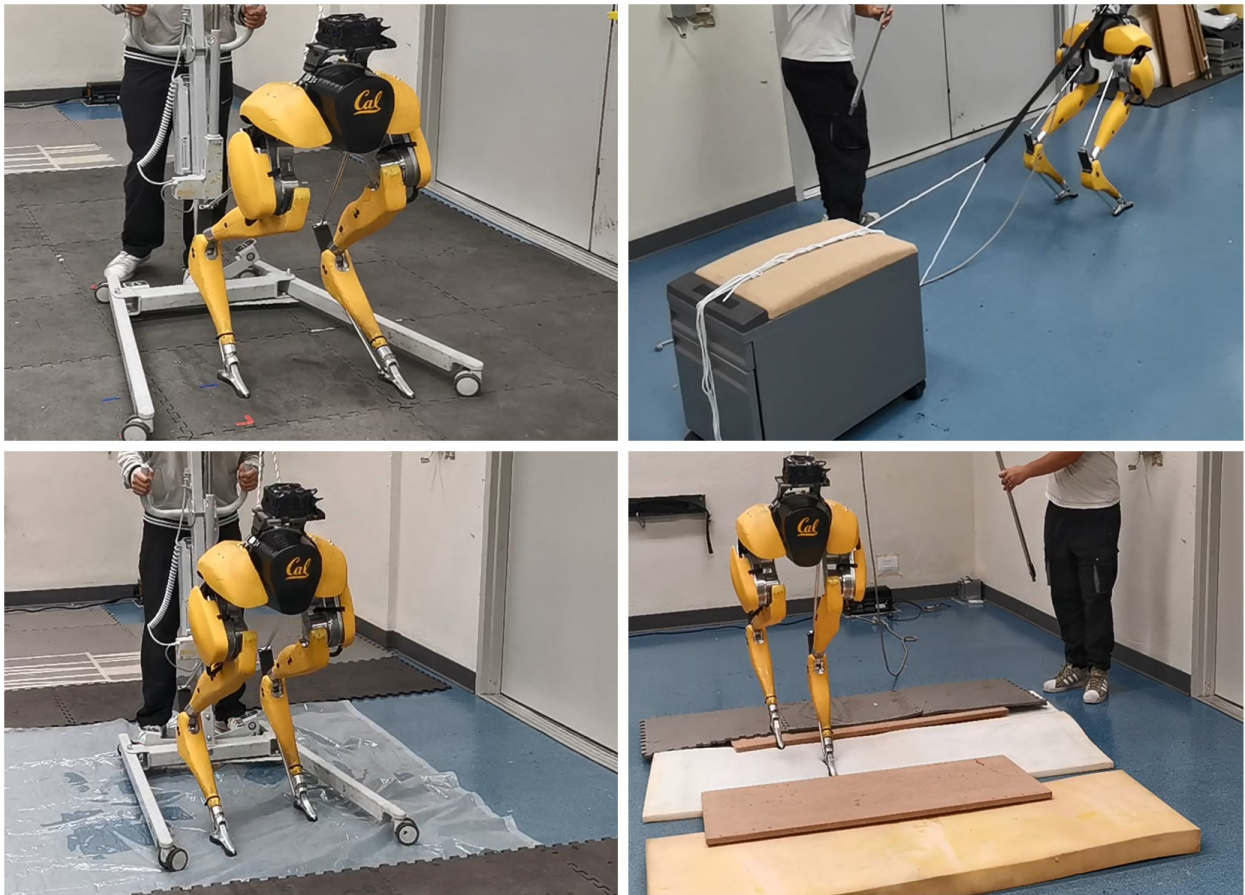


Figure 6.1: We demonstrate the performance of A-RMA with a bipedal robot in several challenging setups which includes slippery surfaces, foam, and pulling a payload. Some of these were never seen during training and the policy was deployed without any calibration or real-world finetuning. Note that bipeds are inherently more unstable compared to quadrupeds, which makes each of these much more challenging than for a quadruped robot. Videos at <https://ashish-kmr.github.io/a-rma/>

this gap by another round of finetuning the base policy using the imperfect extrinsics estimated from adaptation module instead of conditioning on the perfect extrinsics. We call our method *A-RMA* due to the adaptation of RMA policy itself, as outlined in Figure 6.2. Similar to the base policy and adaptation module in RMA, *A-RMA policy adaptation* step is also trained in simulation. We evaluate A-RMA on a complex Cassie bipedal robot and show evaluations on several challenging environments in both simulation and real world.

6.1 Related Work

Model-based Control for Bipedal Robots Locomotion of bipedal robots has been traditionally approached via notions of gait stability such as ZMP criterion [176] or Capturability [142] using robot’s reduced-order models [94, 186, 105, 46, 35, 56]. Although these methods can effectively control humanoid robots with flat feet, they often walk conservatively. Alternatively, Hybrid Zero Dynamics (HZD) [58, 32, 57, 103, 150] based techniques can also generate stable periodic walking based on input-output linearization using robot’s full-order model. However, HZD-based controllers for 3D robots typically need extensive parameter tuning in both simulation and in the real world, and are hard to adapt to the environment changes. In this work, we use reinforcement learning to learn our walking controllers, using the HZD-based walking gaits only as reference motions to produce natural looking gaits. This allows us to generate more diverse and effective stable walking behaviours since we don’t have hard constraints on periodicity and neither do we require a precise model.

RL-based Control for Legged Robots Reinforcement learning for legged locomotion has shown promising results in learning walking controllers that can be successfully deployed in the real world [104, 96, 90, 72, 136]. Data-driven methods could either use reference motions [136, 104] to learn walking behaviours, learn residuals over predefined foot motions [102, 75], or learn without any motion priors and foot trajectory generators [96, 63, 72]. In this work, we use a hybrid approach where we warm start the learning process using a HZD-based gait library, but subsequently reduce the dependency on them by lowering the costs related to imitating the gait library.

Recent works have also built bipedal controllers using RL. Model-based RL in [22, 21] is used to learn a walking controller on Cassie in simulation to track a velocity. Alternatively, work in [184, 185] learns model free residuals over reference motions, while work in [79] learns walking policies that can track a commanded planar velocity on Cassie in the real world. Although residual control speeds up training, the corrections that can be applied to the reference trajectory is limited which limits the diversity of behaviours of the controller. This can be addressed by learning policies which don’t rely on reference motions during deployment, but instead add motion constraints during training by either using reference motions or through foot fall constraints via reward terms [104, 159, 161]. We follow this general approach in this chapter. More specifically, we use reference trajectories for imitation via reward terms, except that we only use them to warm start the learning process and then reduce the dependency on them and use energy minimization [50] to allow for diverse behaviors which can be more general than the reference motions.

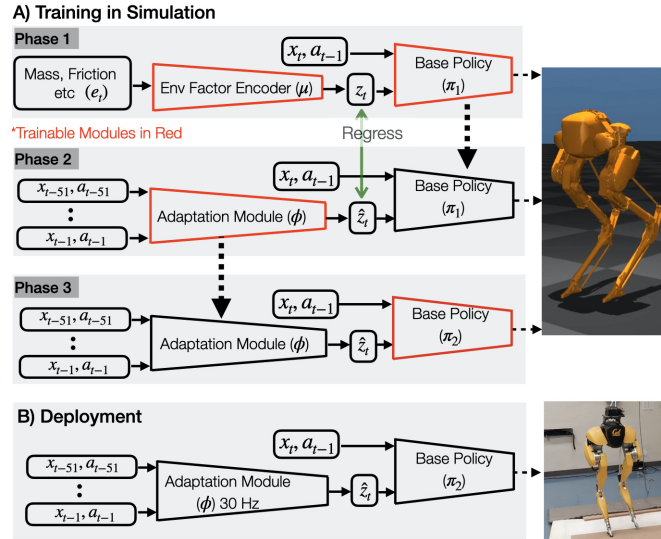


Figure 6.2: We show the training and deployment phases of A-RMA. The first two phases are the same as RMA [96]. We additionally add a third phase in which the base policy is fine-tuned again with PPO while keeping the adaptation module fixed, to account for imperfect estimation of extrinsics. We found this to be critical for reliable performance in the real world.

Simulation to Real World Transfer Sim-to-real methods allow deployment of walking policies in the real world after training them in simulation, which is a safe and inexpensive data source. Parameters finetuning on the hardware is one way to bridge the gap for model-based methods [57, 103, 188]. Alternatively, several works use domain randomization which varies the system properties in simulation in order to cover the uncertainty in the real world [154, 171, 137, 199, 195, 136, 104, 160, 159, 161]. These methods achieve robustness at the cost of optimality, where they try to learn a single environment-agnostic behaviour for all deployment scenarios. Instead, we learn an adaptive policy which enables transfer to widely varying deployment scenarios.

System Identification and Adaptation An adaptive policy conditions to environment variations instead of being agnostic to them. These variations can be explicitly estimated during deployment either through a module that is trained in simulation [196] or can be estimated by optimizing for high returns using evolutionary algorithms [193]. Predicting the exact parameters is both unnecessary and difficult, which in turn leads to poor empirical performance. Instead, a low dimensional latent embedding can be used [96, 136, 201] which contains an implicit estimate of the environment. At test time, this latent embedding can be optimized using policy gradients from real-world rollouts [136], Bayesian optimization [197], or random search [195]. An alternate approach is to use meta learning to learn to adapt online [47]. These methods tend to take multiple rollouts to adapt in the real world [165, 28] which is prohibitive for several scenarios. In this work, we follow the approach of [96] which learns a feed forward policy to estimate the latent environment vector, enabling rapid adaptation in fractions of a second.

6.2 General Walking Controller

Our walking controller contains a base policy which produces the target joint points for the robot, and an adaptation module which uses proprioceptive history to continually estimate the extrinsics vector. Such a vector contains information about the environment that can be used by the base policy to adapt. In this section, we describe how we train the base policy for bipeds since trivially training the policy from scratch results in unnatural gaits, unlike prior work in quadrupeds [96]. Instead we use reference motions to bootstrap learning but unlike prior methods which use reference motions [102, 104], we reduce the dependence on these reference motions as training progresses to learn optimal behaviors in a data driven way.

Walking Policy

We train an adaptive walking policy (π_1) using model free reinforcement learning to learn a stable walking policy while imitating reference motions generated using a HZD-based gait library. Here, the gait library consists of a collection of periodic walking gaits at different walking velocities and walking heights. The base walking policy (π) takes two arguments as input 1) the current state x_t , 2) the extrinsics vector $z_t \in \mathbb{R}^8$ and then predicts the next action a_t . The predicted action a_t is the target position for the 10 actuated robot joints which is converted to torque using a PD controller. The extrinsics vector z_t is a compressed version of the environment vector $e_t \in \mathbb{R}^{147}$ generated by μ . The environment vector e_t includes things such as friction, inertia, center of mass, etc (see Sec III-C), but the extrinsics vector z_t only retains as much information from e_t as is necessary for the base policy to adapt. We thus have,

$$z_t = \mu(e_t), \quad (6.1)$$

$$a_t = \pi_1(x_t, a_{t-1}, g_t, z_t). \quad (6.2)$$

We implement μ and π_1 as MLPs and jointly train them end-to-end using model-free reinforcement learning to maximize the following expected return of the policy π :

$$J(\pi) = \mathbb{E}_{\tau \sim p(\tau|\pi)} \left[\sum_{t=0}^{T-1} \gamma^t r_t \right],$$

where $\tau = \{(x_0, a_0, r_0), (x_1, a_1, r_1) \dots\}$ is the trajectory of the agent when executing policy π_1 , and $p(\tau|\pi_1)$ represents the likelihood of the trajectory under π_1 .

Reward Function

The reward function encourages the agent to track the goal commanded and to imitate the reference motion from the gait library. Let's denote the robot's actual motor positions as \hat{q}_m , pelvis translation position and velocity as \hat{q}_p and $\dot{\hat{q}}_p$, the pelvis rotation and rotation velocity as \hat{q}_r and $\dot{\hat{q}}_r$. Let the corresponding quantities for the reference motion be denoted by q_m^r , q_p^r , \dot{q}_p^r , q_r^r and \dot{q}_r^r respectively.

We additionally denote the robot’s torque as u and the robot’s ground reaction force as F . The reward at each time step t is a weighted sum of:

1. Motor imitation: $\exp[-\rho_1 \|q_m^r - \hat{q}_m\|_2^2]$
2. Pelvis position imitation: $\exp[-\rho_2 \|q_p^r - \hat{q}_p\|_2^2]$
3. Pelvis velocity imitation: $\exp[-\rho_3 \|\dot{q}_p^r - \hat{\dot{q}}_p\|_2^2]$
4. Pelvis rotation imitation: $\exp[-\rho_4 (1 - \cos(q_r^r - \hat{q}_r))]$
5. Rotational velocity imitation: $\exp[-\rho_5 \|\dot{q}_r^r - \hat{\dot{q}}_r\|_2^2]$
6. Torque penalty: $\exp[-\rho_6 \|u\|_2^2]$
7. Ground reaction force penalty: $\exp[-\rho_7 \|F\|_2^2]$,

where the corresponding weights of each of the reward terms are $[0.3, 0.24, 0.15, 0.13, 0.06, 0.06, 0.06]^T$, and ρ_1 to ρ_7 are $[5.0, 0.1, 5.0, 5.0, 1.0, 5e^{-7}, 1.25e^{-5}]$. To stabilize the pelvis, we set the desired roll and pitch velocity to 0, and the desired yaw velocity is from the user command $c(t)$. We compute the desired pelvis translational and rotational positions by integrating the corresponding desired velocity. Moreover, we decay the coefficients of the imitation terms as the training progresses to reduce the dependence on reference motions and learn optimal data driven behaviours.

Environment Variations and Terrains

During the learning of the walking policy, we vary the ground friction, robot’s motor friction, mass and center of mass of the robot and robot links. We train over fractal terrain, flat terrain, slopes and discrete terrains such as steps (see Table 6.1).

6.3 Walking Controller with Rapid Adaptation

Following the process described in the previous section, we now have a base policy which can use the extrinsics vector to adapt. However, this extrinsics vector uses privileged simulation information and is unavailable in the real world. To resolve this, we generalize the idea proposed in [96] of learning an adaptation module using the history of commanded actions and robots proprioception to estimate these extrinsics online. The insight is that the discrepancy between what was commanded to the robot and the actual joint positions contains information about these extrinsics. However, such an estimation could be noisy as the extrinsics vector might not be fully observable. To fix this, we further finetune the base policy to learn to walk adaptively with an imperfect extrinsics vector. We now describe this in detail below.

Adaptation Module for Estimating Extrinsic

Privileged environment information e_t is not available in the real world, and consequently its encoded extrinsics vector z_t is not accessible during deployment. Similar to [96], we estimate the extrinsics online from the proprioceptive history $(x_{t-k:t-1}, a_{t-k:t-1})$ using the adaptation module ϕ . In our experiments, we use $k = 70$ which roughly corresponds to 2s. The estimated extrinsic vector is thus given by,

$$\hat{z}_t = \phi(x_{t-k:t-1}, a_{t-k:t-1}).$$

We can train the adaptation module in simulation via supervised learning to minimize: $\text{MSE}(\hat{z}_t, z_t) = \|\hat{z}_t - z_t\|^2$, where $z_t = \mu(e_t)$. We model ϕ as a 1-D CNN to capture temporal correlations.

We collect trajectories by unrolling the base policy π_1 with the \hat{z}_t predicted by the randomly initialized function ϕ , and then pair it with the ground truth z_t to train ϕ . We iteratively repeat this until convergence.

Finetuning with Estimated Extrinsic

For our setting, the extrinsics estimated by the adaptation module ϕ is imperfect since z_t might not be fully observable from the proprioception history. We observed this as the regression error to estimate z_t did not become low enough during the training of the adaptation module. This causes the base policy π_1 to experience a significant drop in performance since it is trained with perfect extrinsics, and noisy estimation introduces a domain gap. To overcome this issue, we propose to further finetune the base policy π_1 with the imperfect extrinsics predicted by the adaptation module ϕ trained in phase 2, see Fig. 6.2. The adaptation module is kept frozen and the base policy is finetuned using model free reinforcement learning with the reward described in Section 6.2. This gives us the final base policy π_2 which is used in deployment.

6.4 Experimental Setup

Hardware

We use the Cassie robot for our experiments. It is a dynamic, life-sized, and underactuated bipedal robot which has 20 DoFs and is introduced in details in [103, Sec. II]. It has 10 actuated rotational joints $q_m = [q_{1,2,3,4,7}]^T$ (abduction, rotation, hip pitch, knee, and toe motors), and four passive joints $q_{5,6}^{L/R}$ (shin and tarsus joints). Its floating base, the robot pelvis $q_p = [q_x, q_y, q_z, q_\psi, q_\theta, q_\phi]^T$, has 6 DoFs for sagittal q_x , lateral q_y , vertical q_z , roll q_ψ , pitch q_θ , and yaw q_ϕ , respectively. The entire robot coordinate $q \in \mathbb{R}^{20}$ includes all robot DoFs whereas the observable DoFs $q^o \in \mathbb{R}^{17}$ excludes the pelvis translational position $q_{x,y,z}$ which can not be reliably obtained by the sensors on the robot hardware. There is an onboard Intel NUC on the robot, which runs our policy. The policy updates at 30 Hz and the joint-level PD controller operates at 2 kHz.

Parameter	Range	Unit
Link Mass	$[0.7, 1.3] \times \text{default}$	kg
Link Mass Center	$[0.7, 1.3] \times \text{default}$	m
Joint Damping	$[0.3, 4.0] \times \text{default}$	Nms/rad
Spring Stiffness	$[0.95, 1.05] \times \text{default}$	Nm/rad
Ground Friction Ratio	$[0.3, 3.0]$	1
Fractal Terrain Height	$[0.0, 0.12]$	m

Table 6.1: The range of dynamics parameters that are extended from [104, Table II] and are varied in the phase 1 of A-RMA.

Simulation Setup

We use the simulation environment for reinforcement learning on Cassie developed in [104], which itself is based on an open source MuJoCo simulator [172, 4]. During training, each episode has a maximum number of time steps of $T=2500$ (83 s). In each episode, we resample a new command $c(t) = [\dot{q}_x^d, \dot{q}_y^d, q_z^d, \dot{q}_\phi^d]^T$ and randomize the environment parameters every 8 s. The randomization range of the command is between $[-1.0, -0.3, 0.65, -30^\circ]^T$ and $[1.0, 0.3, 1.0, 30^\circ]^T$. The episode will terminate early if the pelvis height falls below 0.55 m, or if the tarsus joints $q_6^{L/R}$ hit the ground [104].

State-Action Space

Observation The observation $\mathbf{x}_t = (\mathbf{q}_{t-4:t}^o, \mathbf{a}_{t-4:t-1}, \mathbf{q}_{m,t}^r, \mathbf{q}_{m,t+1}^r, \mathbf{q}_{m,t+4}^r, \mathbf{q}_{m,t+7}^r, c_t)$ at time t consists of 4 parts. There are observable robot states $\mathbf{q}^o = [q^o, \dot{q}^o]$ at the current and the past 4 time steps, and the actions \mathbf{a} from the past 4 time steps. There is also reference motion $\mathbf{q}_m^r = [q_m^r, \dot{q}_m^r]$ which includes the reference motor position and motor velocity at current time t and future time steps (next 1, 4, 7 time steps) obtained from a HZD-based gait library [104]. Moreover, the command c_t is also part of the observation. c_t includes desired sagittal and lateral walking speed, walking height, and turning yaw rate ($c_t = [\dot{q}_x^d, \dot{q}_y^d, q_z^d, \dot{q}_\phi^d]$).

Action The action $\mathbf{a}_t = q_m^d$ is the target position for the 10 actuated motors on Cassie which are first passed through a low-pass filter [104] before being sent to the PD controller to generate the torque u for each motor.

Environment and Terrain variations

We randomize the environment parameters that includes robot modelling errors, sensor noise and delay, and terrain variations during the training. Based on previous work [104], the range of the randomization that is newly introduced is presented in Table. 6.1. Furthermore, we also introduce fractal-like terrain variations to make the policies robust.

	Min Friction	Feasible Cmd Range	Tracking Error
HZD Controller [103]	0.3	39/147	[0.1554, 0.0811]
Robust MLP [104]	0.3	147/147	[0.0869, 0.1150]
RMA [96]	0.2	147/147	[0.0908, 0.0965]
A-RMA (Ours)	0.2	147/147	[0.0849, 0.0952]

Table 6.2: **Comparison of A-RMA with baseline controllers in MATLAB Simulink.** We observe that all RL-controllers can track the entire command range. Of these, A-RMA and RMA generalize beyond the training friction range, maintaining stability in as low as a 0.2 friction coefficient. A-RMA has the best tracking performance in both $[\dot{q}_x^d, \dot{q}_y^d]$ compared to all the baseline controllers.

Training Details

Base Policy and Environment Encoder The base policy (π) is a MLP with 2 hidden layers of size 512 each. The environment encoder (μ) has 1 hidden layer of size 256, and output (z) size of 8. We train the policy with PPO using a batch size of 65536 and minibatch size of 8192.

Adaptation Module The adaptation module (ϕ) contains 3 convolution layers (kernels sizes 8, 5, 5 and strides 4, 1, 1 respectively) followed by 1 hidden layer of size 256. The output layer (\hat{z}) is 8 dimensional. We train the policy with Adam optimizer using the same batch size, for a total of 2000 iterations, resampling a new batch in every iteration.

Base Policy Finetuning Stage In the last finetuning stage of A-RMA, we freeze the adaptation module and finetune the base policy using PPO with a batch size of 65536 and minibatch size of 8192 for 2000 iterations.

6.5 Results and Analysis

We first validate A-RMA in both MuJoCo and MATLAB Simulink (a high-fidelity simulator), and then present the results of RMA deployed on a bipedal robot Cassie in the real world. All A-RMA results in simulation and the real world are from the same policy which was only trained in MuJoCo and was deployed in MATLAB Simulink and the real world without any finetuning during test time. In this chapter, we focus more on the control performance with a nominal walking height $q_z = 0.98$ m, *i.e.*, we do not change the walking height q_z during the test though the policy is trained with variable walking height.

	MTTF (s)	Return	Tracking Err	Mean Jerk
A-RMA-static	7.1	158.5	0.79	0.64
RMA [96]	11.4	273.6	0.29	0.92
A-RMA	14.0	335.8	0.30	0.30
A-RMA-priv	14.2	340.0	0.27	0.28

Table 6.3: **Comparison of A-RMA with baseline controllers in MuJoCo.** We observe that A-RMA’s performance is very close to A-RMA-priv, which has access to privileged simulation information, and is better than all the baseline controllers for the metrics: MTTF (max episode length = 20s), Returns and Mean Jerk experienced. The tracking performance (Mean Tracking Error) is similar to RMA. Note that RMA experiences a large drop compared to A-RMA-priv because of unobservability of the entire extrinsics vector from proprioception. A-RMA-static sees a sharp drop in performance compared to A-RMA, validating the importance of a continually and rapidly updating extrinsics vector.

Simulation Validation

The Simulink simulator provides us a safe validation environment to compare locomotion controllers while providing very high fidelity dynamics which is much more accurate than MuJoCo. We additionally show comparisons of A-RMA to its other adaptive variants in MuJoCo. Please note that the agent has no access to the Simulink data during the policy training in MuJoCo. We benchmark the A-RMA with the following locomotion controllers on Cassie as baselines: 1) a HZD-based controller (HZD) developed in [103] based on [57], 2) a robust RL-based locomotion controller represented by MLP (Robust MLP) developed in [104], 3) A-RMA-static in which we estimate the latent in the first time step and then freeze it for the rest of the episode, 4) RMA [96]. Finally as oracle, we also show the performance of A-RMA-priv which has access to privileged simulation information for reference. Please note that, for all the RL-based controllers, we utilize the same formulation of rewards and range of dynamics randomization presented in Sec. 6.4 during training.

We benchmark these controllers using the following metrics: 1) the converged return, 2) command tracking performance on a nominal ground, 3) Minimum friction successfully handled, 4) Mean time to Fall (MTTF), 5) Mean Jerk of all the joints. In MuJoCo, the reported metrics are computed over 10 episodes with an episodic timeout of 20s during which we randomize all the parameters shows in Table 6.1.

Converged Return and MTTF We report the performance of A-RMA-priv to understand the maximum achievable performance. As shown in Table 6.3, going from A-RMA-priv to RMA shows a sharp performance fall in both the metrics. This sharp fall is a consequence of non-negligible phase 2 regression loss as the extrinsics vector might not be fully observable from the proprioception history. We show that the proposed A-RMA recovers this performance drop and

approximately matches the performance of the A-RMA-priv. We additionally see that A-RMA-static has a substantial performance fall indicating the importance of rapid online adaptation via estimated extrinsics.

Command Tracking We compare the different locomotion controllers on command tracking performance where we sample different desired sagittal walking speed \dot{q}_x^d and lateral walking speed \dot{q}_y^d . The command tracking performance includes two parts: the range of the command for which a controller is able to maintain gait stability (Feasible Command Set [104]), and tracking error between the desired and actual robot walking velocities.

We test the range of command $[\dot{q}_x^d, \dot{q}_y^d]^T$ from $[-1.0, -0.3]^T$ to $[1.0, 0.3]^T$ with a resolution of $[0.1, 0.1]^T$ with a nominal walking height $q_z = 0.98$ on Cassie with different locomotion controllers in simulation. These count for 147 different commands and a command is considered feasible if the controller can maintain stability for 10 seconds. As demonstrated earlier in [104], HZD controller is only able to stably track a limited number of commands at nominal walking height. Specifically, the lateral walking speed cannot exceed 0.1 m/s while walking forwards, and there are only 39/147 feasible commands using the HZD controller. However, all of the RL-based controllers can cover the entire range of the testing commands, *i.e.*, there are 147/147 feasible commands using locomotion controllers using Robust MLP, RMA, and A-RMA. In terms of the tracking errors, as shown in Fig. 6.3, A-RMA manifests the least tracking errors with the only $[0.0849, 0.0952]^T \text{ m/s}$ compared to other RL-based controllers (Table. 6.2 and Table. 6.3). Moreover, as seen in Fig. 6.3, RMA and A-RMA produce less oscillations than Robust MLP. A-RMA has an overall superior performance in command tracking.

Mean Jerk We measure the jerk experienced in the motors in MuJoCo for the baseline controllers. We observe in Table 6.3 that A-RMA almost matches the smoothness of the A-RMA-priv, while RMA and A-RMA-static seem to do much worse. The additional phase 3 in A-RMA allows the base policy to learn smooth behaviours while accounting for the imperfect extrinsics estimator.

Slippery Ground We use MATLAB Simulink to quantify the ability of locomotion controller to maintain gait stability on slippery ground. Fig. 6.4 shows that both HZD and Robust MLP fail to maintain robot balance when the ground friction coefficient is set to 0.2 while RMA and A-RMA succeed to control the robot to walk forwards and backwards. We do a line search on friction values and observe that the minimal ground friction for which HZD and Robust MLP can maintain stability is 0.3 while RMA and A-RMA can go down to 0.2. This is significant because bipedal robots are inherently unstable and become very hard to control as the coefficient of friction approaches zero. Note that the RL-controllers were not trained for friction value of 0.2 (see Table 6.1), demonstrating that RMA and A-RMA have better generalization at test time than Robust MLP. Furthermore, as shown in Fig. 6.4d, Cassie does not deviate a lot in the lateral direction (with command $\dot{q}_y^d = 0$) when it walks backwards using A-RMA, while there exists a large drift to the right for RMA in Fig. 6.4c. This showcases A-RMA’s superior tracking performance even on a slippery ground.

Real World Experiments

We deploy A-RMA on a Cassie bipedal robot in the real world, and test in four scenarios: 1) tracking variable commands on a nominal ground, 2) robot walking while towing a heavy load with cable, 3) slippery ground, and 4) rough terrain with variable softness. We use the same policy for all our experiments which was trained in MuJoCo simulation and is deployed without any finetuning or calibration. The results are demonstrated in Fig. 6.5. Videos are available at [41] and included in the video attachment.

As shown in Fig. 6.5a, 6.5b, Cassie is able to track varying commands including forwards, backwards, and sideways walking speed while maintaining gait stability and low ground impacts.

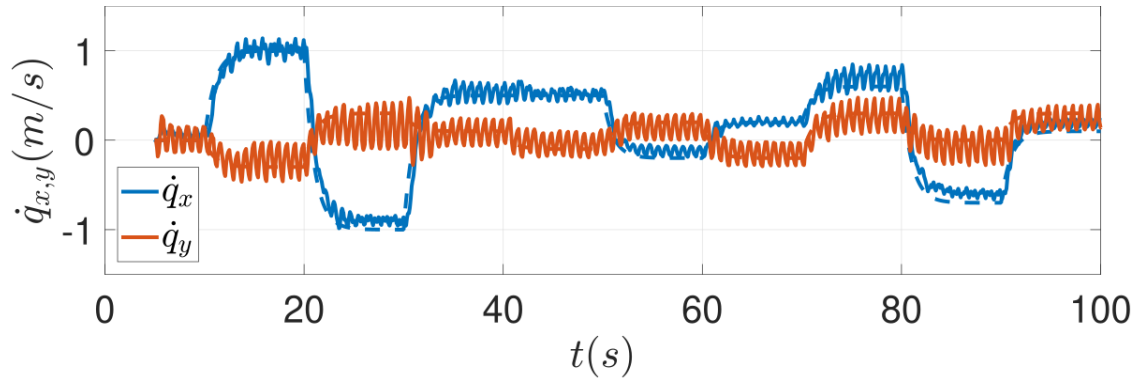
In the load carrying task, Cassie controlled by A-RMA is able to tow the load (around 40 kg with wheels) while maintaining forward walking speed with no significant drift to other directions, as shown in Fig. 6.5c. During the tests, the robot is able to stay robust to the pulling force from the cable which fluctuates depending on the speed of the payload and the tension in the string making it hard to model. Despite, slack/taut changes happening during the trials, the robot is able to maintain stability against such hybrid mode switches.

During the slippery ground test, we cover the ground by a plastic sheet with water in between in order to reduce the ground friction coefficient, as shown in Fig. 6.5d. When the robot steps onto the plastic sheet, there are significant slips between the robot feet and ground, *i.e.*, unexpected contact changes which makes controlling a life-sized robot with such low ground traction very challenging. However, the proposed policy is able to adapt to the changes in ground friction and contacts, and therefore able to maintain gait stability on such a slippery ground while tracking varying commands in both forward and lateral directions.

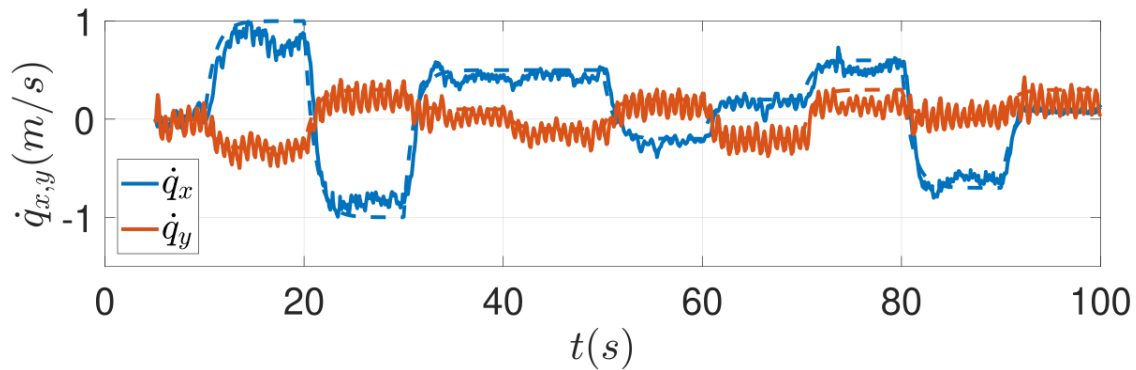
We also introduce challenging types of terrains and contacts in the tests presented in Fig. 6.5e, where we randomly place soft foams and wooden planks on the ground and let the robot walk onto that region. Such random changes of softness on the ground will change the contact type between the robot feet and ground: the contact region is on the sole when robot steps on a rigid plank while the contact can happen anywhere on the robot foot if it step into the soft foam, which making the contact very hard to model. Cassie, using A-RMA, is not only able to step on the rigid plank but also onto the soft foams without losing balance. Please note that the softness of the ground is not randomized during training as simulating a soft contact with high fidelity is still an open question. However, A-RMA is still able to generalize to the softness changes in the real world despite never having seen it during training.

6.6 Summary

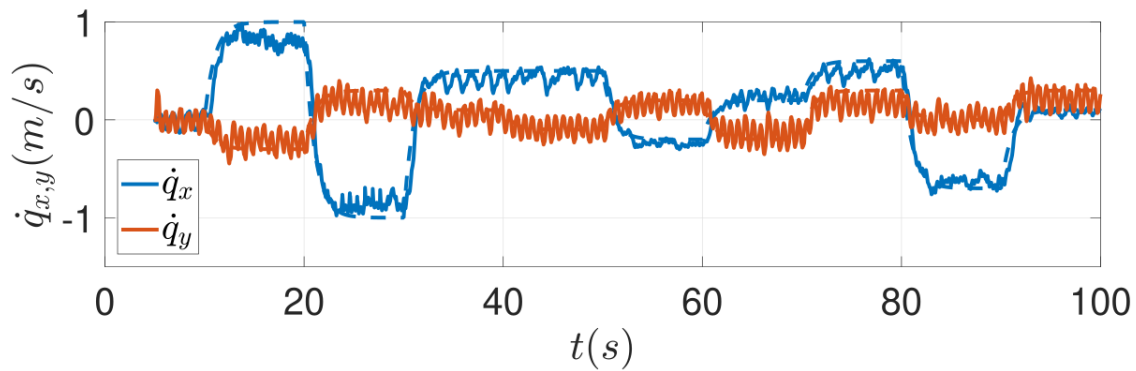
We demonstrate an extension of RMA (which we call A-RMA) on a life-sized bipedal robot, Cassie. It extends the RMA algorithm by accounting for imperfect extrinsics estimation from the adaptation module. This is done by adding an additional base policy finetuning phase using model-free RL on the imperfect extrinsics. A-RMA, although only trained in simulation, shows generalization to terrains beyond what is seen during training without additional real-world finetuning or calibration.



(a) Robust MLP



(b) RMA



(c) A-RMA

Figure 6.3: Plot of tracking error in sagittal and lateral directions different controllers. RMA and A-RMA produce less oscillations than the Robust MLP once the lateral velocity has stabilized.

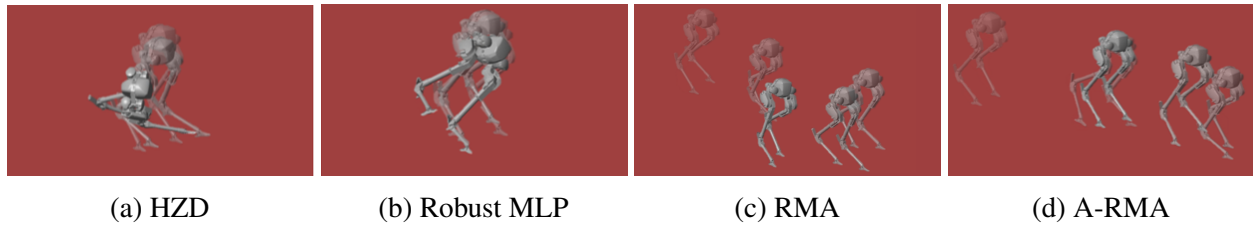


Figure 6.4: Trajectories of Cassie on slippery ground with friction coefficient of 0.2. We observe that A-RMA has minimal lateral deviation and is more stable than RMA, which shows significant deviation in the lateral walking direction.

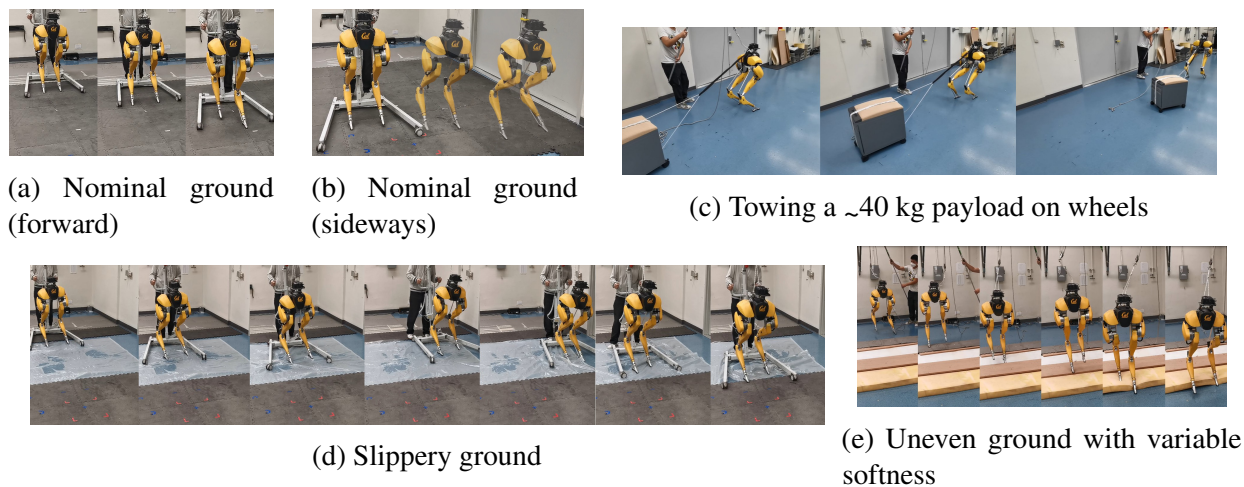


Figure 6.5: Real world deployment of Cassie walking on nominal ground, towing a payload, walking on a slippery surface, and on rough terrain with variable softness of the ground. The payload and rough terrain tests were repeated twice with consistent results. The same A-RMA policy was deployed in all these deployment scenarios without any real world fine-tuning, task specific tuning, or calibration. We observe that A-RMA maintains stability in all of these deployment scenarios despite never having seen some of them during training.

Chapter 7

Conclusion

To summarize, this thesis takes a significant leap by demonstrating purely learning-based controllers that can successfully control highly dynamic systems in the real world in a diverse set of environments. The key benefit of learning-based methods over classical methods is that they require minimal modifications to work on a task and have the promise of paving the path towards the development of a robot generalist – capable of solving a diverse set of tasks in the real world reliably. The algorithm proposed in this thesis is called Rapid Motor Adaptation (RMA), and its design is guided by the following three principles:

- using a principled approach by using data-driven search via reinforcement learning,
- using energy efficiency as the penalty for learning natural behaviors,
- enabling rapid adaptation to new unseen scenarios in fractions of a second for real world generalization.

RMA is first developed in the context of blind quadruped walking on challenging terrains in the real world and then is later extended to visual walking to allow quadrupeds to traverse even more complex terrains such as stepping stones that are beyond the reach of blind agents. Since the core algorithm does not assume anything specific to walking, we show that it can also be applied to in-hand rotation with a four-fingered anthropomorphic hand to rotate a diverse set of objects in the real world. In the same spirit, we show how RMA can be used to learn a single universal controller capable of flying drones of diverse morphologies, and finally for blind bipedal walking.

Bibliography

- [1] Karen E Adolph et al. “How do you learn to walk? Thousands of steps and dozens of falls per day”. In: *Psychological science* (2012).
- [2] Zofia Afelt, Janusz Blaszczyk, and Czeslawa Dobrzecka. “Speed control in animal locomotion: transitions between symmetrical and nonsymmetrical gaits in the dog”. In: *Acta Neurobiol Exp* (1983).
- [3] Ananye Agarwal et al. “Legged locomotion in challenging terrains using egocentric vision”. In: *Conference on Robot Learning*. PMLR. 2022, pp. 403–415.
- [4] Agility Robotics. *cassie-mujoco-sim*. (2018). URL: <https://github.com/osudr1/cassie-mujoco-sim>.
- [5] Ayush Agrawal et al. “Vision-aided Dynamic Quadrupedal Locomotion on Discrete Terrain using Motion Libraries”. In: *arXiv preprint arXiv:2110.00891* (2021).
- [6] MYM Ahmed and N Qin. “Surrogate-based aerodynamic design optimization: Use of surrogates in aerodynamic design optimization”. In: *International Conference on Aerospace Sciences and Aviation Technology*. 2009.
- [7] R McN Alexander. “The gaits of bipedal and quadrupedal animals”. In: *The International Journal of Robotics Research* (1984).
- [8] R McN Alexander and AS Jayes. “A dynamic similarity hypothesis for the gaits of quadrupedal mammals”. In: *Journal of zoology* 201.1 (1983), pp. 135–152.
- [9] Arthur Allshire et al. “Transferring Dexterous Manipulation from GPU Simulation to a Remote Real-World Trifinger”. In: *arXiv* (2021). DOI: 10.48550/arXiv.2108.09779.
- [10] Aaron D Ames et al. “Rapidly exponentially stabilizing control lyapunov functions and hybrid zero dynamics”. In: *IEEE Transactions on Automatic Control* (2014).
- [11] Taylor Apgar et al. “Fast Online Trajectory Optimization for the Bipedal Robot Cassie.” In: *Robotics: Science and Systems*. 2018.
- [12] Yunfei Bai and C. Karen Liu. “Dexterous Manipulation Using Both Palm and Fingers”. In: *International Conference on Robotics and Automation (ICRA)*. 2014. DOI: 10.1109/ICRA.2014.6907059.

- [13] Monica Barragan, Nikolai Flowers, and Aaron M. Johnson. “MiniRHex: A Small, Open-source, Fully Programmable Walking Hexapod”. In: *RSS Workshop*. 2018.
- [14] Dominik Belter, Przemysław Łabcki, and Piotr Skrzypczyński. “Estimating terrain elevation maps from sparse and uncertain multi-sensor data”. In: *2012 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. 2012, pp. 715–722. DOI: 10.1109/ROBIO.2012.6491052.
- [15] John EA Bertram. “Constrained optimization in human walking: cost minimization and gait plasticity”. In: *Journal of experimental biology* 208.6 (2005), pp. 979–991.
- [16] Dimitri P Bertsekas. *Dynamic programming: deterministic and stochastic models*. Prentice-Hall, Inc., 1987.
- [17] Gerardo Bleedt et al. “MIT Cheetah 3: Design and control of a robust, dynamic quadruped robot”. In: *IROS*. 2018.
- [18] Adrian Boeing and Thomas Bräunl. “Leveraging multiple simulators for crossing the reality gap”. In: *2012 12th International Conference on Control Automation Robotics & Vision (ICARCV)*. IEEE. 2012.
- [19] Josh C Bongard and Hod Lipson. “Nonlinear system identification using coevolution of models and tests”. In: *IEEE Transactions on Evolutionary Computation* (2005).
- [20] Roberto Calandra et al. “Bayesian optimization for learning gaits under uncertainty”. In: *Annals of Mathematics and Artificial Intelligence* (2016).
- [21] Guillermo A Castillo et al. “Hybrid zero dynamics inspired feedback control policy design for 3d bipedal locomotion using reinforcement learning”. In: *IEEE International Conference on Robotics and Automation*. 2020, pp. 8746–8752.
- [22] Guillermo A. Castillo et al. “Velocity Regulation of 3D Bipedal Walking Robots with Uncertain Dynamics Through Adaptive Neural Network Controller”. In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2020, pp. 7703–7709. DOI: 10.1109/IROS45743.2020.9341569.
- [23] R Omar Chavez-Garcia et al. “Learning ground traversability from simulations”. In: *RA-L* (2018).
- [24] Tao Chen, Jie Xu, and Pulkit Agrawal. “A System for General In-Hand Object Re-Orientation”. In: *Conference on Robot Learning (CoRL)*. 2022. DOI: 10.48550/arXiv.2111.03043.
- [25] Joel Chestnutt. *Navigation planning for legged robots*. Carnegie Mellon University, 2007.
- [26] Annett Chilian and Heiko Hirschmüller. “Stereo camera based navigation of mobile robots on rough terrain”. In: *IROS*. 2009.
- [27] Krzysztof Choromanski et al. “Optimizing simulations with noise-tolerant structured exploration”. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018.

- [28] Ignasi Clavera et al. “Learning to Adapt in Dynamic, Real-World Environments through Meta-Reinforcement Learning”. In: *ICLR*. 2019.
- [29] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. “Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)”. In: *International Conference on Learning Representations (ICLR)*. 2016. DOI: 10.48550/arXiv.1511.07289.
- [30] Whitney G Cole, Scott R Robinson, and Karen E Adolph. “Bouts of steps: The organization of infant exploration”. In: *Developmental psychobiology* (2016).
- [31] Erwin Coumans, Xue Bin Peng, and Yuxiang Yang. *Convex MPC controller for AI*. https://github.com/google-research/motion_imitation/.
- [32] Xingye Da et al. “From 2D design of underactuated bipedal gaits to 3D implementation: Walking with speed tracking”. In: *IEEE Access* 4 (2016), pp. 3469–3478.
- [33] Xingye Da et al. “Learning a contact-adaptive controller for robust, efficient legged locomotion”. In: *arXiv preprint arXiv:2009.10019* (2020).
- [34] Nikhil Chavan Daffe et al. “Extrinsic Dexterity: In-Hand Manipulation with External Forces”. In: *International Conference on Robotics and Automation (ICRA)*. 2014. DOI: 10.1109/ICRA.2014.6907062.
- [35] Hongkai Dai, Andrés Valenzuela, and Russ Tedrake. “Whole-body motion planning with centroidal dynamics and full kinematics”. In: *IEEE-RAS International Conference on Humanoid Robots*. 2014, pp. 295–302.
- [36] Ewen Dantec et al. “Whole body model predictive control with a memory of motion: Experiments on a torque-controlled talos”. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021, pp. 8202–8208.
- [37] Martin De Lasa, Igor Mordatch, and Aaron Hertzmann. “Feature-based locomotion controllers”. In: *ACM Transactions on Graphics (TOG)* (2010).
- [38] Jared Di Carlo et al. “Dynamic locomotion in the mit cheetah 3 through convex model-predictive control”. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2018.
- [39] Peter Eckert et al. “Comparing the effect of different spine and leg designs for a small bounding quadruped robot”. In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2015, pp. 3128–3133.
- [40] Alejandro Escontrela et al. “Zero-shot terrain generalization for visual locomotion policies”. In: *arXiv preprint arXiv:2011.05513* (2020).
- [41] *Experiment Video*. (2022). URL: <https://youtu.be/HSdFHX0qQqg>.
- [42] Péter Fankhauser, Michael Bloesch, and Marco Hutter. “Probabilistic terrain mapping for mobile robots with uncertain localization”. In: *IEEE Robotics and Automation Letters* 3.4 (2018), pp. 3019–3026.

- [43] Péter Fankhauser et al. “Robot-centric elevation mapping with uncertainty estimates”. In: *Mobile Service Robotics*. World Scientific, 2014, pp. 433–440.
- [44] Péter Fankhauser et al. “Robust rough-terrain locomotion with a quadrupedal robot”. In: *ICRA*. 2018.
- [45] Ronald Fearing. “Implementing a Force Strategy for Object Re-orientation”. In: *International Conference on Robotics and Automation (ICRA)*. 1986. DOI: 10.1109/ROBOT.1986.1087655.
- [46] Siyuan Feng et al. “3D walking based on online optimization”. In: *2013 13th IEEE-RAS International Conference on Humanoid Robots*. 2013, pp. 21–27.
- [47] Chelsea Finn, Pieter Abbeel, and Sergey Levine. “Model-agnostic meta-learning for fast adaptation of deep networks”. In: *International Conference on Machine Learning*. PMLR. 2017.
- [48] Zipeng Fu et al. “Coupling Vision and Proprioception for Navigation of Legged Robots”. In: *arXiv preprint arXiv:2112.02094* (2021).
- [49] Zipeng Fu et al. “Coupling Vision and Proprioception for Navigation of Legged Robots”. In: *Computer Vision and Pattern Recognition (CVPR)*. 2022. DOI: 10.48550/arXiv.2112.02094.
- [50] Zipeng Fu et al. “Minimizing Energy Consumption Leads to the Emergence of Gaits in Legged Robots”. In: *CoRL*. 2021.
- [51] Scott Fujimoto, Herke Hoof, and David Meger. “Addressing function approximation error in actor-critic methods”. In: *ICML*. 2018.
- [52] Noriatsu Furukawa et al. “Dynamic Regrasping Using a High-speed Multifingered Hand and a High-speed Vision System”. In: *International Conference on Robotics and Automation (ICRA)*. 2006. DOI: 10.1109/ROBOT.2006.1641181.
- [53] Siddhant Gangapurwala et al. “Real-Time Trajectory Adaptation for Quadrupedal Locomotion using Deep Reinforcement Learning”. In: *International Conference on Robotics and Automation (ICRA)*. 2021.
- [54] Christian Gehring et al. “Practice makes perfect: An optimization-based approach to controlling agile motions for a quadruped robot”. In: *IEEE Robotics & Automation Magazine* (2016).
- [55] Hartmut Geyer, Andre Seyfarth, and Reinhard Blickhan. “Positive force feedback in bouncing gaits?” In: *Proceedings of the Royal Society of London. Series B: Biological Sciences* (2003).
- [56] Yukai Gong and Jessy Grizzle. “Angular Momentum about the Contact Point for Control of Bipedal Locomotion: Validation in a LIP-based Controller”. In: *arXiv preprint arXiv:2008.10763* (2020).
- [57] Yukai Gong et al. “Feedback control of a cassie bipedal robot: Walking, standing, and riding a segway”. In: *American Control Conference*. 2019, pp. 4559–4566.

- [58] J W. Grizzle et al. “3D Bipedal Robotic Walking: Models, Feedback Control, and Open Problems”. In: *IFAC Symposium on Nonlinear Control Systems*. 2010.
- [59] Xiaoxiao Guo, Wei Li, and Francesco Iorio. “Convolutional neural networks for steady flow approximation”. In: *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 2016.
- [60] Abhishek Gupta et al. “Learning Dexterous Manipulation for a Soft Robotic Hand from Human Demonstration”. In: *International Conference on Intelligent Robots and Systems (IROS)*. 2016. DOI: 10.1109/IROS.2016.7759557.
- [61] Jérôme Guzzi et al. “Path planning with local motion estimations”. In: *RA-L* (2020).
- [62] Tuomas Haarnoja et al. “Learning to Walk Via Deep Reinforcement Learning”. In: *Robotics: Science and Systems*. 2019.
- [63] Tuomas Haarnoja et al. “Learning to walk via deep reinforcement learning”. In: *arXiv preprint arXiv:1812.11103* (2018).
- [64] Li Han and Jeffrey C Trinkle. “Dextrous Manipulation by Rolling and Finger Gaiting”. In: *International Conference on Robotics and Automation (ICRA)*. 1998. DOI: 10.1109/ROBOT.1998.677060.
- [65] Josiah Hanna and Peter Stone. “Grounded action transformation for robot learning in simulation”. In: *AAAI*. 2017.
- [66] Milton Hildebrand. “Symmetrical gaits of horses”. In: *Science* (1965).
- [67] Donald F Hoyt and C Richard Taylor. “Gait and the energetics of locomotion in horses”. In: *Nature* (1981).
- [68] Wenlong Huang et al. “Generalization in Dexterous Manipulation via Geometry-Aware Multi-Task Learning”. In: *arXiv* (2021). DOI: 10.48550/arXiv.2111.03062.
- [69] Marco Hutter et al. “Anymal-a highly mobile and dynamic quadrupedal robot”. In: *IROS*. 2016.
- [70] Jemin Hwangbo. *RaisimGymTorch*. <https://raisim.com/sections/RaisimGymTorch.html>. 2020-2021.
- [71] Jemin Hwangbo, Joonho Lee, and Marco Hutter. “Per-contact iteration method for solving contact dynamics”. In: *IEEE Robotics and Automation Letters* (2018). URL: www.raisim.com.
- [72] Jemin Hwangbo et al. “Learning agile and dynamic motor skills for legged robots”. In: *Science Robotics* (2019).
- [73] Dong Jin Hyun et al. “Implementation of trot-to-gallop transition and subsequent gallop on the MIT Cheetah I”. In: *IJRR* (2016).
- [74] Chieko Sarah Imai et al. “Vision-Guided Quadrupedal Locomotion in the Wild with Multi-Modal Delay Randomization”. In: *arXiv:2109.14549* (2021).

- [75] Atil Iscen et al. “Policies modulating trajectory generators”. In: *Conference on Robot Learning*. PMLR. 2018.
- [76] Deepali Jain, Atil Iscen, and Ken Caluwaerts. “From pixels to legs: Hierarchical learning of quadruped locomotion”. In: *arXiv preprint arXiv:2011.11722* (2020).
- [77] AS Jayes and R McN Alexander. “Mechanics of locomotion of dogs (*Canis familiaris*) and sheep (*Ovis aries*)”. In: *Journal of Zoology* (1978).
- [78] Fabian Jenelten et al. “Perceptive locomotion in rough terrain—online foothold optimization”. In: *RA-L* (2020).
- [79] Tobias Johannink et al. “Residual reinforcement learning for robot control”. In: *International Conference on Robotics and Automation*. 2019, pp. 6023–6029.
- [80] Aaron M Johnson et al. “Tail assisted dynamic self righting”. In: *Adaptive Mobile Robotics*. World Scientific, 2012.
- [81] Mrinal Kalakrishnan et al. “Fast, robust quadruped locomotion over challenging terrain”. In: *2010 IEEE International Conference on Robotics and Automation*. IEEE. 2010.
- [82] Mrinal Kalakrishnan et al. “Learning locomotion over rough terrain using terrain templates”. In: *IROS*. 2009.
- [83] Yiannis Karayiannidis, Christian Smith, Danica Kragic, et al. “Adaptive Control for Pivoting with Visual and Tactile Feedback”. In: *International Conference on Robotics and Automation (ICRA)*. 2016. DOI: 10.1109/ICRA.2016.7487159.
- [84] Gagan Khandate, Maxmillian Haas-Heger, and Matei Ciocarlie. “On the Feasibility of Learning Finger-gaiting In-hand Manipulation with Intrinsic Sensing”. In: *International Conference on Robotics and Automation (ICRA)*. 2022. DOI: 10.1109/icra46639.2022.9812212.
- [85] Mahdi Khoramshahi et al. “Piecewise linear spine for speed–energy efficiency trade-off in quadruped robots”. In: *Robotics and Autonomous Systems* (2013).
- [86] Donghyun Kim et al. “Vision aided dynamic exploration of unstructured terrain with a small-scale quadruped robot”. In: *ICRA*. 2020.
- [87] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *ICLR*. 2015.
- [88] Alexander Kleiner and Christian Dornhege. “Real-time localization and elevation mapping within urban search and rescue scenarios”. In: *Journal of Field Robotics* 24.8-9 (2007), pp. 723–745.
- [89] Jens Kober, J Andrew Bagnell, and Jan Peters. “Reinforcement learning in robotics: A survey”. In: *The International Journal of Robotics Research* (2013).
- [90] Nate Kohl and Peter Stone. “Policy gradient reinforcement learning for fast quadrupedal locomotion”. In: *IEEE International Conference on Robotics and Automation, 2004*. Vol. 3. IEEE. 2004, pp. 2619–2624.

- [91] J. Zico Kolter, Mike P Rodgers, and Andrew Y. Ng. “A control architecture for quadruped locomotion over rough terrain”. In: *ICRA*. 2008.
- [92] Sylvain Koos, Jean-Baptiste Mouret, and Stéphane Doncieux. “Crossing the reality gap in evolutionary robotics by promoting transferable controllers”. In: *Proceedings of the 12th annual conference on Genetic and evolutionary computation*. 2010.
- [93] Donald L Kramer and Robert L McLaughlin. “The behavioral ecology of intermittent locomotion”. In: *American Zoologist* (2001).
- [94] Scott Kuindersma, Frank Permenter, and Russ Tedrake. “An efficiently solvable quadratic program for stabilizing dynamic locomotion”. In: *IEEE International Conference on Robotics and Automation*. 2014, pp. 2589–2594.
- [95] Ashish Kumar et al. “Adapting Rapid Motor Adaptation for Bipedal Robots”. In: *arXiv preprint arXiv:2205.15299* (2022).
- [96] Ashish Kumar et al. “RMA: Rapid Motor Adaptation for Legged Robots”. In: *RSS*. 2021.
- [97] Ashish Kumar et al. “RMA: Rapid Motor Adaptation for Legged Robots”. In: *Robotics: Science and Systems (RSS)*. 2021. DOI: 10.15607/RSS.2021.XVII.011.
- [98] Vikash Kumar, Emanuel Todorov, and Sergey Levine. “Optimal Control with Learned Local Models: Application to Dexterous Manipulation”. In: *International Conference on Robotics and Automation (ICRA)*. 2016. DOI: 10.1109/ICRA.2016.7487156.
- [99] Vikash Kumar et al. “Learning Dexterous Manipulation Policies from Experience and Imitation”. In: *arXiv* (2016). DOI: 10.48550/arXiv.1611.05095.
- [100] In-So Kweon and Takeo Kanade. “High-resolution terrain map from multiple sensor data”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 14.2 (1992), pp. 278–292.
- [101] In-So Kweon et al. “Terrain mapping for a roving planetary explorer”. In: *IEEE International Conference on Robotics and Automation*. IEEE. 1989, pp. 997–1002.
- [102] Joonho Lee et al. “Learning quadrupedal locomotion over challenging terrain”. In: *Science robotics* (2020).
- [103] Zhongyu Li, Christine Cummings, and Koushil Sreenath. “Animated Cassie: A Dynamic Relatable Robotic Character”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2020.
- [104] Zhongyu Li et al. “Reinforcement Learning for Robust Parameterized Locomotion Control of Bipedal Robots”. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. 2021, pp. 2811–2817. DOI: 10.1109/ICRA48506.2021.9560769.
- [105] Zhongyu Li et al. “Vision-aided autonomous navigation of underactuated bipedal robots in height-constrained environments”. In: *arXiv preprint arXiv:2109.05714* (2021).
- [106] Timothy P. Lillicrap et al. “Continuous control with deep reinforcement learning.” In: *ICLR*. 2016.

- [107] Jack M Loomis et al. “Visual space perception and visually directed action.” In: *Journal of experimental psychology: Human Perception and Performance* 18.4 (1992), p. 906.
- [108] Jingru Luo and Kris Hauser. “Robust trajectory optimization under frictional contact with iterative learning”. In: *Autonomous Robots* (2017).
- [109] Octavio Antonio Villarreal Magana et al. “Fast and continuous foothold adaptation for dynamic locomotion through cnns”. In: *RA-L* (2019).
- [110] Viktor Makoviychuk et al. “Isaac Gym: High Performance GPU-Based Physics Simulation For Robot Learning”. In: *arXiv* (2021). DOI: 10.48550/arXiv.2108.10470.
- [111] Zachary Manchester and Scott Kuindersma. “Variational contact-implicit trajectory optimization”. In: *Robotics Research*. Springer, 2020.
- [112] Gabriel B Margolis et al. “Learning to Jump from Pixels”. In: *arXiv preprint arXiv:2110.15344* (2021).
- [113] Duane W Marhefka et al. “Intelligent control of quadruped gallops”. In: *IEEE/ASME Transactions On Mechatronics* 8.4 (2003), pp. 446–456.
- [114] Carlos Mastalli et al. “On-line and on-board planning and perception for quadrupedal locomotion”. In: *2015 IEEE International Conference on Technologies for Practical Robot Applications*. 2015.
- [115] Carlos Mastalli et al. “Trajectory and foothold optimization using low-dimensional models for rough terrain locomotion”. In: *ICRA*. 2017.
- [116] Jonathan S Matthis and Brett R Fajen. “Visual control of foot placement when walking over complex terrain.” In: *Journal of experimental psychology: human perception and performance* 40.1 (2014), p. 106.
- [117] Takahiro Miki et al. “Elevation Mapping for Locomotion and Navigation using GPU”. In: *arXiv preprint arXiv:2204.12876* (2022).
- [118] Takahiro Miki et al. “Learning robust perceptive locomotion for quadrupedal robots in the wild”. In: *Science Robotics* 7.62 (2022), eabk2822.
- [119] Hirofumi Miura and Isao Shimoyama. “Dynamic walk of a biped”. In: *IJRR* (1984).
- [120] Volodymyr Mnih et al. “Asynchronous methods for deep reinforcement learning”. In: *ICML*. 2016.
- [121] Amir A Mohagheghi, Renato Moraes, and Aftab E Patla. “The effects of distant and on-line visual information on the control of approach phase and step over an obstacle during locomotion”. In: *Experimental brain research* 155.4 (2004), pp. 459–468.
- [122] Igor Mordatch, Zoran Popović, and Emanuel Todorov. “Contact-invariant optimization for hand manipulation”. In: *Proceedings of the ACM SIGGRAPH/Eurographics symposium on computer animation*. 2012.

- [123] Andrew Morgan et al. “Complex In-Hand Manipulation via Compliance-Enabled Finger Gaiting and Multi-Modal Planning”. In: *IEEE Robotics and Automation Letters (RA-L)* (2022). DOI: 10.1109/LRA.2022.3145961.
- [124] Eadweard Muybridge. “The horse in motion”. In: *Library of Congress Prints and Photographs Division* (1878). URL: <http://www.loc.gov/pictures/resource/ppmsca.23778/>.
- [125] Ofir Nachum et al. “Multi-Agent Manipulation via Locomotion using Hierarchical Sim2Real”. In: *CoRL*. 2020.
- [126] Anusha Nagabandi et al. “Deep Dynamics Models for Learning Dexterous Manipulation”. In: *Conference on Robot Learning (CoRL)*. 2019. DOI: 10.48550/arXiv.1909.11652.
- [127] Michael Neunert, Thiago Boaventura, and Jonas Buchli. “Why off-the-shelf physics simulators fail in evaluating feedback controller performance—a case study for quadrupedal robots”. In: *Advances in Cooperative Robotics*. World Scientific, 2017.
- [128] Allison M. Okamura, Niels Smaby, and Mark R. Cutkosky. “An Overview of Dexterous Manipulation”. In: *International Conference on Robotics and Automation (ICRA)*. 2000. DOI: 10.1109/ROBOT.2000.844067.
- [129] OpenAI et al. “Learning Dexterous In-Hand Manipulation”. In: *The International Journal of Robotics Research (IJRR)* (2019). DOI: 10.1177/0278364919887447.
- [130] OpenAI et al. “Solving Rubik’s Cube with a Robot Hand”. In: *arXiv preprint arXiv:1910.07113* (2019). DOI: 10.48550/arXiv.1910.07113.
- [131] Yiyuan Pan et al. “GEM: Online Globally Consistent Dense Elevation Mapping for Unstructured Terrain”. In: *IEEE Transactions on Instrumentation and Measurement* 70 (2021), pp. 1–13. DOI: 10.1109/TIM.2020.3044338.
- [132] Yiyuan Pan et al. “GPU accelerated real-time traversability mapping”. In: *2019 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. 2019, pp. 734–740. DOI: 10.1109/ROBIO49542.2019.8961816.
- [133] Aftab E Patla. “Understanding the roles of vision in the control of human locomotion”. In: *Gait & posture* 5.1 (1997), pp. 54–69.
- [134] Xue Bin Peng, Glen Berseth, and Michiel Van de Panne. “Terrain-adaptive locomotion skills using deep reinforcement learning”. In: *ACM Transactions on Graphics (TOG)* 35.4 (2016), pp. 1–12.
- [135] Xue Bin Peng et al. “Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning”. In: *ACM Transactions on Graphics (TOG)* 36.4 (2017), pp. 1–13.
- [136] Xue Bin Peng et al. “Learning Agile Robotic Locomotion Skills by Imitating Animals”. In: *RSS*. 2020.
- [137] Xue Bin Peng et al. “Sim-to-real transfer of robotic control with dynamics randomization”. In: *ICRA*. 2018.

- [138] Robert Platt, Andrew H. Fagg, and Roderic A. Grupen. “Manipulation Gaits: Sequences of Grasp Control Tasks”. In: *International Conference on Robotics and Automation (ICRA)*. 2004. DOI: 10.1109/ROBOT.2004.1307247.
- [139] Delyle T Polet and John EA Bertram. “An inelastic quadrupedal model discovers four-beat walking, two-beat running, and pseudo-elastic actuation as energetically optimal”. In: *PLoS computational biology* (2019).
- [140] Delyle T Polet, Ryan T Schroeder, and John EA Bertram. “Reducing gravity takes the bounce out of running”. In: *Journal of Experimental Biology* 221.3 (2018).
- [141] Michael Posa, Cecilia Cantu, and Russ Tedrake. “A direct method for trajectory optimization of rigid bodies through contact”. In: *The International Journal of Robotics Research* (2014).
- [142] Jerry Pratt et al. “Capturability-based analysis and control of legged locomotion, Part 2: Application to M2V2, a lower-body humanoid”. In: *The international journal of robotics research* 31.10 (2012), pp. 1117–1133.
- [143] Haozhi Qi et al. “In-Hand Object Rotation via Rapid Motor Adaptation”. In: *arXiv preprint arXiv:2210.04887* (2022).
- [144] Nicolaus A Radford et al. “Valkyrie: Nasa’s first bipedal humanoid robot”. In: *Journal of Field Robotics* 32.3 (2015), pp. 397–419.
- [145] Ilija Radosavovic et al. “State-Only Imitation Learning for Dexterous Manipulation”. In: *International Conference on Intelligent Robots and Systems (IROS)*. 2021. DOI: 10.1109/IROS51168.2021.9636557.
- [146] Marc H. Raibert. “Hopping in legged systems-modeling and simulation for the two-dimensional one-legged case”. In: *IEEE Transactions on Systems, Man, and Cybernetics* (1984).
- [147] Marc H Raibert. “Trotting, pacing and bounding by a quadruped robot”. In: *Journal of biomechanics* 23 (1990), pp. 79–98.
- [148] Aravind Rajeswaran et al. “Learning Complex Dexterous Manipulation with Deep Reinforcement Learning and Demonstrations”. In: *Robotics: Science and Systems (RSS)*. 2018. DOI: 10.15607/RSS.2018.XIV.049.
- [149] Nathan Ratliff et al. “CHOMP: Gradient optimization techniques for efficient motion planning”. In: *2009 IEEE International Conference on Robotics and Automation*. IEEE. 2009.
- [150] Jenna Reher and Aaron D Ames. “Control Lyapunov functions for compliant hybrid zero dynamic walking”. In: *arXiv preprint arXiv:2107.04241* (2021).
- [151] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. “A reduction of imitation learning and structured prediction to no-regret online learning”. In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. 2011.
- [152] Nikita Rudin et al. “Learning to walk in minutes using massively parallel deep reinforcement learning”. In: *Conference on Robot Learning*. PMLR. 2022, pp. 91–100.

- [153] Daniela Rus. “In-Hand Dexterous Manipulation of Piecewise-Smooth 3-D Objects”. In: *The International Journal of Robotics Research (IJRR)* (1999). DOI: 10.1177/02783649922066268.
- [154] Fereshteh Sadeghi and Sergey Levine. “Cad2rl: Real single-image flight without a single real image”. In: *Robotics: Science and Systems*. 2013.
- [155] Uluc Saranli, Martin Buehler, and Daniel E Koditschek. “RHex: A simple and highly mobile hexapod robot”. In: *The International Journal of Robotics Research* (2001).
- [156] Jean-Philippe Saut et al. “Dexterous Manipulation Planning Using Probabilistic Roadmaps in Continuous Grasp Subspaces”. In: *International Conference on Intelligent Robots and Systems (IROS)*. 2007. DOI: 10.1109/IROS.2007.4399090.
- [157] John Schulman et al. “High-Dimensional Continuous Control Using Generalized Advantage Estimation”. In: *4th International Conference on Learning Representations*. 2016.
- [158] John Schulman et al. “Proximal policy optimization algorithms”. In: *arXiv:1707.06347* (2017).
- [159] Jonah Siekmann et al. “Blind Bipedal Stair Traversal via Sim-to-Real Reinforcement Learning”. In: *Robotics: Science and Systems*. 2021.
- [160] Jonah Siekmann et al. “Learning Memory-Based Control for Human-Scale Bipedal Locomotion”. In: *Robotics: Science and Systems*. 2020.
- [161] Jonah Siekmann et al. “Sim-to-real learning of all common bipedal gaits via periodic reward composition”. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021, pp. 7309–7315.
- [162] Leon Sievers, Johannes Pitz, and Berthold Bäuml. “Learning Purely Tactile In-Hand Manipulation with a Torque-Controlled Hand”. In: *International Conference on Robotics and Automation (ICRA)* (2022). DOI: 10.48550/arXiv.2204.03698.
- [163] Satinder P Singh and Richard C Yee. “An upper bound on the loss from approximate optimal-value functions”. In: *Machine Learning* 16.3 (1994), pp. 227–233.
- [164] Laura Smith et al. “Legged Robots that Keep on Learning: Fine-Tuning Locomotion Policies in the Real World”. In: *ICRA*. 2022.
- [165] Xingyou Song et al. “Rapidly Adaptable Legged Robots via Evolutionary Meta-Learning”. In: *IROS*. 2020.
- [166] Koushil Sreenath et al. “A compliant hybrid zero dynamics controller for stable, efficient and fast bipedal walking on MABEL”. In: *IJRR* (2011).
- [167] Manoj Srinivasan and Andy Ruina. “Computer optimization of a minimal biped model discovers walking and running”. In: *Nature* (2006).
- [168] Balakumar Sundaralingam and Tucker Hermans. “Relaxed-Rigidity Constraints: Kinematic Trajectory Optimization and Collision Avoidance for In-Grasp Manipulation”. In: *Autonomous Robots* (2019). DOI: 10.1007/s10514-018-9772-z.

- [169] Jie Tan et al. “Sim-to-Real: Learning Agile Locomotion For Quadruped Robots”. In: *RSS*. 2018.
- [170] Clark Teeple et al. “Controlling Palm-Object Interactions Via Friction for Enhanced In-Hand Manipulation”. In: *IEEE Robotics and Automation Letters (RA-L)* (2022). DOI: 10.1109/LRA.2022.3143578.
- [171] Josh Tobin et al. “Domain randomization for transferring deep neural networks from simulation to the real world”. In: *IROS*. 2017.
- [172] 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*. IEEE. 2012.
- [173] Vassilios Tsounis et al. “Deepgait: Planning and control of quadrupedal gaits using deep reinforcement learning”. In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 3699–3706.
- [174] Herke Van Hoof et al. “Learning Robot In-Hand Manipulation with Tactile Features”. In: *International Conference on Humanoid Robots (Humanoids)*. 2015. DOI: 10.1109/HUMANOIDS.2015.7363524.
- [175] Christopher L Vaughan and Mark J O’Malley. “Froude and the contribution of naval architecture to our understanding of bipedal locomotion”. In: *Gait & posture* (2005).
- [176] Miomir Vukobratovic and Branislav Borovac. “Zero-moment point-thirty five years of its life”. In: *International journal of humanoid robotics* 1.01 (2004), pp. 157–173.
- [177] Xingxing Wang. *Unitree Robotics*. <https://www.unitree.com/>.
- [178] Lorenz Wellhausen and Marco Hutter. “Rough Terrain Navigation for Legged Robots using Reachability Planning and Template Learning”. In: *IROS*. 2021.
- [179] Paul J Werbos. “Backpropagation through time: what it does and how to do it”. In: *Proceedings of the IEEE* 78.10 (1990), pp. 1550–1560.
- [180] Martin Wermelinger et al. “Navigation planning for legged robots in challenging terrain”. In: *IROS*. 2016.
- [181] WonikRobotics. *AllegroHand*. <https://www.wonikrobotics.com/>. 2013.
- [182] Zhaoming Xie et al. “ALLSTEPS: Curriculum-driven Learning of Stepping Stone Skills”. In: *Computer Graphics Forum*. Wiley Online Library. 2020.
- [183] Zhaoming Xie et al. “Dynamics Randomization Revisited: A Case Study for Quadrupedal Locomotion”. In: *ICRA*. 2021.
- [184] Zhaoming Xie et al. “Feedback control for cassie with deep reinforcement learning”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2018, pp. 1241–1246.
- [185] Zhaoming Xie et al. “Learning locomotion skills for cassie: Iterative design and sim-to-real”. In: *Conference on Robot Learning*. 2020, pp. 317–329.

- [186] Xiaobin Xiong and Aaron D Ames. “Coupling reduced order models via feedback control for 3d underactuated bipedal robotic walking”. In: *IEEE-RAS International Conference on Humanoid Robots*. 2018.
- [187] Bowen Yang et al. “Real-time optimal navigation planning using learned motion costs”. In: *ICRA*. 2021.
- [188] Lizhi Yang et al. “Bayesian Optimization Meets Hybrid Zero Dynamics: Safe Parameter Learning for Bipedal Locomotion Control”. In: *arXiv preprint arXiv:2203.02570* (2022).
- [189] Ruihan Yang et al. “Learning Vision-Guided Quadrupedal Locomotion End-to-End with Cross-Modal Transformers”. In: *ICLR*. 2022.
- [190] Yuxiang Yang et al. “Data efficient reinforcement learning for legged robots”. In: *Conference on Robot Learning*. PMLR. 2020.
- [191] Yuxiang Yang et al. “Fast and Efficient Locomotion via Learned Gait Transitions”. In: *CoRL*. 2021.
- [192] KangKang Yin, Kevin Loken, and Michiel Van de Panne. “Simbicon: Simple biped locomotion control”. In: *ACM Transactions on Graphics* (2007).
- [193] Wenhao Yu, C. Karen Liu, and Greg Turk. “Policy Transfer with Strategy Optimization”. In: *ICLR*. 2018.
- [194] Wenhao Yu, Greg Turk, and C Karen Liu. “Learning symmetric and low-energy locomotion”. In: *ACM Transactions on Graphics (TOG)* (2018).
- [195] Wenhao Yu et al. “Learning fast adaptation with meta strategy optimization”. In: *RA-L* (2020).
- [196] Wenhao Yu et al. “Preparing for the Unknown: Learning a Universal Policy with Online System Identification”. In: *RSS*. 2017.
- [197] Wenhao Yu et al. “Sim-to-Real Transfer for Biped Locomotion”. In: *IROS*. 2019.
- [198] Wenhao Yu et al. “Visual-Locomotion: Learning to Walk on Complex Terrains with Vision”. In: *5th Annual Conference on Robot Learning*. 2021.
- [199] W. Yu et al. “Sim-to-Real Transfer for Biped Locomotion”. In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2019, pp. 3503–3510. DOI: 10.1109/IROS40897.2019.8968053.
- [200] Dingqi Zhang et al. “Learning a Single Near-hover Position Controller for Vastly Different Quadcopters”. In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2023, pp. 1263–1269.
- [201] Wenxuan Zhou, Lerrel Pinto, and Abhinav Gupta. “Environment probing interaction policies”. In: *ICLR*. 2019.
- [202] J Zico Kolter and Andrew Y Ng. “The stanford littledog: A learning and rapid replanning approach to quadruped locomotion”. In: *The International Journal of Robotics Research* (2011).

- [203] Matt Zucker et al. "An optimization approach to rough terrain locomotion". In: *2010 IEEE International Conference on Robotics and Automation*. IEEE. 2010.
- [204] Matt Zucker et al. "Optimization and learning for rough terrain legged locomotion". In: *The International Journal of Robotics Research* (2011).

Appendix A

Chapter 2 Supplementary Material

A.1 Metrics

We use several metrics (in SI units) to evaluate and compare the performance of RMA against baselines:

- Success Rate: Average rate of successfully completing the task as defined in the next section.
- Time to Fall (TTF): Measures the time before a fall. We divide it by the maximum duration of the episode and report a normalized value between 0 and 1.
- Reward: Average step forward reward plus lateral reward over multiple episodes as defined in chapter 2 (RL Rewards) .
- Distance: Average distance covered in an episode. For real-world experiments, we report the normalized distance, where we normalize by the maximum distance which is specific to the task.
- Adaptation Samples: Number of control steps to explore in the testing environment needed for the motor policy to adapt.
- Torque: Squared L2 norm of torques at every joint $\|\boldsymbol{\tau}^t\|^2$.
- Jerk: Squared L2 norm of delta torques $\|\boldsymbol{\tau}^t - \boldsymbol{\tau}^{t-1}\|^2$.
- Ground Impact: Squared L2 norm of delta ground reaction forces at every foot $\|\boldsymbol{f}^t - \boldsymbol{f}^{t-1}\|^2$.

A.2 Additional Training and Deployment Details

The training pipeline is shown in Algorithm 1, and the deployment pipeline is shown in Algorithm 2.

Algorithm 1 RMA Training

Phase 1 Randomly initialize the base policy π ; Randomly initialize the environmental factor encoder μ ; Empty replay buffer D_1 **for** $0 \leq \text{itr} \leq N_{\text{itr}}^1$ **do**

- for** $0 \leq i \leq N_{\text{env}}$ **do**
 - $x_0, e_0 \leftarrow \text{envs}[i].\text{reset}()$ **for** $0 \leq t \leq T$ **do**
 - $z_t \leftarrow \mu(e_t)$ $a_t \leftarrow \pi(x_t, a_{t-1}, z_t)$ $x_{t+1}, e_{t+1}, r_t \leftarrow \text{envs}[i].\text{step}(a_t)$ Store $((x_t, e_t), a_t, r_t, (x_{t+1}, e_{t+1}))$ in D_1

end

Update π and μ using PPO [158] Empty D_1

end

Phase 2 Randomly initialize the adaptation module ϕ parameterized by θ_ϕ ; Empty mini-batch D_2

for $0 \leq \text{itr} \leq N_{\text{itr}}^2$ **do**

- for** $0 \leq i \leq N_{\text{env}}$ **do**
 - $x_0, e_0 \leftarrow \text{envs}[i].\text{reset}()$ **for** $0 \leq t \leq T$ **do**
 - $\hat{z}_t \leftarrow \phi(x_{t-k:k}, a_{t-k-1:k-1})$ $z_t \leftarrow \mu(e_t)$ $a_t \leftarrow \pi(x_t, a_{t-1}, \hat{z}_t)$ $x_{t+1}, e_{t+1}, - \leftarrow \text{envs}[i].\text{step}(a_t)$ Store (\hat{z}_t, z_t) in D_2

end

$\theta_\phi \leftarrow \theta_\phi - \lambda_{\theta_\phi} \nabla_{\theta_\phi} \frac{1}{TN_{\text{env}}} \sum \|\hat{z}_t - z_t\|^2$ Empty D_2

end

We use PPO [158] to train the base policy and the environmental factor encoder. We train for total 15,000 iterations. During each iteration, we collect a batch of 80,000 state-action transitions, which is evenly divided into 4 mini-batches. Each mini-batch is fed into the base policy and the Environment Factor Encoder in sequence for 4 rounds to compute the loss and error back-propagation. The loss is the sum of surrogate policy loss and 0.5 times the value loss. We clip the action log probability ratios between 0.8 and 1.2, and clip the target values to be within the 0.8 – 1.2 times range of the corresponding old values. We exclude the entropy regularization of the base policy, but constrain the standard deviation of the parameterized Gaussian action space to be large than 0.2 to ensure exploration. λ and γ in the generalization advantage estimation [157] are set to 0.95 and 0.998 respectively. We use the Adam optimizer [87], where we set the learning rate to $5e-4$, β to (0.9, 0.999), and ϵ to $1e-8$. The reference implementation can be found in the RaisimGymTorch Library [70].

If we naively train our agent with the reward function aggregating all the terms, it learns to fall because of the penalty terms. To prevent this collapse, we follow the strategy described in [72]. In addition the scaling factors of all reward terms, we apply a small multiplier k_t to the penalty terms 3 – 10, as defined in chapter 2 (RL rewards). We start the training with a very small k_0 set to 0.03, and then exponentially increase the these coefficients using a fixed curriculum: $k_{t+1} = k_t^{0.997}$, where t is the iteration number. The learning process is shown in Figure A.1.

Algorithm 2 RMA Deployment

Process 1 operating at 100 Hz $t \leftarrow 0$ **while not fall do**
 | $a_t \leftarrow \pi(x_t, a_{t-1}, \hat{z}_{\text{async}})$ $x_{t+1} \leftarrow \text{env.step}(a_t)$ $t \leftarrow t + 1$
end

Process 2 operating at 10 Hz **while not fall do**
 | $\hat{z}_{\text{async}} \leftarrow \phi(x_{t-k:k}, a_{t-k-1:k-1})$
end

A.3 Additional Real-World Adaptation Analysis

In addition to the oil-walking experiments in chapter 2, we also analyze the gait patterns and the torque profile for the mass adaptation case, shown in Figure A.2. We throw a payload of 5kg on the back of the robot in the middle of a run and plot the torque profile of the knee, gait pattern, and the 2th and 7th components of the extrinsics vector \hat{z}_t as shown in Figure A.2. We observe that the additional payload disturbs the regular motion of the robot, after which it enters the adaptation phase and finally recovers from the disturbance. When the payload lands on the robot, it is noticeable that the plotted components of the extrinsics vector change in response to the slip. Post adaptation, we see that the torque stabilizes to a higher magnitude than before to account for the payload and the gait time period is roughly recovered.

A.4 Additional Simulation Testings

In Figure A.3, we further test RMA in extreme simulated environments and show its performance in three types of environment variations: the payloads added on the base of the A1 robot, the terrain elevation variation (z-scale used in the fractal terrain generator, details chapter 2 (Simulation Setup)), and the friction coefficient between the robot feet and the terrain. We show the superiority of RMA across all the cases in terms of Success Rate, TTF and Reward as defined in Section A.1.

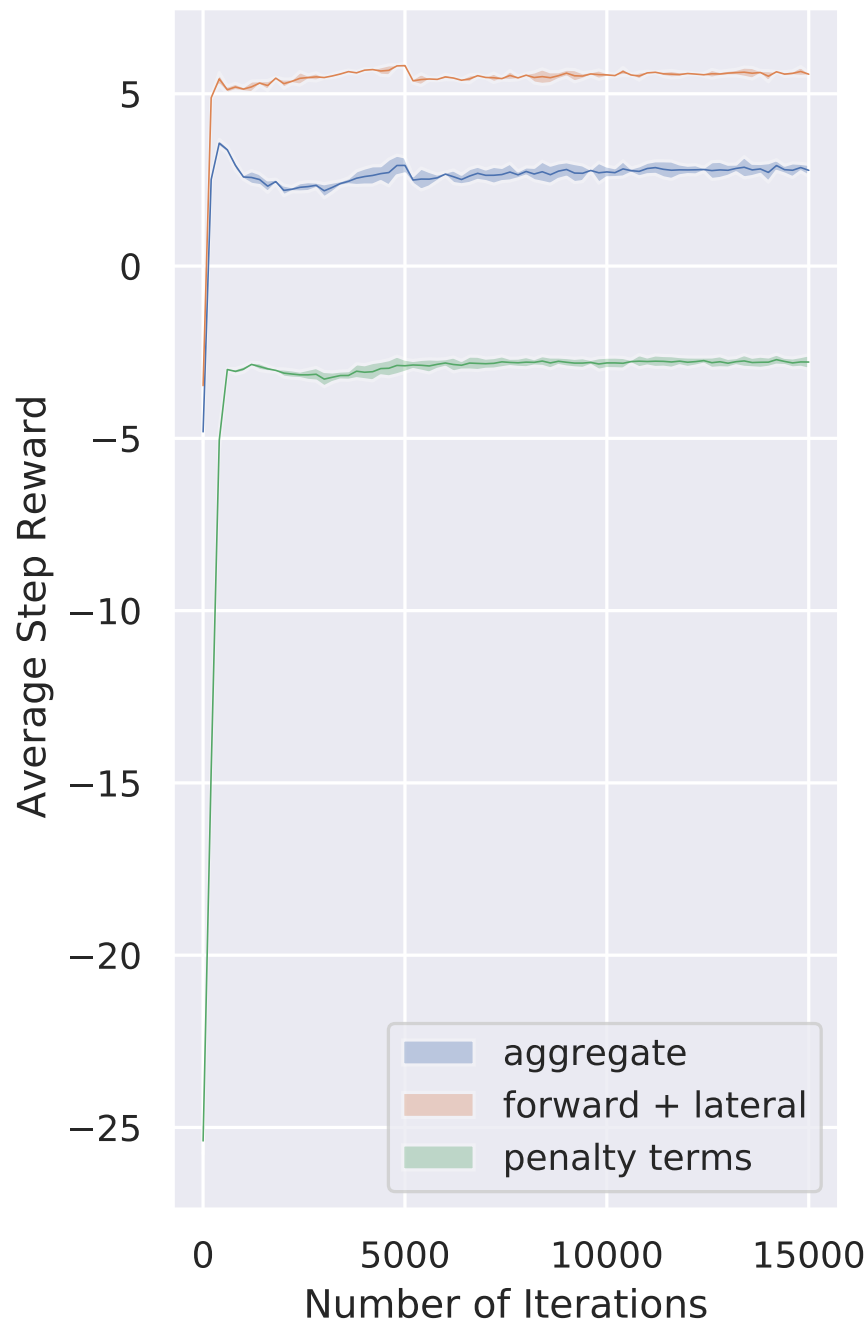


Figure A.1: We plot the average step reward during the total 15,000 training iterations. We show the converging trend of the reward aggregating all reward terms, forward + lateral reward, and sum of penalty terms. It also shows the necessity of applying a small multiplier to the penalty terms at the beginning of training; otherwise, the robot will only have negative experience initially and unable to learn to walk quickly.

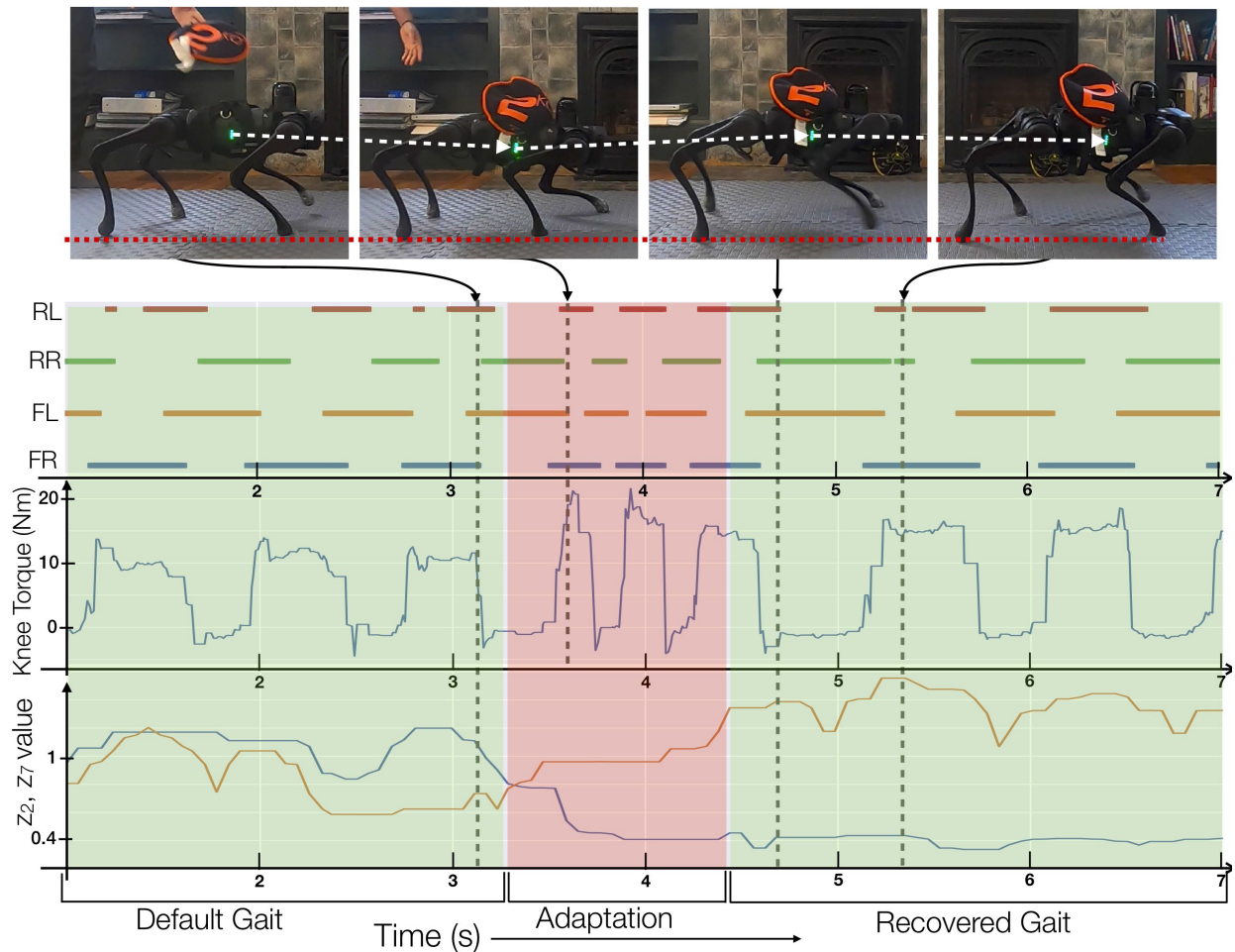


Figure A.2: We analyze the change in behavior of RMA as we throw a payload of 5kg on the back of the robot. As a note, we have flipped the images so that that movement appears from left to right which is why the label on the sandbag appears to be 2Kg. We plot the torque profile of the knee and the gait pattern. The bottom plot shows median filtered 2nd and 7th components of the extrinsics vector \hat{z} predicted by the adaptation module. When the 5kg payload is thrown on the back of the robot, we see a dip in the center of mass of the robot, which the adaptation module subsequently recovers from. In the bottom plot, we see a jump in response in the plotted components of the estimated extrinsics vector, indicating that the additional payload has been detected by the adaptation module. Note that post adaptation, the recovered gait time period is roughly similar to the original, the torque magnitudes have increased and the extrinsics vector continues to capture the presence of the 5Kg payload on the back of the robot.

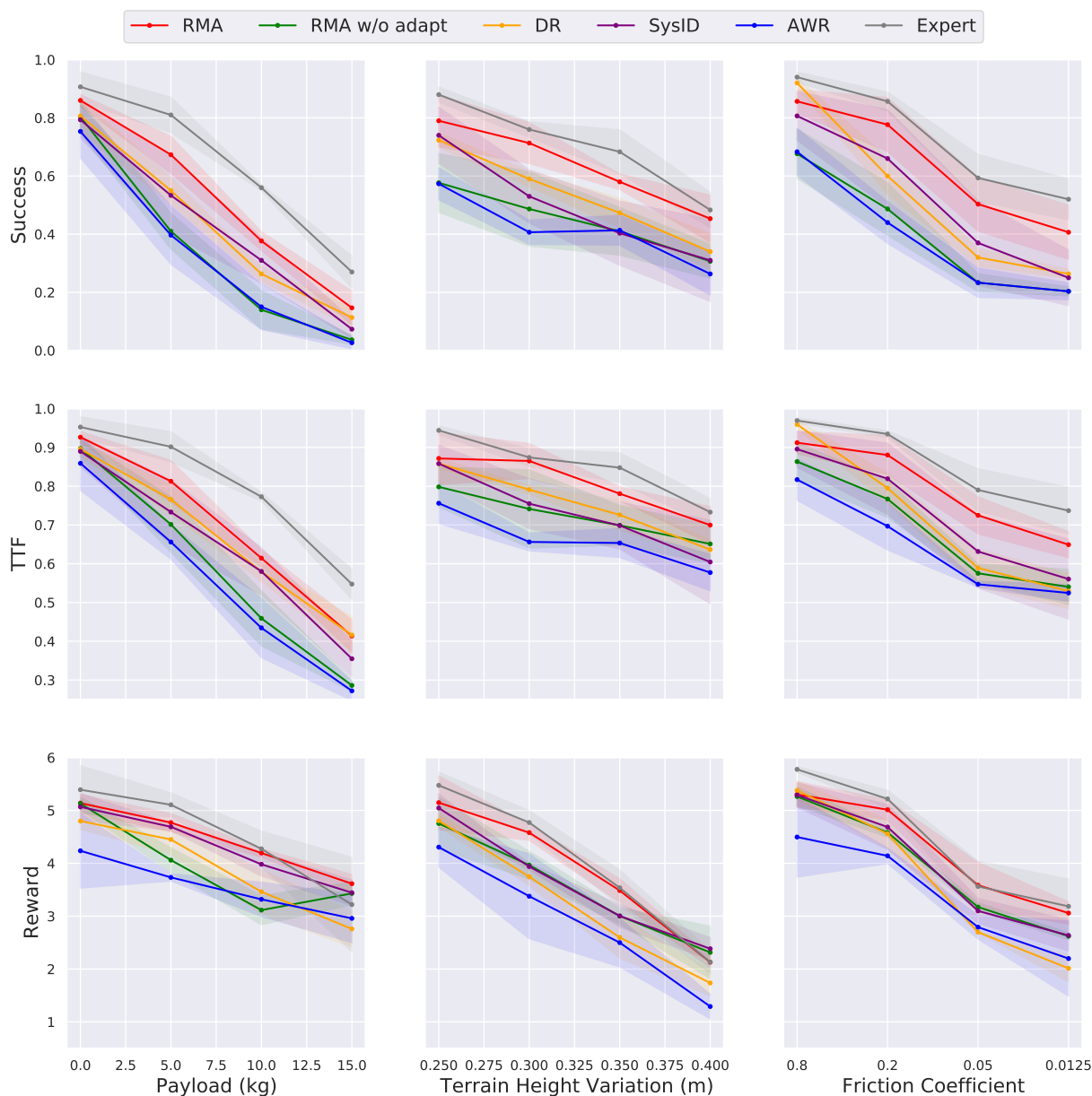


Figure A.3: **Simulation Generalization Results:** We further compare the generalization performance of our method to baseline methods in simulation. We pick three physics parameters that may vary to a large degree in the real world: the payload on robot, the terrain height variation, and the friction coefficient between the robot feet and the terrain. We set other environment parameters according to the training range as described in chapter 2. Baselines and metrics are defined in Section A.1. We report the mean and standard deviation of the performance of 3 randomly initialized policies, which is characterized by the average of 100 testing trials in given settings. Despite no testing environment samples, RMA performs the best, the closest to Expert’s performance. For reference, A1 robot without additional payloads weighs 12 kg, and is 0.35 m tall. The static friction coefficient between rubber and concrete is 1.00.

Appendix B

Chapter 3 Supplementary Material

B.1 Experimental Setup and Training Details

Hardware Details: We use Unitree’s A1 robot as our hardware platform [177]. A1 is a medium sized quadruped with 18 DoFs out of which 12 are actuated. Its mass is about 12 kg. We measure the joint positions and velocities of the 12 joints from the motor encoders and roll and pitch angles from the IMU sensor. To sense the foot contacts with ground in the real world, the foot sensor readings are taken on-board. The feet of the robot are deformable rubber membranes with filled air and air pressure sensors attached. When the foot membranes are deformed, readings from the pressure sensor will increase. We binarize the pressure reading by setting a threshold. Whenever the pressure reading from a foot is above that threshold, we treat it as a foot contact. The deployed policy uses position control for the joints of the robots. We use a PD controller to convert target joint position to torque with fixed gains ($K_p = 55$ and $K_d = 0.8$).

Simulation Setup: We use the A1 URDF [177] to simulate the A1 robot in the RaiSim simulator [71]. We generate complex terrains using the inbuilt fractal terrain generator (`flat terrain` for structured gait emergence: number of octaves = 2, fractal lacunarity = 2.0, fractal gain = 0.25, frequency = 10Hz, amplitude = 0.23m; `uneven terrain` for unstructured gait emergence: number of octaves = 2, fractal lacunarity = 2.0, fractal gain = 0.25, frequency = 20Hz, amplitude = 0.27m). We simulate each episode for a maximum of 1000 steps and terminate the episode earlier if the height of the robot drops below 0.28m, magnitude of the body roll exceeds 0.4 radians or the pitch exceeds 0.2 radians. The control frequency of the policy is 100Hz, and the simulation time step is 0.025s.

Environmental Variations: All environmental variations with their ranges are listed in chapter 3. The environment information of e_t in RMA [96] includes center of mass position and the payload (3 dimensions), motor strength (12 dimensions), friction (1 dimension), linear speed in x direction v_x (1 dimension), linear speed in y direction v_y (1 dimension) and yaw speed ω_{yaw} (1 dimension),

making it a 19-dim vector. We smooth v_x , v_y and ω_{yaw} by using exponential averaging with values of history time steps and a smoothing factor of 0.2.

State Space: The state is 30 dimensional containing the joint positions (12 dimensions), joint velocities (12 dimensions), roll and pitch angles of the torso, and binarized foot contact indicators (4 dimensions). We did not use any classical state estimator to measure the base velocity or orientation.

Action Space: The action space is 12 dimensional corresponding to the target joint position for the 12 robot joints. The speed up policy learning, the policy network outputs the delta target joint positions, which are converted to target joint positions by adding the joint angles of the initial standing state. The delta target joint positions at HAA, HFE and KFE are restricted to $[-0.15, 0.15]$, $[-0.4, 0.4]$ and $[-0.4, 0.4]$ in radians respectively. The predicted 12-dim target joint angles $a = \hat{\mathbf{q}}$ is converted to torques τ using a PD controller: $\tau = K_p (\hat{\mathbf{q}} - \mathbf{q}) + K_d (\hat{\dot{\mathbf{q}}} - \dot{\mathbf{q}})$. K_p and K_d are manually-specified gains, and the target joint velocities $\hat{\dot{\mathbf{q}}}$ are set to 0.

Reward: We use the energy consumption-based reward throughout all settings. On uneven terrain for unstructured gaits, we add the same extra natural penalties in [96] to explicitly minimize ground impacts.

Hyperparameters of Policy Learning: PPO [158] and Adam optimizer [87] are used to train the base policy and the environmental factor encoder in RMA [96]. The total number of training epoch is 15,000. During each epoch, a batch of 100,000 state-action transitions, which is split into 4 mini-batches, is sampled from the current policy. Each mini-batch is used for 4 times to compute the loss and backpropagation. The total loss is the surrogate policy loss plus half the value loss. The action log probability ratio is clipped between 0.8 and 1.2, and the target value is clipped to 0.8 – 1.2 times the range of the value that is computed in previous iteration. We lower bound the standard deviation of the parameterized Gaussian action space to 0.2 to encourage exploration. We set λ and γ in the generalization advantage estimation [157] to 0.95 and 0.998 respectively. We set the learning rate of the optimizer to $5e-4$, β to (0.9, 0.999), and ϵ to $1e-8$.

B.2 Details of the MPC baseline

We use a reference implementation [31] of convex MPC [38] for Unitree A1. The parameters for gait generation to enforce constraints on ground reaction forces are fine-tuned for walking gait, trotting gait and bouncing gait. For the walking gait, we test at target speeds v^{target} in the range from 0.1m/s to 0.7m/s every 0.1m/s. We set the duty factors of 4 legs to $[0.8, 0.8, 0.8, 0.8]$, initial leg phases to $[0, 0.25, 0.5, 0]$, and the stance duration to $-0.5v^{\text{target}} + 0.75$ seconds. Only the Rear-Left foot is initialized in swing phase. For the trotting gait, we test at target speeds v^{target} in the range from 0.5m/s to 1.5m/s every 0.1m/s. We set the duty factors of 4 legs to $[0.6, 0.6, 0.6, 0.6]$, initial leg phases to $[0.9, 0, 0, 0.9]$, and the stance duration to $-0.15v^{\text{target}} + 0.325$ seconds. The

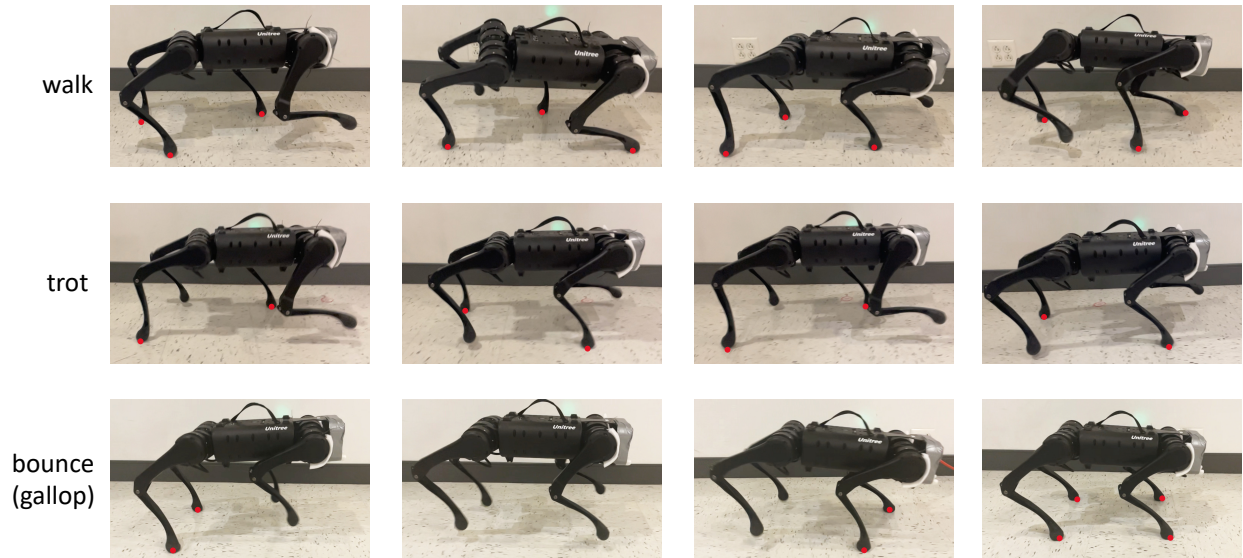


Figure B.1: Key frames of the walking gait, trotting gait and bouncing (galloping) gait. Red dots indicate foot contacts.



Figure B.2: Walking with 1 kg payload (two bottles of 500 ml water are strapped on the back of the robot). The 1 kg payload is out of the normal perturbation of environment parameters used in simulation training.

Front-Right and Rear-Left feet are initialized in swing phase. For the bouncing gait, we test at target speeds v^{target} in the range from 1.0m/s to 2.0m/s every 0.1m/s. We set the duty factors of 4 legs to $[0.35, 0.35, 0.35, 0.35]$, initial leg phases to $[0, 0, 0, 0]$, and the stance duration to 0.04 seconds. All feet are initialized to the stand phase.

B.3 Additional Experiment Results

Emergence of Walking, Trotting and Bouncing

In Figure B.1 of the Supplementary, we show four key frames of the walking at low target speed (0.375 m/s), trotting at median target speed (0.9 m/s) and bouncing (galloping) gaits at high target speed (1.5 m/s). At high speed, the robot gallops for the first few seconds, where the front two legs leave the ground first and also contact the ground first after the flight phase. Then, it switches to using bouncing gait where the four legs hit the ground at approximately the same time.

Robustness Tests

In addition to Figure 5 of the main text, Figure B.2 in the Supplementary shows the the key frames of walking gait under 1 kg payload. The walking gait is robust to the extra payload that is out of the normal perturbation of environment parameters used in simulation training.

Appendix C

Chapter 4 Supplementary Material

C.1 Proof of Theorem 3.1

Theorem. $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, R, \gamma)$ be an MDP with state space \mathcal{S} , action space \mathcal{A} , transition function $P : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$, reward function $R : \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$ and discount factor γ . Let $V^1(s)$ be the value function of the phase 1 policy that is trained to be close to optimal value function $V^*(s)$, i.e., $|V^*(s) - V^1(s)| < \epsilon$ for all $s \in \mathcal{S}$, and $\pi^1(s)$ be the greedy phase 1 policy obtained from $V^1(s)$. Suppose the phase 2 policy operates in a different state space \mathcal{S}' given by an invertible mapping $f : \mathcal{S} \rightarrow \mathcal{S}'$. If the phase 2 policy is close to phase 1 $|\pi^1(s) - \pi^2(f(s))| < \eta \forall s$ and R, P are Lipschitz continuous, then the return of phase 2 policy is close to optimal everywhere, i.e., for all s ,

$$\left| V^*(s) - V^{\pi^2}(f(s)) \right| < \frac{2\epsilon\gamma + \eta c}{1 - \gamma}$$

where $c \propto \sum_{s \in \mathcal{S}} V^*(s)$ is a large but bounded constant.

Proof. Since the function f maps the state spaces $\mathcal{S}, \mathcal{S}'$, we can assume for convenience that both policies π^1, π^2 operate in the same state space \mathcal{S} . To obtain an action for $s' \in \mathcal{S}'$, we can simply query $\pi^2(f^{-1}(s'))$. Assume that the reward function R and transition function P are Lipschitz

$$|R(s_t, a_t) - R(s_t, a'_t)| \leq L_R |a_t - a'_t| \quad (\text{C.1})$$

$$|P(s_{t+1} | s_t, a_t) - P(s_{t+1} | s_t, a'_t)| \leq L_P |a_t - a'_t| \quad (\text{C.2})$$

for all states s_t, s_{t+1} and actions a_t, a'_t . We generalize the structure of proof for the upper bound on the distance in approximate optimal-value functions [163, 16] to the teacher-student setting. Let s_0 be the point where the distance between V^* and V^{π^S} is maximal

$$s_0 = \operatorname{argmax}_s V^*(s) - V^{\pi^S}(s) \quad (\text{C.3})$$

Let Q^T be the Q-function corresponding to V^T . π^T is obtained by maximizing $Q^T(s, a)$ over actions $\pi^T(s) = \operatorname{argmax}_a Q^T(s, a)$. Note that in general the value function V^{π^T} may be different from V^T .

Let a^* be the action taken by the optimal policy at state s_0 , while $a^T = \pi^T(s_0)$ be the action taken by the teacher. Then, the return of the teacher's greedy action a^T must be highest under teacher's value function Q^T ,

$$Q^T(s_0, a^*) \leq Q^T(s_0, a^T) \quad (\text{C.4})$$

We can expand each side of (C.4) above to get

$$R(s_0, a^*) + \gamma \sum_{s \in \mathcal{S}} P(s | s_0, a^*) V^T(s) \leq R(s_0, a^T) + \gamma \sum_{s \in \mathcal{S}} P(s | s_0, a^T) V^T(s) \quad (\text{C.5})$$

Notice that $|V^*(s) - V^T(s)| < \epsilon$ implies that $V^T(s) \in [V^*(s) - \epsilon, V^*(s) + \epsilon]$ which we can plug into the inequality above to get

$$\begin{aligned} & R(s_0, a^*) - R(s_0, a^T) \\ & \leq \gamma \sum_{s \in \mathcal{S}} [P(s | s_0, a^T) V^*(s) - P(s | s_0, a^*) V^*(s) + \epsilon (P(s | s_0, a^T) + P(s | s_0, a^*))] \\ & \leq \gamma \sum_{s \in \mathcal{S}} [P(s | s_0, a^T) V^*(s) - P(s | s_0, a^*) V^*(s)] + 2\epsilon\gamma \end{aligned} \quad (\text{C.6})$$

We can now write a bound for $V^*(s_0) - V^{\pi^S}(s_0)$. Let a^S be the action taken by the student policy at state s_0 . Then,

$$\begin{aligned} & V^*(s_0) - V^{\pi^S}(s_0) \\ & = R(s_0, a^*) - R(s_0, a^S) + \gamma \sum_{s \in \mathcal{S}} P(s | s_0, a^*) V^*(s) - P(s | s_0, a^S) V^{\pi^S}(s) \quad (\text{substitute (C.1)}) \\ & \leq R(s_0, a^*) - R(s_0, a^T) + L_R |a^S - a^T| + \gamma \sum_{s \in \mathcal{S}} P(s | s_0, a^*) V^*(s) - P(s | s_0, a^S) V^{\pi^S}(s) \\ & \leq R(s_0, a^*) - R(s_0, a^T) + L_R \eta + \gamma \sum_{s \in \mathcal{S}} P(s | s_0, a^*) V^*(s) - P(s | s_0, a^S) V^{\pi^S}(s) \end{aligned}$$

We can now use (C.6) to write

$$\begin{aligned} & V^*(s_0) - V^{\pi^S}(s_0) \\ & \leq 2\epsilon\gamma + L_R \eta + \gamma \sum_{s \in \mathcal{S}} P(s | s_0, a^T) V^*(s) - P(s | s_0, a^S) V^{\pi^S}(s) \quad (\text{substitute (C.2)}) \\ & \leq 2\epsilon\gamma + L_R \eta + \gamma \sum_{s \in \mathcal{S}} P(s | s_0, a^S) (V^*(s) - V^{\pi^S}(s)) + \eta\gamma L_P \sum_{s \in \mathcal{S}} V^*(s) \\ & = 2\epsilon\gamma + L_R \eta + \gamma (V^*(s) - V^{\pi^S}(s)) + \eta\gamma L_P \sum_{s \in \mathcal{S}} V^*(s) \\ & \quad (\text{since } s_0 \text{ is the state with highest difference between } V^* \text{ and } V^{\pi^S}) \\ & \leq 2\epsilon\gamma + L_R \eta + \gamma (V^*(s_0) - V^{\pi^S}(s_0)) + \eta\gamma L_P \sum_{s \in \mathcal{S}} V^*(s) \end{aligned}$$

Rearranging yields,

$$V^*(s_0) - V^{\pi^S}(s_0) \leq \frac{2\epsilon\gamma + \eta (L_R + \gamma L_P \sum_{s \in \mathcal{S}} V^*(s))}{1 - \gamma}$$

Since s_0 is the state at which the difference between V^* and V^{π^S} is maximal, we can claim

$$V^*(s) - V^{\pi^S}(s) \leq \frac{2\epsilon\gamma + \eta c}{1 - \gamma}$$

for all states $s \in \mathcal{S}$, where $c = (L_R + \gamma L_P \sum_{s \in \mathcal{S}} V^*(s))$ □

C.2 Rewards

Previous work [96, 48] has shown that task agnostic energy minimization based rewards can lead to the emergence of stable and natural gaits that obey high-level commands. We use this same basic reward structure along with penalties to prevent behavior that can damage the robot on complex terrain. Now onwards, we omit the time subscript t for simplicity.

- *Absolute work penalty* $-|\boldsymbol{\tau} \cdot \mathbf{q}|$ where $\boldsymbol{\tau}$ are the joint torques. We use the absolute value so that the policy does not learn to get positive reward by exploiting inaccuracies in contact simulation.
- *Command tracking* $v_x^{\text{cmd}} - |v_x^{\text{cmd}} - v_x| - |\omega_z^{\text{cmd}} - \omega_z|$ where v_x is velocity of robot in forward direction and ω_z is yaw angular velocity (x, z are coordinate axes fixed to the robot).
- *Foot jerk penalty* $\sum_{i \in \mathcal{F}} \|\mathbf{f}_t^i - \mathbf{f}_{t-1}^i\|$ where \mathbf{f}_t^i is the force at time t on the i^{th} rigid body and \mathcal{F} is the set of feet indices. This prevents large motor backlash.
- *Feet drag penalty* $\sum_{i \in \mathcal{F}} \mathbb{I}[f_z^i \geq 1\text{N}] \cdot (|v_x^i| + |v_y^i|)$ where \mathbb{I} is the indicator function, and v_x^i, v_y^i is velocity of i^{th} rigid body. This penalizes velocity of feet in the horizontal plane if in contact with the ground preventing feet dragging on the ground which can damage them.
- *Collision penalty* $\sum_{i \in \mathcal{C} \cup \mathcal{T}} \mathbb{I}[\mathbf{f}^i \geq 0.1\text{N}]$ where \mathcal{C}, \mathcal{T} are the set of calf and thigh indices. This penalizes contacts at the thighs and calves of the robot which would otherwise graze against edges of stairs and discrete obstacles.
- *Survival bonus* constant value 1 at each time step to prioritize survival over following commands in challenging situations.

The scales for these are $-1\text{e-}4, 7, -1\text{e-}4, -1\text{e-}4, -1, 1$. Notice that these reward functions do not define any gait priors and the optimal gait is allowed emerge via RL. Target heading values h_t^{cmd} are sampled and commanded angular velocities is computed as $(\omega_z^{\text{cmd}})_t = 0.5 \cdot (h_t^{\text{cmd}} - h_t)$ where h_t is the current heading value. When walking on terrain, $(v_x^{\text{cmd}})_t = 0.35\text{m/s}$ and heading is varied in $h_t^{\text{cmd}} \in [-10^\circ, 10^\circ]$. On flat ground, one of three sampling modes are chosen uniformly at random - curve following, in-place turning and complete stop. In the curve following regime $(v_x^{\text{cmd}})_t \in [0.2\text{m/s}, 0.75\text{m/s}]$ while $h_t^{\text{cmd}} \in [-60^\circ, 60^\circ]$. For in-place turning, $(v_x^{\text{cmd}})_t = 0$ and $h_t^{\text{cmd}} \in [-180^\circ, 180^\circ]$. In complete stop, $(v_x^{\text{cmd}})_t = 0, h_t^{\text{cmd}} = 0$. This scheme is designed to mimic the distribution of commands the robot sees during operation. We terminate if the pitch exceeds 90° or if the base or head of the robot collides with an object.

Observation	a	b	σ
Joint angles (left hips)	1.0	0.1	0.01
Joint angles (right hips)	1.0	-0.1	0.01
Joint angles (front thighs)	1.0	0.8	0.01
Joint angles (rear thighs)	1.0	1.0	0.01
Joint angles (calves)	1.0	-1.5	0.01
Joint velocity	0.05	0.0	0.05
Angular velocity	0.25	0.0	0.05
Orientation	1.0	0.0	0.02
Scandots height	5.0	0.0	0.07
Scandots horizontal location	–	–	0.01

Table C.1: During training, ground truth observations \mathbf{o}_t are shifted, normalized and noised to get observations \mathbf{o}'_t which are passed to the policy. $\mathbf{o}'_t = a(\mathbf{o}_t - b) + \epsilon$ where $\epsilon \sim \mathcal{N}(0, \sigma)$. We tabulate a, b, σ for each kind of observation above. a, b values for scandots horizontal locations are blank since these locations are fixed with respect to the robot and not passed to the policy.

C.3 Experimental Setup and Implementation Details

Hardware

We use the Unitree A1 robot. The robot has 12 actuated joints, 3 per leg at hip, thigh and calf joints. The robot has a front-facing Intel RealSense depth camera in its head. The compute consists of a small GPU (Jetson NX) capable of ≈ 0.8 TFLOPS and an UPboard with Intel Quad Core Atom X5-8350 containing 4GB ram and 1.92GHz clock speed. The UPboard and Jetson are on the same local network. Since depth processing is an expensive operation we run the convolutional backbone on the Jetson’s GPU and send the depth latent over a UDP socket to the UPboard which runs the base policy. The policy operates at 50Hz and sends joint position commands which are converted to torques by a low-level PD controller running at 400Hz with stiffness $K_p = 40$ and damping $K_d = 0.5$.

Simulation Setup We use the IsaacGym (IG) simulator with the `legged_gym` library [152] to develop walking policies. IG can run physics simulation on the GPU and has a throughput of around $2e5$ time-steps per second on a Nvidia RTX 3090 during phase 1 training with 4096 robots running in parallel. For phase 2, we can render depth using simulated cameras calibrated to be in the same position as the real camera on the robot. Since depth rendering is expensive and memory intensive, we get a throughput of 500 time-steps per second with 256 parallel environments. We run phase 1 for 15 billion samples (13 hours) and phase 2 for 6 million samples (6 hours).

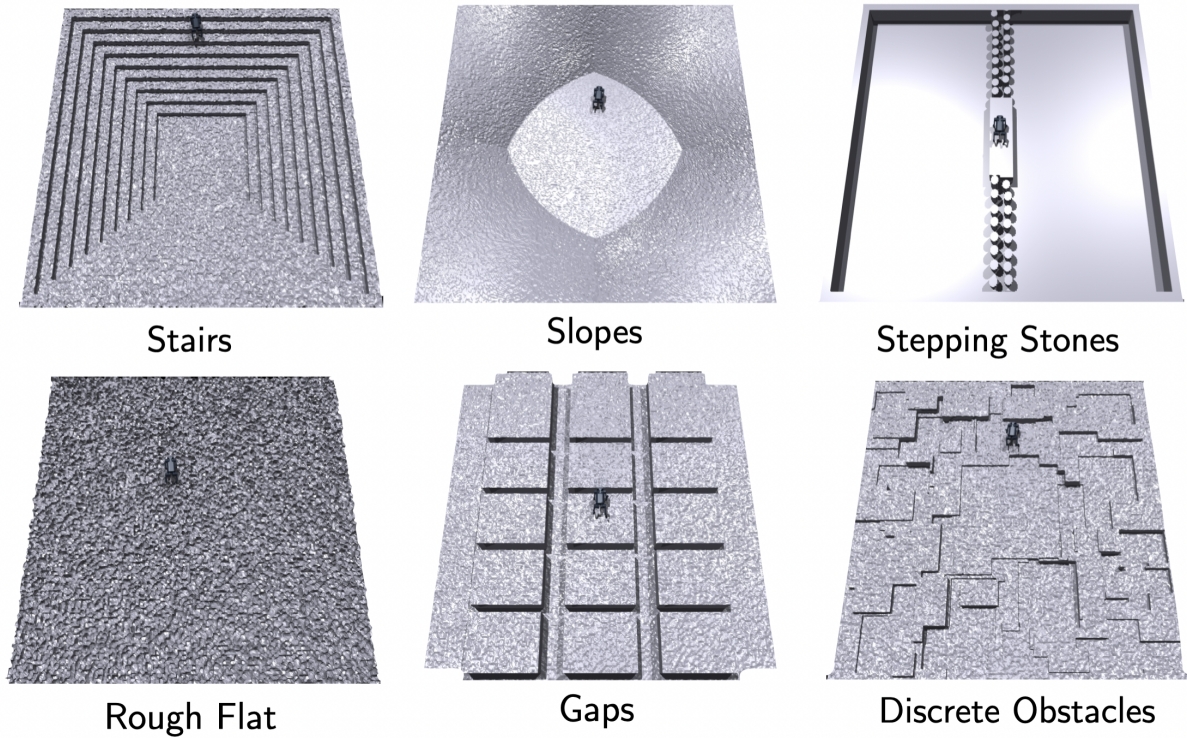


Figure C.1: Set of terrain we use during training

Name	Range
Height map update frequency*	[80ms, 120ms]
Height map update latency*	[10ms, 30ms]
Added mass	[-2kg, 6kg]
Change in position of COM	[-0.15m, 0.15m]
Random pushes	Every 15s at 0.3m/s
Friction coefficient	[0.3, 1.25]
Height of fractal terrain	[0.02m, 0.04m]
Motor Strength	[90%, 110%]
PD controller stiffness	[35, 45]
PD controller damping	[0.4, 0.6]

Figure C.2: Parameter randomization in simulation. * indicates that randomization is increased to this value over a curriculum.

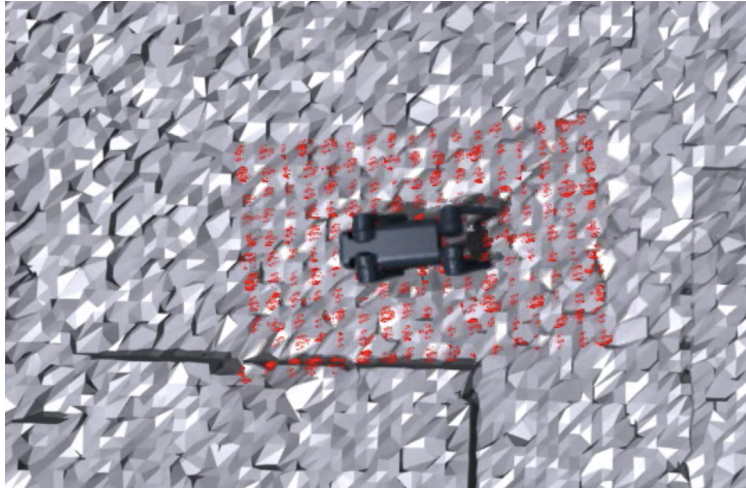


Figure C.3: The privileged baseline receives terrain information from all around the robot including from around the hind feet.

Environment We construct a large elevation map with 100 sub-terrains arranged in a 20×10 grid. Each row has the same type of terrain arranged in increasing difficulty while different rows have different terrain. Each terrain has a length and width of 8m. We add high fractals (upto 10cm) on flat terrain while medium fractals (4cm) on others. Terrains are shown in Figure C.1 with randomization ranges described in Table C.2.

Policy architecture The elevation map compression module β consists of an MLP with 2 hidden layers. The GRUs G^1, G^2 are single layer while the feed-forward networks F^1, F^2 have two hidden layers with ReLU non-linearities. The convolutional depth backbone γ consists of a series of 2D convolutions and max-pool layers.

Appendix D

Chapter 5 Supplementary Material

D.1 Additional Results and Analysis

Per-Object Result Result Analysis.

In Figure D.1, we show an almost linear correlation between object mass and the average torque commanded by our policy. This indicates our policy behave differently for objects with different weights and commands smaller torque when object is lighter for energy efficiency.

In Section 5.3, we show the average performance over multiple challenging objects. To give more insights about the working and failure cases of our approach, we analyze the performance on each of the 33 objects we used in experiments. The results are shown in Figure D.2, Figure D.3, and Figure D.4.

Our policy can achieve almost perfect stable and continuous in-hand rotation in 22 out of 33 objects. This set includes objects with different scale, mass, coefficient of frictions, and shapes. Notably, for objects with very high center of mass (the plastic bottle and paper towel), the policy can still perform stable and dynamic in-hand rotations. For more challenging objects including cubes, heavy object with larger aspect ratio (Pear and Kiwi Fruit), and small objects, our policy can still perform about 10 to 20 seconds in-hand rotations. The failure cases are mostly object falling from the fingertips because of incorrect contact positions.

Multi-Axis In-Hand Object Rotation

We explore the possibility of using our approach to do multi-axis in-hand object rotation. We qualitatively demonstrate preliminary results in the project website. We find this is a harder task than single-axis training and needs about $1.5\times$ training time. Our policy achieves a smooth gait transition between rotation over different axes.

D.2 Ablation Experiments

We analyze different design choices for our method in simulation.

Training with Different Randomization Parameters. Our approach applies a large range of physical randomization during training. In Table D.1, we compare the performance difference with our method but without physical randomization.

The *No Rand* entry in Table D.1 use the same reward as our method. However, we do not apply any physical randomization during training. The object scale, mass, coefficient of friction, and other physical properties are fixed. This method performs much worse than our method with physical randomization, especially in the out-of-distribution case.

Method	Within Training Distribution				Out-of-Distribution			
	RotR (\uparrow)	TTF (\uparrow)	ObjVel (\downarrow)	Torque (\downarrow)	RotR (\uparrow)	TTF (\uparrow)	ObjVel (\downarrow)	Torque (\downarrow)
No Rand	171.96 \pm 20.63	0.63 \pm 0.07	0.48 \pm 0.07	1.94 \pm 0.32	88.02 \pm 14.27	0.37 \pm 0.06	0.98 \pm 0.16	2.26 \pm 0.29
Ours	222.27 \pm 21.20	0.82 \pm 0.02	0.29 \pm 0.05	1.20 \pm 0.19	160.60 \pm 10.22	0.68 \pm 0.07	0.47 \pm 0.07	1.20 \pm 0.17

Table D.1: We report the performance of our method without physical randomization during training (the *No Rand* entry). It performs much worse than our method, especially in the out-of-distribution case.

Length of Proprioceptive History. In Table D.2, we compare and report the performance with different proprioceptive history length during training the adaptation module. We report the performance a history length 10, 20, and 30. We observe the performance keeps increasing when using a longer history, but saturates at about 30. Considering the efficiency during inference time, we decide to use T=30 in final experiments.

Method	Within Training Distribution				Out-of-Distribution			
	RotR (\uparrow)	TTF (\uparrow)	ObjVel (\downarrow)	Torque (\downarrow)	RotR (\uparrow)	TTF (\uparrow)	ObjVel (\downarrow)	Torque (\downarrow)
Ours-T10	215.81 \pm 17.55	0.80 \pm 0.02	0.30 \pm 0.04	1.19 \pm 0.16	150.58 \pm 7.07	0.61 \pm 0.05	0.51 \pm 0.08	1.18 \pm 0.14
Ours-T20	220.46 \pm 19.72	0.82 \pm 0.02	0.30 \pm 0.04	1.21 \pm 0.17	157.22 \pm 8.11	0.64 \pm 0.04	0.48 \pm 0.07	1.20 \pm 0.15
Ours-T30	222.27 \pm 21.20	0.82 \pm 0.02	0.29 \pm 0.05	1.20 \pm 0.19	160.60 \pm 10.22	0.68 \pm 0.07	0.47 \pm 0.07	1.20 \pm 0.17

Table D.2: We compare the effect of using different proprioceptive history length during training the adaptation module. Our approach is not sensitive to this hyperparameter. The performance increases when we increase the history length but saturates when the history length is long enough at T=30.

Additional Baseline for Robust Domain Randomization (DR). In the main text, the Robust Domain Randomization (DR) baseline only receives \mathbf{o}_t as the input, which contains the robot joint positions and actions in the past three timesteps. In Table D.3, we study the effect of including longer robot observations.

Method	Within Training Distribution				Out-of-Distribution			
	RotR (\uparrow)	TTF (\uparrow)	ObjVel (\downarrow)	Torque (\downarrow)	RotR (\uparrow)	TTF (\uparrow)	ObjVel (\downarrow)	Torque (\downarrow)
Expert	233.71 \pm 25.24	0.85 \pm 0.01	0.28 \pm 0.05	1.24 \pm 0.19	165.07 \pm 15.65	0.71 \pm 0.04	0.42 \pm 0.06	1.24 \pm 0.16
DR-MLP-T3	176.12 \pm 26.47	0.81 \pm 0.02	0.34 \pm 0.05	1.42 \pm 0.06	140.80 \pm 17.51	0.63 \pm 0.02	0.64 \pm 0.06	1.48 \pm 0.20
DR-MLP-T5	165.26 \pm 30.66	0.76 \pm 0.04	0.37 \pm 0.05	1.26 \pm 0.09	115.13 \pm 16.59	0.57 \pm 0.06	0.59 \pm 0.08	1.27 \pm 0.08
DR-MLP-T10	140.95 \pm 13.66	0.72 \pm 0.06	0.39 \pm 0.09	1.52 \pm 0.41	98.32 \pm 10.79	0.54 \pm 0.05	0.57 \pm 0.06	1.51 \pm 0.39
DR-MLP-T20	42.86 \pm 3.66	0.20 \pm 0.02	0.62 \pm 0.03	1.84 \pm 0.19	60.21 \pm 5.19	0.34 \pm 0.05	0.82 \pm 0.09	1.95 \pm 0.20
DR-MLP-T30	49.83 \pm 32.35	0.31 \pm 0.19	1.32 \pm 1.04	1.52 \pm 0.41	36.33 \pm 20.21	0.24 \pm 0.14	1.75 \pm 1.28	2.03 \pm 0.40
DR-LSTM-T10	115.28 \pm 32.99	0.56 \pm 0.15	0.49 \pm 0.08	2.00 \pm 0.45	73.60 \pm 23.88	0.38 \pm 0.12	0.77 \pm 0.19	1.91 \pm 0.41
DR-LSTM-T20	94.33 \pm 33.78	0.44 \pm 0.10	0.57 \pm 0.08	1.93 \pm 0.28	60.18 \pm 17.85	0.28 \pm 0.06	0.91 \pm 0.15	1.85 \pm 0.27
DR-LSTM-T30	75.61 \pm 9.30	0.36 \pm 0.04	0.62 \pm 0.12	1.91 \pm 0.25	48.32 \pm 4.81	0.24 \pm 0.03	1.04 \pm 0.18	1.84 \pm 0.22
Ours	222.27\pm21.20	0.82\pm0.02	0.29\pm0.05	1.20\pm0.19	160.60\pm10.22	0.68\pm0.07	0.47\pm0.07	1.20\pm0.17

Table D.3: We compare the domain randomization baseline with extended temporal input. However, we find that both the MLP and LSTM networks suffer optimization difficulty. The performance decreases when the input contains longer observation sequences.

To fuse robot observations from multiple steps, we consider two different architectures. In MLP, we flatten and concatenate the observations over the time dimension and directly feed it into the policy network. We also explore to use an LSTM network to capture the temporal correlation of input observations. The input will first pass through an LSTM network and then feed into the policy network. The results are shown in Table D.3.

We find that the MLP network is hard to optimize via PPO when temporal length is higher. We see a clear trend of decreasing performance when we increase the temporal length from 3 to 30. LSTM performs better than MLP when the temporal length is larger but still suffer the optimization difficulty.

We also consider a temporal convolution network but find it cannot be successfully optimize and only produce random policies.

D.3 Implementation Details

Physical Randomization Parameter. We apply domain randomization during training the base policy as well as the adaptation module. The parameters are listed in Table D.4 (Train Range). During simulation evaluation, we use a larger randomization range to test the performance of out-of-distribution generalization.

We use cylindrical objects for training. The cylindrical objects has a radius of 8 cm and a height sampled from the following discretized list: [0.8, 0.85, 0.9, 0.95, 1.0, 1.05, 1.1, 1.15, 1.2] cm. The whole object will be scaled by the randomized object scale parameter specified in Table D.4.

Following [129], we apply a random disturbance force to the object during training. The force scale is $2m$ where m is the object mass. The force is decayed by 0.9 every 80 ms following [129]. The force is re-sampled at each time-step with a probability 0.25. During out-of-distribution testing, we increase the force scale to be $4m$. We also add a noise to the joint position sampled from a uniform distribution $\mathcal{U}(0, 0.005)$ to make it more robust to the real-world noisy joint readings.

Parameter	Train Range	Test Range
Object Shape	Cylindrical	Cylindrical, Cube, and Sphere
Object Scale	[0.70, 0.86]	[0.66, 0.90]
Mass	[0.01, 0.25] kg	[0.01, 0.30] kg
Center of Mass	[-1.00, 1.00] cm	[-1.25, 1.25] cm
Coefficient of Friction	[0.3, 3.0]	[0.2, 3.5]
External Disturbance	(2, 0.25)	(4, 0.25)
PD Controller Stiffness	[2.9, 3.1]	[2.6, 3.4]
PD Controller Damping	[0.09, 0.11]	[0.08, 0.12]

Table D.4: Randomization Range of Physics Parameters. We sample the value of Object Scale, Mass, Center of Mass, Coefficient of Friction, Stiffness, and Damping from a uniform distribution where the lower and upper values are specified in the table. The specification of External Disturbance is specified in the text.

We also include 10% sphere objects and 10% cubes in the out-of-distribution evaluation. The canonical sphere object is of diameter 8 cm and the cube object is of side length 8 cm. They are also scaled by the randomized scale parameter.

Stable Precision Grasp Generation. Our approach assumes the object is grasped at the beginning of the episode. To achieve this, we start from a canonical grasping position using fingers. Then we add a relative offset to the joint positions sampled from $\mathcal{U}(-0.25, 0.25)$ rad. Then we forward the simulation by 0.5 s. We save the grasping pose if all the following conditions are satisfied:

1. The distance between the fingertip and the object should be smaller than 10 cm.
2. At least two fingers are in contact with the object.
3. The object’s height should be above 14.5 cm higher than the center of the palm.

In practise, we discretized (each region is separated by 0.2) the scales specified in Table D.4 and pre-sampled 50,000 grasping poses for the robot hand and the object for each scale.

Reward Definition. Our reward function (subscript t omitted for simplicity) has the following component:

1. $r_{rot} \doteq \max(\min(\boldsymbol{\omega} \cdot \hat{\mathbf{k}}, r_{\max}), r_{\min})$ is the rotation reward. $\hat{\mathbf{k}}$ is a desired rotation axis in the world coordinate. In the experiment, we use $\hat{\mathbf{k}} = [0, 0, 1]^\top$. We use $r_{\max} = 0.5$ and

$r_{\min} = -0.5$ to clip the rotation reward.

2. $r_{\text{pose}} \doteq -\|\mathbf{q} - \mathbf{q}_{\text{init}}\|_2^2$ is the hand pose penalty. $\lambda_{\text{pose}} = -0.3$.
3. $r_{\text{torque}} \doteq -\|\boldsymbol{\tau}\|_2^2$ is the torque penalty. $\lambda_{\text{torque}} = -0.1$.
4. $r_{\text{work}} \doteq -\boldsymbol{\tau}^\top \dot{\mathbf{q}}$ is the energy consumption penalty. $\lambda_{\text{work}} = -2.0$.
5. $r_{\text{linvel}} \doteq -\|\mathbf{v}\|_2^2$ is a object linear velocity penalty. $\lambda_{\text{linvel}} = -0.3$.

Our final reward function is then defined as

$$r = r_{\text{rot}} + \lambda_{\text{pose}} r_{\text{pose}} + \lambda_{\text{linvel}} r_{\text{linvel}} + \lambda_{\text{work}} r_{\text{work}} + \lambda_{\text{torque}} r_{\text{torque}}.$$

Discussion on Reward Choice. Our reward function contains a few different components. We clip the rotation reward because otherwise the policy will learn to rotate as fast as possible ignoring the stability requirement. Without the pose penalty, the policy performs worse because the learned gait is unnatural and it does not learn to break and establish new contact. Please see an example in this link. The energy penalty and linear velocity penalty greatly decrease the commanded torque and object’s linear velocity. This helps encourage the policy to learn a more stable and smooth gait to solve the task, and empirically yields better sim-to-real performance. To speed up training, we terminate the episode when the object falls out of the fingertips (about 14.5 cm above the palm). Note that this is different when we do evaluation where we reset the episode when the object falls out of fingers (about 10.0 cm) which is more similar to the cases in the real-world.

Network Architecture. The base policy π is a multi-layer perceptron (MLP) which takes in the state $\mathbf{o}_t \in \mathbb{R}^{96}$ and the extrinsics vector $\mathbf{z}_t \in \mathbb{R}^8$, and outputs a 16-dimensional action vector \mathbf{a}_t . There are four layers with hidden unit dimension [512, 256, 128, 16]. We use ELU [29] as the activation function. The extrinsics encoder μ is also a three layer MLP with hidden unit dimension [256, 128, 8] and encodes $\mathbf{e}_t \in \mathbb{R}^9$ and output $\mathbf{z}_t \in \mathbb{R}^8$. We use ReLU as the activation function.

The input to the adaptation module is the robot joint position and actions in past 30 timesteps (corresponding to 1.5 seconds). This module first encode the joint position and action pairs into a 32-dimensional representations for each timestep via a two-layer MLP (with hidden unit dimension [32, 32]). Then, we apply three 1-D convolutional layers over the time dimension to capture temporal correlations in the input. The input channel number, output channel number, kernel size, and stride of each layer are [32, 32, 9, 2], [32, 32, 5, 1], [32, 32, 5, 1]. The flattened CNN output is linearly projected by a linear layer to estimate $\hat{\mathbf{z}}_t$. We use Adam optimizer [87] to minimize MSE loss. The learning rate is $3\text{e}-4$.

Optimization Details. During base policy training, We jointly optimize the policy π and the object property encoder network μ using PPO [158]. In each PPO iteration, we collect samples from 16,384 environments with 8 agent steps each (corresponding to 0.4 seconds). We train 5 epochs with a batch size 32,768, leading to 20 gradient updates on this dataset. The learning rate is $5\text{e}-3$. We optimize for 100,000 gradient updates. It takes about 500 million agent steps which corresponds to roughly 7,000 hours real-world time. The advantage of our approach is that we can utilize the huge amount of simulation samples to learn a complex behavior, which will be infeasible in the real-world.

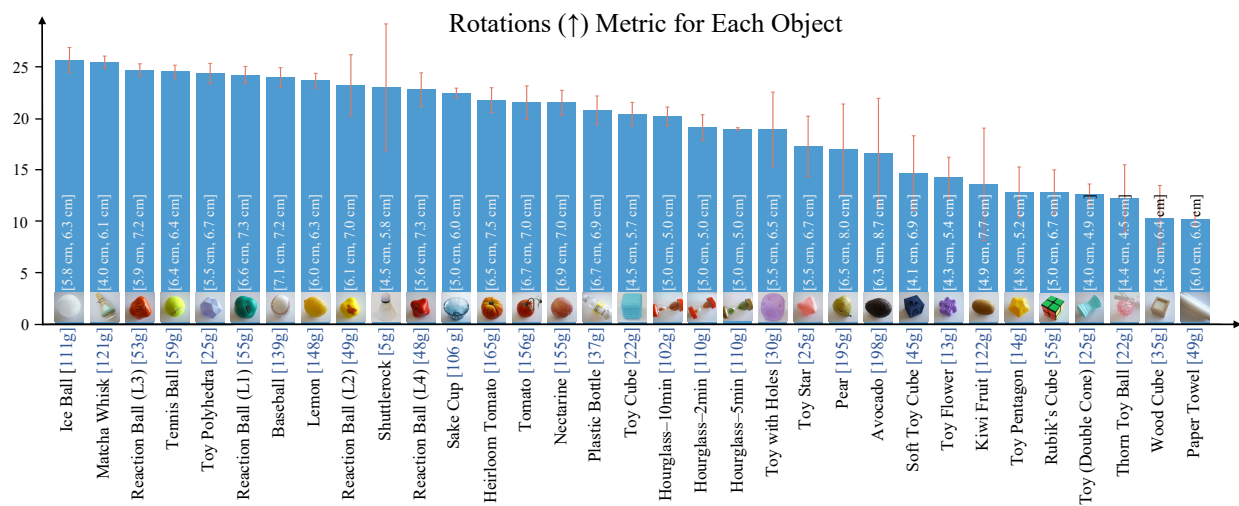


Figure D.2: The performance for each object in terms of the number of rotations achieved. Each object is annotated with the mass and the shorted and longest diameter of the cross section with the fingertips. We find that sphere and light objects are easier to rotate compared to irregular objects such as avocado, cube, pentagon, and toy flower. The paper towel is with the lowest number of rotation because it keeps colliding with the fingers due to its size (see the video for details).

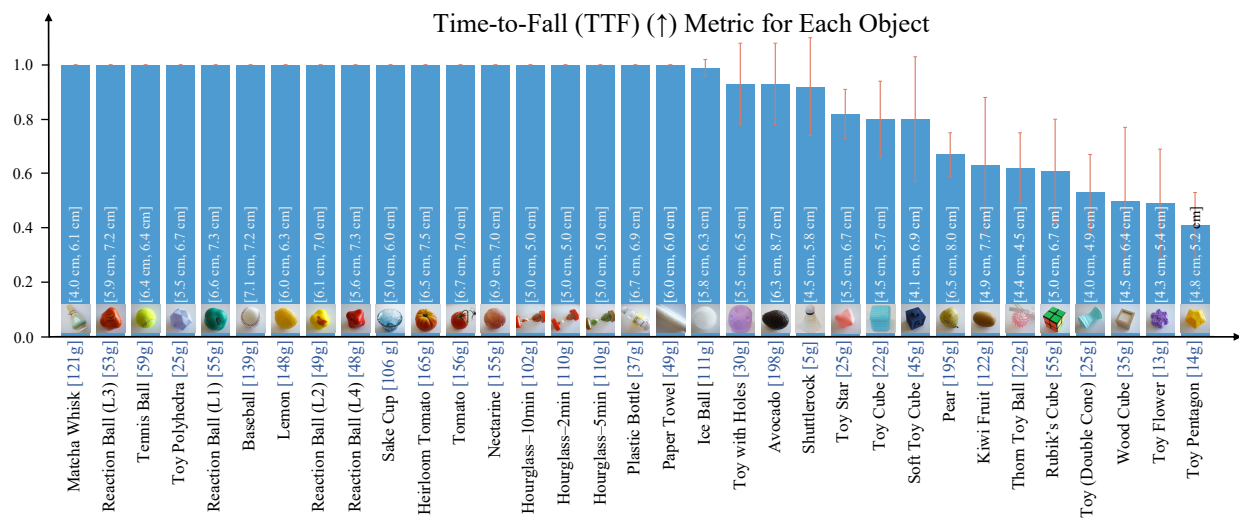


Figure D.3: The stability for each objects in terms of the normalized time-to-fall metric. Each object is annotated with the mass and the shorted and longest diameter of the cross section with the fingertips. Our policy can keep more than half of the objects stable for more than 30 seconds and rotate more than 15 seconds for all the objects. We also find it's easier to maintain the stability for spherical objects than the cube or pentagon objects.

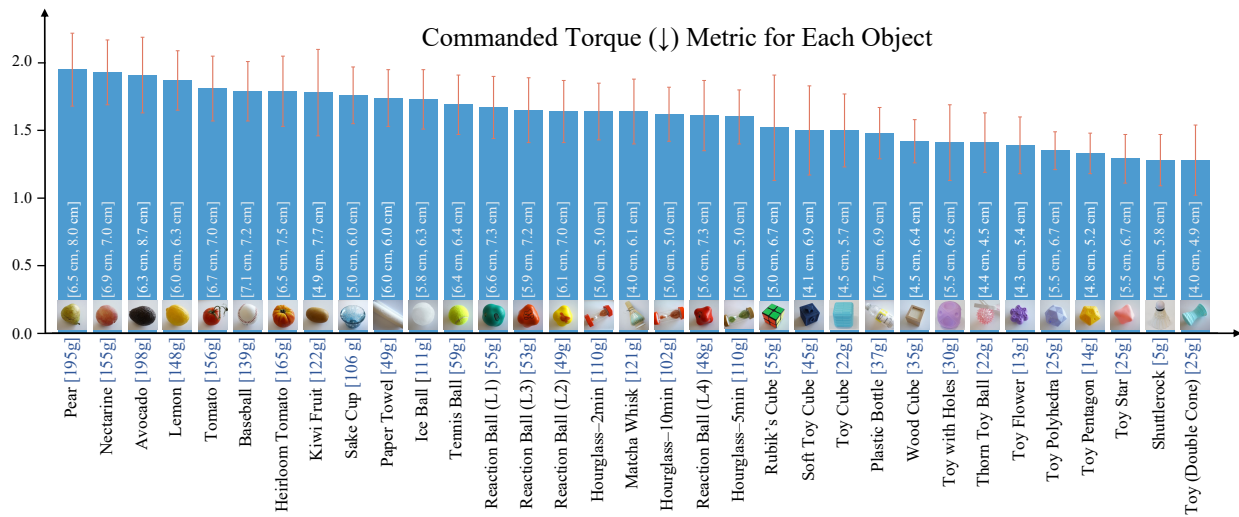


Figure D.4: **The commanded torques during rotating different objects.** Each object is annotated with the mass and the shortest and longest diameter of the cross section with the fingertips. We find a general trend that the commanded torques correlate with the mass of the object, which suggests our policy can adapt and command different torques according to the estimated object mass.