Agile Legged Robots through Reinforcement Learning and Optimal Control

Yuxiang Yang

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2024

Reading Committee:

Byron Boots, Chair

Dieter Fox

Abhishek Gupta

Jie Tan

Program Authorized to Offer Degree:

Computer Science & Engineering

University of Washington

Abstract

Agile Legged Robots through Reinforcement Learning and Optimal Control

Yuxiang Yang

Chair of the Supervisory Committee:
Byron Boots
Computer Science & Engineering

This thesis addresses the challenge of developing agile legged robot controllers capable of high-speed, precise, and rapidly adaptive behaviors in real-world scenarios. While recent advancements have demonstrated impressive hardware capabilities of legged robots in controlled environments, deploying these controllers in complex, real-world environments remains a significant challenge. Traditional optimal control methods, which rely on predefined physics models to optimize motor commands, can precisely track desired motions but cannot plan for complex, long-horizon trajectories due to computational constraints. On the other hand, reinforcement learning frameworks offer the potential to learn versatile, perception-integrated motion policies end-to-end. However, they often lack the precision and robustness of optimal control methods and require extensive tuning in reward shaping and sim-to-real, for effective real-world application. In this thesis, we propose a hierarchical framework that merges the versatility of reinforcement learning with the precision of optimal control for enhanced agility of legged robots. We explore key challenges in training and deploying this framework and suggest methods to extend it to novel environments and tasks.

We introduce the proposed hierarchical learning-control framework in three stages. First, we develop an early version of this framework for learning energy-efficient gait transitions in high-speed locomotion, with a high-level gait policy and a low-level centroidal controller. We then expand the interface between the high-level policy and the low-level controller

for advanced control of continuous jumping motions, and restructure the low-level optimal control problem for GPU-accelerated training. Lastly, to achieve real-world terrain-aware jumping, we integrate perception into the framework and redesign critical components for robust real-world performance. With this final version of our framework, we achieve high-speed, animal-like jumping on challenging terrains such as stairs and stepping stones.

Next, we extend the proposed hierarchical learning framework to novel terrains and tasks. By incorporating semantic information into the perception pipeline, we enable the legged robot to navigate quickly and safely through complex offroad terrains such as rocks, mud, or vegetation. Furthermore, we increased the capabilities of quadrupedal robots from basic locomotion to versatile loco-manipulation, with a novel lightweight gripper design and a restructured hierarchical framework optimized for teleoperation.

This thesis presents general-purpose algorithmic frameworks for perception-integrated, highly dynamic motion control of legged robots, as well as open-source implementations that can be readily deployed or further developed by the broader robotics community. To demonstrate their robustness and applicability, the methods proposed have been rigorously tested in standard real-robot benchmark tests and in diverse, complex real-world scenarios.

TABLE OF CONTENTS

ACKNOWLEDGMENTS

This dissertation would not have been possible without the enormous and unwavering support I received from my partner, mentors, friends, and collaborators throughout the PhD journey.

I would like to start by thanking my advisor, prof. Byron Boots. Your insightful and honest feedback is invaluable in my growth as a researcher. Thank you for encouraging me to find my true research passion, and supporting me in my pursuit of challenging research problems. In addition, your generosity in robot and equipment purchase has fundamentally expedicted my research progress. You are not only an academic advisor in my research journey, but also an important mentor in my life.

I would also like to thank my mentors at Google Deepmind for their continual support in my PhD journey. A special thanks goes to Jie Tan, who has mentored me since I finished my undergraduate studies in 2018. You not only brought me into the world of research, but also provided me with invaluable advice on how to think, work, and communicate like a researcher. It's a great honor to get to know you and work with you in all these years. Thank you Tingnan, Wenhao, Ken, and Atil, for bringing me into the world of legged robots, and for all the industry-academia discussions over the years.

I would like to thank prof. Dieter Fox, prof. Abhishek Gupta, and prof. Karen Leung, for serving in my thesis committee and providing their valuable feedback on my work. In addition, I would extend my gratitude to prof. Guanya Shi, for all your effort in not only mentoring me, but also working with me hands-on in a lot of my research projects during your postdoc at UW.

My PhD would not have been possible without the invaluable support from my friends

and collaborators. I would like to thank Rosario Scalise and Mateo Guaman Castro for all the help and support in my robot experiments, and thank Xiangyun Meng for all the insightful discussions about robot perception. I would also like to thank Changyi Lin, Yaru Niu and prof. Ding Zhao at Carnegie Mellon University for their generous support in lending me a backup robot and helping me out with hardware designs, and He Li at Notre Dame (now at BDAII) for all the insightful discussions about optimal control frameworks. I want to thank my friends at the UW Robot Learning Lab - Mohak, Anqi, Jake, Boling, Nolan, Sasha, Sandesh, Carolina, Tyler, Kevin, Sanghun for the friendship and camaraderie. I'd like to extend a special thanks to my long-term friend - Yifeng Zhu at UT Austin, for all your emotional support throughout my PhD years, and for being the witness for my wedding.

Last but definitely not least, a very special thanks goes to Xiaoyi Zhang, my wife, soulmate and life partner. It's so fortunate that we could start our PhD at the same time, and walk through this journey together. Thanks to your deep compassion and thoughtfulness, my PhD journey was marked with not only the development in research skills, but also invaluable amount of *personal* growth. You encouraged me to discover myself, follow my heart, and keep my values even under the most difficult times. You are the eternal light in my journey through life.

In addition to all the human friends, my very last thanks goes to a fluffy boy called Summer, our Siberian Husky dog. I would not have started the entire stair-jumping idea, if we had not accidentally stepped into the shelter and brought you home. Whenever I get some exciting new results on the robot, your swift and effortless gaits always remind me about the huge gap between robots and animals. You have supported me mentally and physically, and improved my work-life-balance significantly over the years.

DEDICATION

To my wife and partner, Xiaoyi.

Chapter 1

INTRODUCTION

Legged robots offer the promise of traversing some of Earth's most challenging terrains, from steep mountain roads to muddy, deformable landscapes. The ability to navigate these terrains at high speed can bring immense benefits, from rapid search-and-rescue operations to remote explorations in dangerous areas. In such scenarios, an agile legged robot must move quickly and precisely, and adapt rapidly to fast-changing environments. Achieving this level of agility necessitates a comprehensive integration of efforts across many fields in robotics, including mechanical design [1, 2], control systems [3, 4], sensor integration [5, 6], and machine learning techniques [7, 8, 9]. While significant advancements have been made in many of these areas, today's legged robots still cannot achieve the same level of agility as their biological counterparts.

Historically, researchers have achieved significant success in developing agile legged locomotion using optimal control (OC) frameworks. By constructing a dynamic model of the robot and optimizing control inputs to follow desired trajectories, these controllers enable reactive and precise tracking of highly dynamic movements, even under severe external perturbations [10, 3, 4, 11]. However, these frameworks encounter several bottlenecks when scaling to real-world complexity. Firstly, the high computation demand that these systems can only plan a few steps ahead, which isn't always sufficient for challenging tasks with long periods of under-actuation [12, 13, 14, 15]. Moreover, these frameworks often struggle in complex terrains with limited choices for foot placement, not only due to the challenges of perception integration [6] but also because of the increased complexity in solving the optimal control problem [12]. Therefore, it can be challenging to use standard optimal-control

methods for agile robot locomotion in challenging real-world environments.

Learning-based frameworks offer a promising alternative to traditional optimal control for agile legged locomotion. These frameworks develop policies that directly map from sensor inputs to motor outputs, and train these policies through extensive environment interactions. [7, 8, 16, 9, 17, 18]. Although these frameworks can learn versatile behaviors and seamlessly integrate perception in an end-to-end manner, they face notable challenges to learn smooth, real-world deployable policies. Without explicit knowledge about the robot structure or physics properties, these frameworks can often get stuck in suboptimal or even infeasible motions, and require additional effort in reward shaping [8], curriculum design [19], or action-space parameterization [20] to learn smooth and deployable trajectories. Another bottleneck for these approaches is the 'sim-to-real' gap [7], where policies trained extensively in simulation overfit to simulation settings and fail to perform as-well in real-world settings. This gap is particularly evident for agile tasks, which push the hardware to its limits and are highly sensitive to even small sim-to-real discrepancies.

An effective framework for agile legged locomotion, capable of fast, precise, and adaptive movements, must combine the strengths of both structured and learning-based approaches. It should incorporate prior physical knowledge to ensure reliable and accurate motion tracking, while also leveraging learning algorithms for versatile, perception-integrated behaviors. This requirement motivates the core research statement of this thesis:

A hierarchical combination of reinforcement learning and optimal control is crucial for enabling agile legged robots with fast, precise, and adaptive motions.

Guided by key research questions arising from this central principal, this thesis develops a hierarchical framework that combines reinforcement learning with optimal control. The framework enables a wide range of agile behaviors on quadrupedal robots, such as high-speed running with energy-efficient gait transition and continuous and terrain-adaptive jumping. We address key challenges in designing such a framework, including environment configuration, training acceleration and perception integration. Additionally, we extend the concept of agility by broadening the testing environments from traditional urban or laboratory set-

tings to rugged off-road terrains, and by broadening the scope of tasks from locomotion to versatile loco-manipulation.

Next, we provide an overview of works presented in this thesis:

A Hierarchical Framework for Agile Legged Locomotion

We start by establishing the main hierarchical learning-control framework introduced in this thesis. We first show how a preliminary version of this framework can learn to specify gaits to a low-level optimal controller. We then expand the scope of this framework for continuous, long-distance jumping, and redesigned the low-level controller for GPU-accelerated training. Lastly, we incorporate perception into this framework to learn high-speed, animal-like jumping on challenging terrains like stairs or stepping stones.

Chapter 3 introduces the initial version of our hierarchical learning-control framework. Motivated by the challenge in planning through contact [21], we introduce a high-level gait policy to learn contact references to a low-level locomotion controller based on model predictive control (MPC), and train this gait policy end-to-end with the low-level controller using reinforcement learning. The resulting policy learns a wide variety of gaits for different locomotion speeds, from low-speed crawling to high-speed fly-trotting, and can seamlessly switch between the gaits based on velocity command.

Chapter 4 advances this hierarchical learning-control framework to the task of continuous, omni-directional jumping. For these motions, the high-level policy not only need to specify an adequate contact timing, but also need to carefully plan a body trajectory for precise take-off and landing. For this purpose, we augment a heuristically designed body pose planner with a learned residual policy. Working together, the robot can learn stable, continuous jumping in multiple directions as well as jump-turns.

Chapter 5 redesigns this hierarchical learning-control framework for efficient training of general locomotion gaits. In this work, we refactored the high-level policy to output the gait timing, reference body motion, and reference foot trajectory simultaneously, so that the policy can coordinate a wide variety of locomotion patterns, from standard walking to long-distance bounding. In addition, we reformulated the low-level optimal control problem

so that it can be solved in parallel on GPU. This reformulation speeds up the training by an order of magnitude, and enables the policy to learn most locomotion gaits in less than 20 minutes on a single GPU.

Building on top of this general framework, *chapter* 6 adds perceptual input into the high-level policy and achieves continuous, high-speed, animal-like jumping on discontinuous terrains like stairs or stepping stones. To increase the robustness of the low-level controller, we further integrated a velocity-based feedback controller together with the optimization-based feedforward controller, so that the low-level controller can prepare for future trajectories while actively counter against dynamics shifts. To encourage policy exploration in feasible regions, we include the cost of the low-level optimal controller as part of the reward function, so that the policy is aware of the underactuated nature of the robot. Lastly, the perceptual information is introduced into the policy via a heightmap predictor, which runs a recurrent neural network to estimate terrain heights from consecutive egocentric depth images. With this framework, a quadrupedal robot can jump up stairs in animal-like high-speed bounding gaits, and crosses two steps in each jumping cycle. In addition, we achieve state-of-the-art performance in jumping over single discontinuities like gaps or steps.

Extending the Scope of Agility for Legged Robots

While the proposed hierarchical learning-control framework achieved unprecedented performance in agile locomotion, the works mentioned so far primarily focused on indoor environments and standard locomotion tasks. In the second part of the thesis, we aim to extend the scope of agility by demonstrating the effectiveness of this framework in new environments and new tasks.

In *chapter* 7, we look into extending the performance of agile legged locomotion in complex off-road terrains. While most prior works focused on the *geometry* of the environment, such as obstacle location and terrain shapes, we find that the terrain *semantics*, such as deformability and contact properties, provide valuable information for off-road locomotion. As it is challenging to accurately recreate these semantics information in simulation, we learn a semantics-conditioned high-level locomotion policy directly in the real world from

human demonstrations. With a pre-trained perception embedding, we train a generalizable, terrain-aware locomotion controller using less than 60 minutes of demonstration data, and find that the resulting policy can perform fast and safe locomotion in a variety of offroad terrains.

Finally, in *chapter* 8, we extend our scope from standard locomotion to loco-manipulation, where the robot not only moves but also *interacts* with the environment. Unlike prior works that use an additional top-mounted robot arm for loco-manipulation, we mount two light-weight, custom-designed grippers on the front legs of the robot, and use the leg's build-in motors in addition to the gripper motors for versatile manipulation. The resulting hardware supports multiple manipulation modes, such as single-arm grasping or bi-arm collaboration. We design a similar hierarchical framework to control this framework, where the high-level framework collects motion references from human teleoperation and the low-level controller tracks the motion reference.

Chapter 2

A BRIEF OVERVIEW OF LEGGED ROBOTS

In this chapter, we provide a brief overview of the history of legged robots, with a particular focus on hardware and controller design. We then focus on quadrupedal robots for the rest of the thesis. We review standard control frameworks for quadrupedal robots, including optimal control based controllers and reinforcement learning based controllers. The hierarchical framework proposed in this thesis builds on top of these frameworks for agile and adaptive locomotion.

2.1 Legged Robot Hardware

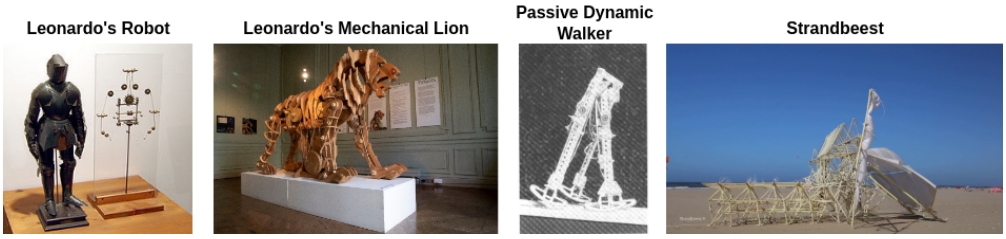


Figure 2.1: Examples of passive walking mechanisms. From left to right: the mechanical knight [22] and mechanical lion [23] conceptualized by Leonardo da Vinci (1495), the Passive Dynamic Walker (1990) [24], and Strandbeest (2020) [24].

Researchers have had a long history developing legged mechanisms that mimic the walking behaviors of humans and animals (Fig. 2.1). Even without active sensing or actuation, pioneers like Leonardo da Vinci conceptualized mechanical systems that could emulate human locomotion, as illustrated by his designs for passive automata ([23, 22]). Recent efforts have brought such passive walking mechanisms to life. Notable examples include Tad McGeer's

passive dynamic walker [24], which can autonomously walk in downward slopes without any external power, and the Strandbeest robots [25] that uses wind power for walking. These early examples demonstrated the potential of legged robots and paved ways for later development of active legged robots.

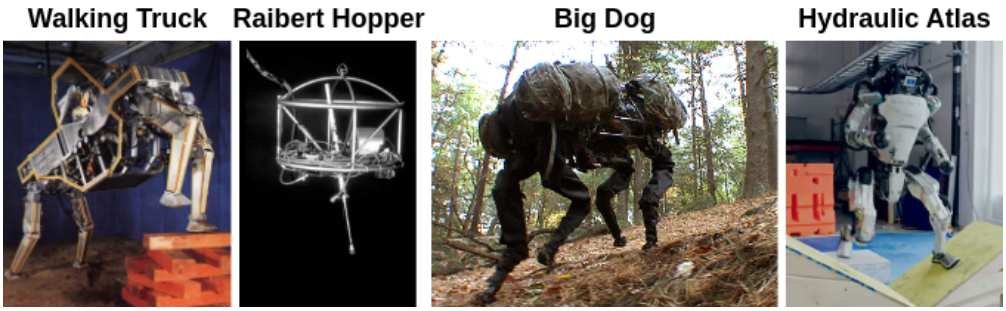


Figure 2.2: Examples of early legged robot designs that use hydraulic actuators. From left to right: the GE Walking Truck (1969) [26], the Raibert Hopper (1983) [27], the Big Dog (2005) [28], and the Hydraulic Atlas (2016) [29]

.

Due to the low power density in early designs of electrical motors, initial developments of active legged robots typically rely on *hydraulic* motors for high performance locomotion (Fig. 2.2). With carefully designed controllers, these hydraulically actuated robots can perform highly dynamic motions such as single leg jumping [27], running [28], and backflipping [29]. However, the bulky size and weight of these actuators, as well as the high maintenance cost, makes it difficult to deploy in real-life scenarios.

Recent development in electric motors and batteries have paved ways for small-size, lightweight legged robots that can be easily deployed in the real world (Fig. 2.3). Built with a compact design and lightweight materials, these robots can perform complex tasks like goal-kicking [30], dancing [31], running [32] and hiking [2] for hours within one battery charge. The adoption of electric powers have also unified the mechanical design of legged robots. For example, many recent quadrupedal robots [31, 32] feature a similar 12-motor design with

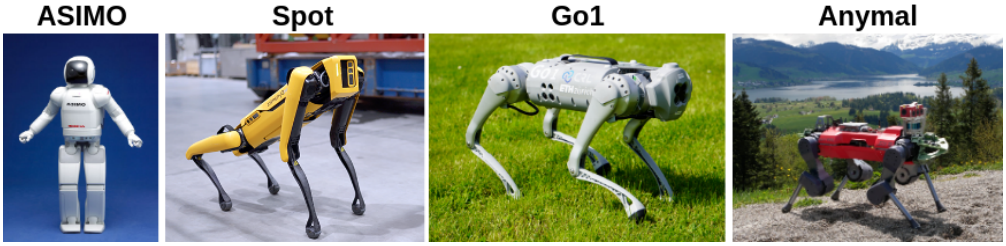


Figure 2.3: Examples of recent legged robots with batteries and electrical actuators. From Left to Right: the Honda ASIMO Robot [30], the Spot [31], Unitree Go1 [32], and the Anymal Robot [2].

backward knees, and recent humanoid robots [33, 34, 35] feature similar design of thigh and hip motors. This unified hardware design also paved ways for unified control frameworks for different legged robots [36, 37], where researchers design universal, hardware-agnostic controllers for different legged robots.

2.2 Model-based Optimal Control for Legged Locomotion

In this section, we review popular optimal-control based frameworks for legged robots, with a specific focus on those frameworks that enable high-speed, dynamic behaviors. We will discuss the dynamics modeling of quadrupedal robots, as well as techniques for high-speed real-time optimization of controller commands under these dynamics models.

2.2.1 Dynamics Model for Legged Robots

We describe two most commonly used dynamics models for legged robots, the *centroidal dynamics model* and *whole body dynamics model*. While more simplified models, such as the pointmass model or the spring-loaded inverted pendulum (SLIP) model are popular in earlier designs of optimal controllers, recent optimal control frameworks with improved computational power tend to adopt these two models for higher accuracy.

At a high level, quadrupedal robots use their foot to manipulate their body through foot contacts. By pushing the ground, each contact foot generates a *ground reaction force* GRF,

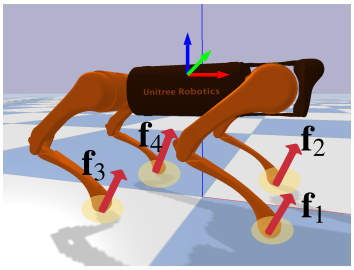


Figure 2.4: Diagram of a quadrupedal robot.

which moves the robot body in desired directions.

Notation We use standard letters (e.g. x) to denote scalars, bold letters (e.g. $\boldsymbol{x}$) to denote vectors, and capitalized bold letters $\mathbf{M}$ to denote matrices. We use $\mathbf{I}_n$ to denote the $n \times n$ identity matrix. $[\cdot]_\times$ convertsa a 3D vector into a 3×3 skew-symmetric matrix for vector cross products, such that $\boldsymbol{a} \times \boldsymbol{b} = [\boldsymbol{a}]_\times \boldsymbol{b}$.

We use the following notation to describe the state of a quadrupedal robot with 12 joints (Fig. 2.4). We denote the robot state as $\boldsymbol{q} = [\boldsymbol{q}_f, \boldsymbol{q}_j] \in \mathbb{R}^{18}$, where $\boldsymbol{q}_f = [p_x, p_y, p_z, \phi, \theta, \psi] \in \mathbb{R}^6$ is the position and orientation (euler angles) of the floating base in world frame, and $\boldsymbol{q}_j \in \mathbb{R}^{12}$ is the position of all robot joints. The base velocity is represented as $\dot{\boldsymbol{q}}_f = [v_x, v_y, v_z, \omega_x, \omega_y, \omega_z]$, which includes the linear and angular velocities of the floating base. Each contact leg $i \in \{1, 2, 3, 4\}$ can generate a ground reaction force $\boldsymbol{f}_i \in \mathbb{R}^3$. We concatenate them together and represent as $\boldsymbol{f}_r = [\boldsymbol{f}_1, \boldsymbol{f}_2, \boldsymbol{f}_3, \boldsymbol{f}_4] \in \mathbb{R}^{12}$.

Whole-body Dynamics The whole body dynamics equation [10] of a quadrupedal robot can be written as:

$$\mathbf{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \mathbf{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{g}(\boldsymbol{q}) = \begin{pmatrix} \mathbf{0} \\ \boldsymbol{\tau} \end{pmatrix} + \mathbf{J}^\top \boldsymbol{f}_r \tag{2.1}$$

Here $\mathbf{M}(\boldsymbol{q}) \in \mathbb{R}^{18\times18}$ represents the generalized mass matrix, $\mathbf{C}(\boldsymbol{q}, \dot{\boldsymbol{q}}) \in \mathbb{R}^{18\times18}$ represents centrifugal and Coriolis force, and $\boldsymbol{g}(\boldsymbol{q}) \in \mathbb{R}^{18}$ represents gravitational forces. $\boldsymbol{\tau} \in \mathbb{R}^{12}$ represents the applied joint torques, and $\mathbf{0}$ is a placeholder vector representing zero external

force directly applied to the robot base. $\mathbf{J} \in \mathbb{R}^{18\times12}$ is the foot jacobian, and $\boldsymbol{f}_r \in \mathbb{R}^{12}$ is the ground reaction forces. This dynamics equation can be derived using standard rigid body dynamics, and the terms can be computed automatically by a rigid body simulator. We refer the readers to the book by Featherstone [38] for a detailed tutorial. The parenthesis in $\mathbf{M}$, $\mathbf{C}$, $\boldsymbol{g}$ highlights that these terms are time-varying and depend on the position and velocity of the robot, which is one of the main challenges in accurate modeling the dyanmics of quadrupedal robots. For simplicity, we omit these parentheses for the rest of the thesis.

We can decompose the terms into blocks, $\mathbf{M} = \begin{pmatrix} \mathbf{M}_{ff} & \mathbf{M}_{fj} \\ \mathbf{M}_{jf} & \mathbf{M}_{jj} \end{pmatrix}$, $\mathbf{C} = \begin{pmatrix} \mathbf{C}_{ff} & \mathbf{C}_{fj} \\ \mathbf{C}_{jf} & \mathbf{C}_{jj} \end{pmatrix}$, $\boldsymbol{g} = \begin{pmatrix} \boldsymbol{g}_f \\ \boldsymbol{g}_j \end{pmatrix}$, $\mathbf{J} = \begin{pmatrix} \mathbf{J}_f & \mathbf{J}_r \end{pmatrix}$, where the subscript $(\cdot)_f$ picks the first 6 dimensions to represent the body, and $(\cdot)_j$ picks the last 12 dimensions to represent the legs. The cross terms $\mathbf{M}_{fj}, \mathbf{M}_{jf}, \mathbf{C}_{fj}, \mathbf{C}_{jf}$ represents the interaction between the robot and its legs, such as the ability to adjust the body pose through leg movements during free-falling without foot contacts.

To better understand this dynamics equation, let us omit these off-diagonal blocks, and approximately decompose the whole body dynamics equation (Eq. 2.1) into the body and joint degree of freedoms:

$$\mathbf{M}_f \ddot{\boldsymbol{q}}_f + \mathbf{C}_f \dot{\boldsymbol{q}}_f + \boldsymbol{g}_f = \mathbf{J}_f^\top \boldsymbol{f}_r \tag{2.2}$$

$$\mathbf{M}_j \ddot{\boldsymbol{q}}_j + \mathbf{C}_j \dot{\boldsymbol{q}}_j + \boldsymbol{g}_j = \boldsymbol{\tau} + \mathbf{J}_j^\top \boldsymbol{f}_r \tag{2.3}$$

Intuitively, Eq.2.2 states the relationship between the ground reaction forces $\boldsymbol{f}_r$ and the base acceleration $\ddot{\boldsymbol{q}}_f$, and Eq. 2.3 states how the ground reaction forces $\boldsymbol{f}_r$ can be controlled from joint torques. Based on this intuition, we can further derive the centroidal dynamics model.

Centroidal dynamics With the adoption of electrical actuators, most modern quadrupedal robots center most of its components (battery, motor, etc.) around its body, and use lightweight materials for legs. For example, the torso and upper thigh joints of the Unitree Go1 [32], Unitree A1 [39], Mini Cheetah [3], and the Spot [31] all take up over 70% of

their total weight. Because of that, we can approximate the robot as a single rigid body with massless legs. Under these assumptions, we can rewrite the base (Eq. 2.2) and joint dynamics (Eq. 2.3) as:

$$\mathbf{M}_f \ddot{\boldsymbol{q}}_f + \boldsymbol{g}_f = \mathbf{J}_f^\top \boldsymbol{f}_r \tag{2.4}$$

$$\mathbf{0} = \boldsymbol{\tau} + \mathbf{J}_j^\top \boldsymbol{f}_r \tag{2.5}$$

where Eq. 2.4 represents the dynamics of ground reaction forces, and Eq. 2.5 represents the relationship between joint torque and end effector force in quasi-static situations.

We can now write Eq. 2.4 in a more compact form:

$$\ddot{\boldsymbol{q}}_f = \mathbf{A}\boldsymbol{f}_r - \boldsymbol{g}_f \tag{2.6}$$

where

$$\mathbf{A} = \mathbf{M}_f^{-1}\mathbf{J}_f^\top = \begin{bmatrix} \mathbf{I}_3/m & \mathbf{I}_3/m & \mathbf{I}_3/m & \mathbf{I}_3/m \\ \mathbf{I}_{\text{base}}^{-1}[\boldsymbol{r}_1]_\times & \mathbf{I}_{\text{base}}^{-1}[\boldsymbol{r}_2]_\times & \mathbf{I}_{\text{base}}^{-1}[\boldsymbol{r}_3]_\times & \mathbf{I}_{\text{base}}^{-1}[\boldsymbol{r}_4]_\times \end{bmatrix} \tag{2.7}$$

is the generalized inverse inertia matrix. The $\boldsymbol{r}_i$s denote the position of the ith foot in the body frame of the robot.

2.2.2 Optimal Control Frameworks

At a high level, optimal control frameworks assume a pre-defined foot contact schedule and use different control strategies for swing and stance legs. For swing legs, these frameworks generate a reference swing foot trajectory by interpolating key waypoints, and tracks this reference trajectory using joint-space or task-space PD control. For stance legs, these frameworks solve an optimization problem to find the optimal ground reaction forces $\boldsymbol{f}_r$ to track a reference body trajectory $\boldsymbol{q}_f^{\text{ref}}$. Note that both the whole-body dynamics (Eq. 2.1) and the centroidal dynamics (Eq. 2.6) are *linear* in terms of control inputs. Therefore, most of these frameworks formulate the stance foot control as a quadratic program (QP) for efficient solving.

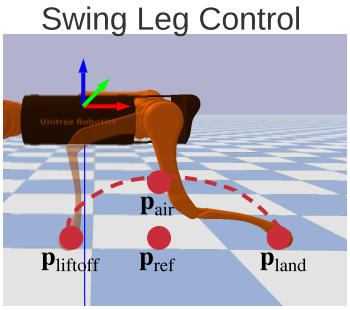


Figure 2.5: Example of a swing foot trajectory that interpolates between the liftoff, mid-air and landing positions.

Swing Leg Control To find the reference swing foot trajectory, a controller usually interpolates between a few key predefined key frames. A common choice of key frames include the lift-off position $\boldsymbol{p}_{\text{lift-off}}$, the mid-air position $\boldsymbol{p}_{\text{air}}$, and the landing position $\boldsymbol{p}_{\text{land}}$. The lift-off position is the recorded foot position when the leg first switches into the swing mode. The mid-air position is a fixed distance above the leg's neutral standing position $\boldsymbol{p}_{\text{ref}}$. The landing position is determined by the Raibert Heuristic [27]:

$$\boldsymbol{p}_{\text{land}} = \boldsymbol{p}_{\text{ref}} + \boldsymbol{v}_{\text{CoM}} T_{\text{stance}}/2 \tag{2.8}$$

which ensures equal forward and backward swings of the leg in the next contact phase.

One-step Optimal Control Given a reference body acceleration $\ddot{\boldsymbol{q}}_f^{\text{ref}}$, we can set up the following quadratic program (QP) to solve for motor control commands:

$$\min_{\boldsymbol{f}_r} \ \|\ddot{\boldsymbol{q}}_f - \ddot{\boldsymbol{q}}_f^{\text{ref}}\|_{\mathbf{U}} + \|\boldsymbol{f}_r\|_{\mathbf{V}} \tag{2.9}$$

$$\text{subject to: } (\mathbf{M}\ddot{\boldsymbol{q}}_f + \mathbf{C}\dot{\boldsymbol{q}}_f + \boldsymbol{g})_{[:6]} = \mathbf{J}_f^\top \boldsymbol{f}_r \quad \text{or} \quad \ddot{\boldsymbol{q}}_f = \mathbf{A}\boldsymbol{f}_r + \boldsymbol{g}_f \tag{2.10}$$

$$f_{i,z} = 0 \qquad \text{if } i \text{ is a swing leg} \tag{2.11}$$

$$f_{\min} \le f_{i,z} \le f_{\max} \qquad \text{if } i \text{ is a stance leg} \tag{2.12}$$

$$-\mu f_{i,z} \le f_{i,x} \le \mu f_{i,z}, \quad -\mu f_{i,z} \le f_{i,y} \le \mu f_{i,z} \qquad i = 1, \dots, 4 \tag{2.13}$$

$\mathbf{U}$ and $\mathbf{V}$ are diagonal weight matrics. Eq. 2.10 represents the whole-body dynamics or the centroidal dynamics, Eq. 2.11 specifies contact schedule, Eq. 2.12 represents the actuator limit in normal forces, and Eq. 2.13 represents approximated friction cone. Given the relative small size of the problem, this QP can typically be solved in less than 1ms on a modern CPU (e.g. Intel I9-9900K or Apple M1 Pro).

Multi-step MPC While the one-step optimal controller provides an efficient way for foot force optimization, it requires an accurate reference acceleration $\ddot{\boldsymbol{q}}_f^{\text{ref}}$, which can be difficult to compute for complex tasks. For example, a galloping gait might require the robot to slow down slightly during landing, and accelerate quickly when its about to take-off. Such intricate motions can be difficult to capture using a constant reference or a standard PD controller. One way to solve this problem is to extend the one-step optimal controller into a receding horizon model predictive control (MPC) problem, and solve it using the same framework [4, 3]. Given the high efficiency of modern QP solvers, it is typically possible to solve a 5 to 10-step QP in less than 5ms, which is feasible for real-time torque control. With an appropriately selected timestep and plan horizon, an MPC framework can plan for more sophiscated trajectories with small periods of under-actuation, such as fly-trotting [40] or bounding [3].

Combination of MPC and WBC Despite the long planning horizon of MPC, note that MPC-based controllers inherently provides lower-quality control solutions compared to the one-step controller. Firstly, to speed up the optimization problem, these controllers typically use the simplified centroidal model and a large timestep, which leads to inaccurate dynamics estimation in long horizons. Moreover, because the both dynamics models are time-varying, the MPC controller have to rely on *estimated* foot positions to set up dynamics equations beyond the first step. Even with rapid replanning, these inaccuracy can lead to poor-quality solutions. To combine the long planning horizon of MPC with the accuracy of WBC, one can warm-start the WBC solver with the solution of MPC, which has proved effective for a

variety of agile tasks [10].

Additional Kinematics-based Feedback Another common problem for optimal control based controllers is that they only compute a *feedforward* term based on robot dynamics to accurately track *future* trajectories, and do not actively counter against unexpected dynamics shifts or perturbations. For example, without prior knowledge about foot sliding, these controllers can keep pushing a sliding leg backwards, and further accelerate this sliding leg. One common way to fix this problem is to add another kinematics-based feedback controller [10]. Assuming static foot contact, one can compute the reference position (and velocity) for the contact foot from the reference body position (and velocity), and further convert that to reference joint positions (and velocities). This joint-space reference can provide an additional feedback torque on top of the feedforward controller, and stabilize the robot in unexpected situations.

2.2.3 The Computation Bottleneck

Despite the success of OC-based controllers, the computation bottleneck for real-time replanning limits the performance of these frameworks in complex long-horizon behaviors, especially those requiring careful motion coordination. We highlight a few common problems below:

Short Planning Horizon Due to the requirement for real-time torque optimization, optimal control based frameworks typically adopt a short planning horizon, and cannot plan for longer trajectories (e.g. jumping with extended periods of air phase) or more complex terrains (e.g. limited choices for foot placement). For example, even with solver warm-start and mixed-precision models [12], it can be still difficult to achieve a real-time planning horizon of longer than 2 seconds. One way to resolve this limitation is to pre-plan the entire trajectory offline [13, 14, 41]. However, these methods typically assume prior knowledge about the terrain and the task, and require additional effort in trajectory interpolation [41, 5, 42] for real-time behavior adaptations.

Foot Placement in Complex Terrains Due to the same computational bottleneck, most optimal control frameworks use the aforementioned Raibert Heuristics [27] for swing foot placement, and cannot plan for swing foot placements together with stance foot forces. Therefore, these frameworks face significant challenges in terrains with scarce footholds, such as stepping stones or wide gaps, where the solver can face significant challenges solving complex foot placement and body motion optimization problems in real time.

Fixed-Gait Assumption Due to the challenge in joint optimization of discrete contact sequences and continuous contact forces, most optimal control frameworks assume a fixed gait pattern and only outputs the desired foot forces according to that pattern. However, for more dynamic behaviors like jumping, it is important to optimize for foot forces and contact sequences *simultaneously*. Because the complexity of such optimization scales exponentially with the planning horizon, solving such optimization problem in real time is computationally intractable.

2.3 Reinforcement Learning for Legged Locomotion

In this section, we provide a brief overview of the reinforcement learning (RL) framework, popular RL algorithms, and existing results in applying RL to legged locomotion.

2.3.1 Reinforcement Learning Problem

Problem Formulation The RL problem is represented as a Markov Decision Process (MDP), which includes the state space $\mathcal{S}$, action space $\mathcal{A}$, transition probability $p(s_{t+1}|s_t, a_t)$, reward function $r : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$, and initial state distribution $p_0(s_0)$. We aim to learn a policy $\pi : \mathcal{S} \mapsto \mathcal{A}$ that maximizes the expected cumulative reward over an episode of length T, which is defined as $J(\pi) = \mathbb{E}_{s_0 \sim p_0(\cdot), s_{t+1} \sim p(\cdot|s_t, \pi(s_t))} \sum_{t=0}^{T} r(s_t, a_t)$.

In the context of robot control, it is common to design the state space to include all robot proprioceptive information (i.e. $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$), in addition to potential perception inputs. The

action space typically corresponds to motor or foot position commands, which is converted to torque using a PD controller.

2.3.2 Solving the RL Problem

Here we review the two common frameworks to solve RL problems, which are adopted in this thesis. At a high level, RL algorithms operate by exploring different actions in the environment, and gradually building a preference towards higher-reward actions.

Policy Gradient Based Algorithms As a classic reinforcement learning algorithm, *policy gradient* (PG) collects trajectories of the current policy, derives a gradient of the policy parameters to maximize the expected cumulative reward, and improves the policy in the direction of that gradient. To reduce the variance of this gradient estimation, *actor-critic* based methods learn an additional value model, known as the *critic*, that estimates the expected cumulative reward of the current action. This critic can stabilize the policy update and lead to more stable learning processes.

As a specific type of actor-critic algorithm, Proximal Policy Optimization (PPO) [43] introduces several key advancements over standard actor-critic for stable and efficient training. The overall idea is to keep the updated policy close to the old policy, and prevent drastic policy changes that make the learning process unstable. As a very popular algorithm, PPO has been deployed in many RL applications in areas such as robotics, natural language processing, game agent design. One unique benefit of PPO is that its data collection process can be performed *in parallel*. Therefore, its training time can be significantly reduced when using parallelized simulators.

Evolutionary Strategies As an alternative to policy-gradient algorithms, evolutionary strategies (ES) do not rely on the MDP structure of the RL problem, and optimizes policies end-to-end directly through trial-and-errors. More specificaly, ES-based algorithms start with an initial policy in the environment, slightly perturb the policy parameters, and col-

lects the total reward achieved by these perturbed policies. It then estimates a direction of improvement through the collected reward signals, and improves the policy in this direction. As a notable example, Augmented Random Search (ARS) [44] introduced a few improvements to stabilize this process, and proves effective in a number of reinforcement learning benchmarks.

Since ES-based methods do not estimate a value function or assume the underlying MDP structure, they can be particularly effective in non-Markovian environments with partial observability or action delays. However, these methods are typically more effective for smaller networks with less than 10,000 parameters, and may not work effectively on large recurrent or convolutional networks.

2.3.3 Common techniques for Legged Locomotion Application

In the long history of designing RL-based locomotion controllers, researchers have developed a few important techniques for effective learning of real-world deployable locomotion policies.

Massively-Parallel Simulation One common concern for RL algorithms is the lack of sample efficiency. For example, to learn a simple blind walking policy, model-free RL algorithms like PPO may require hours or even days of data [20, 7, 45], which is cumbersome to collect in the real world. Therefore, most RL algorithms use simulation as a source of low-cost data collection. Recently, GPU-parallelized simulators like Isaacgym [46] significantly reduced the RL training time with massively parallel data collection. With these frameworks, researchers have demonstrated learning deployable locomotion policies in less than 20 minutes [19], which enables rapid design iterations for RL-based frameworks.

Domain Randomization One bottleneck for RL policies trained in simulation is the *sim-to-real gap*, where the policy overfits to the simulation environment and fails to perform as-well in the real world [7]. While several techniques like domain adaptation [47, 48] and automatic system identification [49] have been proposed to cross this gap, the most commonly

adopted technique is *domain randomization*. This technique trains the policy in randomly generated simulation environments with different dynamic parameters (e.g. body mass, friction coefficient, motor strength), and aims to learn a policy that is robust against these randomizations. Policies that have undergone such training tend to perform more robustly in the real world [7] with a small sim-to-real gap.

Teacher-Student Approach Since domain randomization aims to learn a *single* policy that performs well in all randomized environments, the learned policy can become overly conservative or even completely fail to function in heavily-randomized environments. As an alternative, recent works [8, 50, 18] adopt a *teacher-student* approach to learn policies that are *adaptive* to these dynamics shifts. The overall idea is to first train a *teacher policy* using RL, which takes in ground-truth information about the randomized dynamics so that the policy has sufficient information to make dynamics-aware decisions. That is, the body mass, motor strength, foot friction coefficient, etc. are all included in the state space of the teacher policy.

Once the teacher policy is trained, a *student policy* is then distilled from this teacher policy using imitation learning. Without access to the ground-truth dynamics information, these student policies will either learn to *infer* these parameters (or an embedding of these parameters), or directly learn to imitate the action of the teacher policy. The student policy can then be directly deployed to the real world. Similar frameworks have been adopted for the policy to learn *dynamics*-aware or *terrain*-aware policies, and have proved effective in a range of challenging locomotion tasks [].

2.3.4 Problems

Lastly, we summarize a few challenges remaining for RL-based locomotion controllers.

Sim-to-Real While techniques like domain randomization and teacher-student training have significantly reduced the sim-to-real gap of RL-based policies, this gap is still quite

dominant in highly-dynamic tasks such as jumping [51, 52] or running [53]. Due to inherently unmodeled or under-modeled real-world sim-to-real differences, such as motor characteristics and collision forces, the real world presents a qualitatively different environment that is out-of-distribution from the simulated environments. This is particularly evident for more agile tasks, where the robot need to make full use of the motor capability and the foot contact forces to prepare itself for highly dynamic motions.

Local Optima Another common problem for RL-based frameworks is that the policy can frequently become stuck in local optima. Since the RL algorithm does not have prior knowledge about the robot, it executes an "uniform" exploration across all possible actions in the action space, which may lead to a lot of infeasible action choices. For example, without proper turning, an RL algorithm might learn to propel the robot forward using high-frequency foot oscillations, which is not feasible in the real world. One way to alleviate this problem is to carefully craft a reward function to penalize these edge cases [8]. However, one usually needs to design a complex reward function with around 10 different terms to "regularize" the RL training process [8], which can be time-consuming, especially given the long iteration time of RL. Another way to alleviate this problem is to inject action priors. For example, Iscen et al. [20] introduced cyclic motion references to warm-start the RL training, which has proved effective in learning smooth, real-world deployable policies.

Part I

HIERARCHICAL LEARNING-CONTROL FOR AGILE LEGGED LOCOMOTION

In the first part of the thesis, we develop a general-purpose, hierarchical learning-control framework for agile legged locomotion. We start by designing a hierarchical framework for energy-efficient gait optimization, where a high-level gait policy outputs gait parameters from velocity inputs, and a low-level centroidal controller performs motor control based on the contact information. This is the first, general purpose hierarchical framework that effectively combines the expressiveness of reinforcement learning with the robustness of optimal control.

We then focus on the task of continuous, long-distance *jumping*, which requires careful coordination of the body motion, foot placement and contact timing. We present three works in this direction. For the first work, we learn a high-level *residual policy* on top of a heuristically design motion planner, and achieves stable, versatile, and omni-directional jumps on the real robot. Building on top of this result, we redesign the high-level policy to directly coordinate all aspects of body motion, and refactored the low-level controller so that the entire framework can be trained under GPU acceleration. Compared to previous frameworks, this work reduces the training time by at least an order of magnitude, and learns well-coordinated long-distance bounding gaits with large foot clearances. Lastly, we add perception to this framework, so that the robot can coordinate its jumping motions based on perceived terrain information.

While we primarily focus on jumping tasks in this part, note that this hierarchical framework can be extended to *general* locomotion gaits including crawling, trotting, and running. The perception component can also be used in confunction with arbitrary locomotion gaits.

Chapter 3

A HIERARCHICAL FRAMEWORK FOR FAST AND ENERGY EFFICIENT LOCOMOTION

3.1 Introduction

Fast and energy efficient locomotion is crucial for legged robots to accomplish tasks that traverse long distances. In the natural world, quadrupedal animals demonstrate a wide variety of distinctive locomotion patterns known as gaits [54], such as walking, trotting, bounding and galloping. Each gait is characterized by a unique foot contact schedule in a locomotion cycle. To lower their energy consumption, most quadrupedal animals switch to a preferred gait at each different speed range [54, 55]. While many locomotion gaits have been implemented on quadrupedal robots [3, 56], gait timing is often hand-engineered, and the switch between gaits is based on *ad-hoc* user commands. Can quadruped robots learn energy efficient gaits and natural gait transitions automatically?

In this work, we devise a learning framework in which energy-efficient locomotion controllers emerge automatically. The learned controllers naturally switch between different gaits at different speeds to maximize energy efficiency. Learning speed-adaptive locomotion gait controllers is challenging. Although reinforcement learning (RL) has been used to train policies end-to-end for a wide variety of continuous control tasks [44, 57, 58], these policies are often difficult to deploy safely on real robots without additional sim-to-real effort such as reward shaping [59], domain randomization [7, 60], or meta learning [48, 61]. Alternatively, optimal control based controllers have demonstrated robust performance on a number of quadruped robots [3, 4]. However, since gait patterns involve discrete contact events, it is difficult to optimize them together with other continuous forces. Therefore, most optimal control based controllers assume fixed, pre-selected gait timings.

To address the above challenges, we devise a hierarchical framework that combines the advantages of both RL and optimal control. This hierarchical framework consists of a high-level *gait generator* and a low-level *convex MPC controller*, which decouples the locomotion task into gait generation and motor control. Instead of directly outputting motor commands, the *gait policy* functions as part of the *gait generator* and specifies key gait parameters, which determines the contact schedule for each leg. Based on this contact schedule, the *convex MPC controller* then determines which legs are in contact, and computes the optimal motor command for each leg. We formulate the high-level gait policy learning as a Markov Decision Process (MDP), design a simple reward function based on velocity tracking and energy efficiency, and train the gait policy using evolutionary strategies (ES). We formulate the low-level convex MPC controller using model-predictive control with simplified dynamics [3].

With this hierarchical framework and the simple reward function, the gait policy automatically learns distinctive gait patterns at different locomotion speeds, including slow walking, mid-speed trotting and fast fly-trotting. Moreover, the policy automatically transitions from one gait to another to generate the most efficient gait at all speeds. Thanks to the robustness of our hierarchical framework, the learned gait policy can be deployed successfully on a Unitree A1 robot [39] in various environments (e.g. carpet, grass, short obstacle) without additional data collection and fine-tuning.

The main contributions of this paper include:

- A hierarchical learning framework effectively combines RL with optimal control, which can automatically learn fast and efficient locomotion controllers;

- The learned controllers switch gaits across a wide range of locomotion speeds, similar to those demonstrated in the animal kingdom;

- The learned controller can be deployed directly to the real world, and performs robustly in various environments.

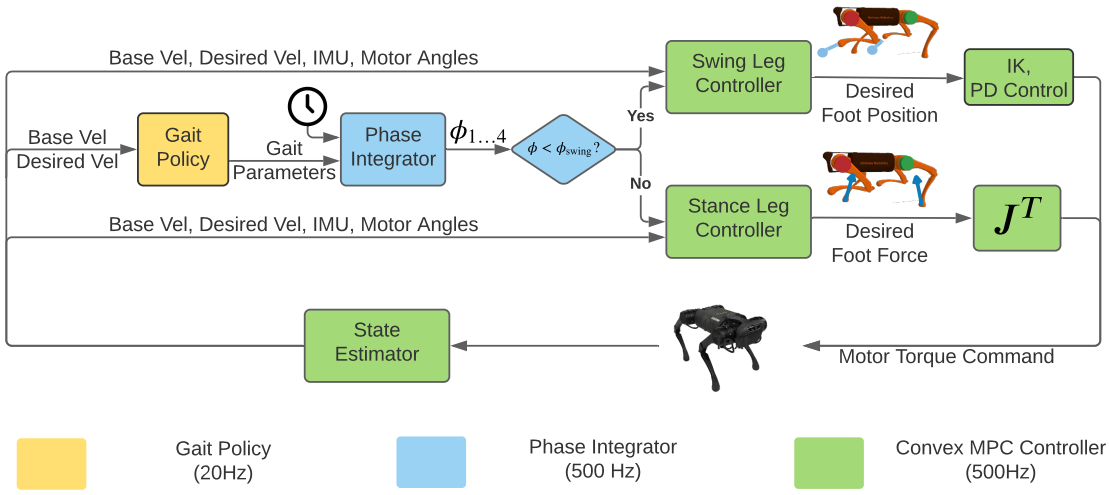


Figure 3.1: Our system consists of a high-level gait generator and a low-level convex MPC controller.

3.2 A Hierarchical Framework for Gait Optimization

3.2.1 Overview

To learn fast and efficient locomotion, we build a hierarchical framework with a high-level gait generator and a low-level convex MPC controller (Fig. 3.1). The high-level gait generator includes a learnable gait policy and a phase integrator. To generate a gait, the gait policy outputs gait parameters, such as frequency, to the *phase integrator*. Based on these parameters and the robot's clock, the phase integrator increments the phase for each leg and determines its contact state. In each locomotion cycle, the phase progresses from 0 to 2π as the foot goes from liftoff to touchdown to the next liftoff. The low-level convex MPC controller consists of separate controllers for swing and stance legs, and controls each leg differently based on its contact state. Additionally, we implement a Kalman Filter-based state estimator for the torso velocity, which cannot be measured directly using onboard sensors and is used by both the gait generator and the convex MPC controller. We run the high-level gait generator at 20Hz to avoid abrupt changes of gait commands, and the low

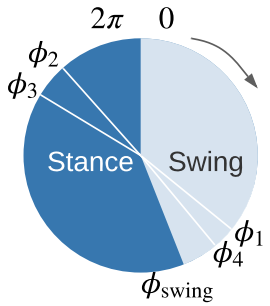


Figure 3.2: We use phases to represent the state of each leg. Each leg is assigned with an independent phase variable $\phi \in [0, 2\pi]$. ϕ_{swing} is the phase threshold that a leg switches from swing to stance. In the figure above, front-right (ϕ_1) and rear-left (ϕ_4) legs are in swing, while front-left (ϕ_2) and rear-right (ϕ_3) legs are in stance.

level controllers at 500Hz for fast replanning and stable torque control.

3.2.2 High-Level Gait Generation

A locomotion gait is determined by a *contact schedule*, the time and duration that each leg is in contact with the ground. To generate foot contact schedules, the phase integrator maintains a set of phase variables $\phi_{1,\ldots,4}$, one for each leg. The phase $\phi_i \in [0, 2\pi)$ denotes the leg's progress in its current gait cycle (Fig. 3.2). Each leg i starts with *swing* at the beginning of a gait cycle ($\phi_i = 0$). As ϕ_i increases monotonically, it switches to *stance* after a threshold $\phi_i > \phi_{\text{swing}}$. After ϕ_i reaches 2π, it wraps back to zero, and starts a new gait cycle in the *swing* phase. The propagation of phase variables, as well as the transition from *swing* to *stance*, are controlled by three key parameters, including stepping frequency f, swing ratio p_{swing} and phase offsets $\theta_2, \theta_3, \theta_4$, which are specified by the gait policy. We choose such parameterization because it is expressive enough to represent a rich set of locomotion gaits. We now describe these parameters in detail:

Stepping Frequency As a notable feature in locomotion, the stepping frequency is usually adjusted as a trade-off between speed and efficiency. While a high stepping fre-

quency allows the robot to run faster, stepping unnecessarily fast can cause additional energy consumption due to excessive leg swing. In our setup, the gait policy outputs the desired leg frequency $f \in (0\text{Hz}, 4\text{Hz}]$, which is used to increment the phase variable in the phase integrator. Specifically, at each control step, the phase is advanced by:

$$\phi[n] \leftarrow \phi[n-1] + 2\pi f \Delta t \tag{3.1}$$

where Δt is the time step of low-level controller (0.002s).

Swing Ratio Another important characteristic of gaits is the proportion of swing time in each gait cycle. For example, while a walking gait is usually characterized by spending less than 50% of a gait cycle in swing phase, a running gait typically requires a much longer swing time. To model this, we define another variable, $p_{\text{swing}} \in (0, 1)$, which controls the switching point in phase $\phi_{\text{swing}} = 2\pi p_{\text{swing}}$ between swing ($\phi < \phi_{\text{swing}}$) and stance ($\phi \geq \phi_{\text{swing}}$) in each gait cycle. Assuming that a leg moves at constant frequency, a larger p_{swing} means that it will spend more time in air in a gait cycle, which usually results in a more dynamic gait.

Phase Offsets Apart from careful design of individual gait cycles, careful coordination among legs is another critical component for efficient locomotion. While we use the same f and p_{swing} across all legs, we allow each individual leg to have a different phase offset. Let $\theta_i \in [0, 2\pi]$ denote the phase offset of leg i compared to first leg (the front-right leg); then the phase of leg i is $\phi_i = \phi_1 + \theta_i$. Note that the order of legs is [front-right, front-left, rear-right, rear-left], or [FR, FL, RR, RL] for short. For example, setting $\theta_4 = 0$ would make the rear-left leg in sync with the front-right leg, which is frequently seen in trotting gaits.

3.2.3 Low-level Optimal Control

The low-level convex MPC controller computes and applies torques for each actuated degree of freedom, given the leg phases from the high-level gait generator. Our low-level convex MPC controller is based on Di Carlo et al. [3]. We briefly describe the controller here for the completeness of the paper. Please refer to the Section. 3.7.1 for more details.

Stance Leg Control In the stance leg controller, we model the robot dynamics based on the Centroidal Dynamics Model [3], where the full robot is simplified as a rigid-body base with massless legs. Each stance leg can generate ground reaction force at the contact point, subject to torque limit and friction cone constraints. These ground reaction forces are solved as a short-horizon MPC problem whose objective is for the robot base to closely track a given reference trajectory. We generate the reference trajectory based on user-specified velocity commands. The optimized contact forces $\boldsymbol{f}$ are then converted to motor torques using the Jacobian transpose method: $\boldsymbol{\tau} = \mathbf{J}^T \boldsymbol{f}$. In our MPC setup, we re-optimize for ground reaction forces every time step (2ms), and only apply the first command in the optimized sequence.

Swing Leg Control The swing leg controller calculates the swing foot trajectories and uses Proportional-Derivative (PD) controllers to track these trajectories. The swing trajectory is computed by fitting a quadratic polynomial over the lift-off, mid-air and landing position of each foot, where the lift-off position is the foot location at the beginning of the swing phase, the landing position is calculated using the Raibert Heuristics [27], and the mid-air location is set to ensure the minimum ground clearance. Please refer to Appendix 3.7.1 for more details. Given the position in the swing trajectory, we convert it to the desired motor position using inverse kinematics, and apply motor torques using PD controllers.

3.3 Learning Gait Policies for Fast and Energy Efficient Locomotion

Since locomotion gait involves discrete contact events, it is difficult to model and optimize them together with other continuous forces. Instead, we formulate a Markov Decision Process and apply Evolutionary Strategies (ES) to discover the most energy efficient gaits at different speeds.

3.3.1 Preliminaries

The reinforcement learning problem is represented as a Markov Decision Process (MDP), which includes the state space $\mathcal{S}$, action space $\mathcal{A}$, transition probability $p(s_{t+1}|s_t, a_t)$, reward

function $r : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$, and initial state distribution $p_0(s_0)$. We aim to learn a policy $\pi : \mathcal{S} \mapsto \mathcal{A}$ that maximizes the expected cumulative reward over an episode of length T, which is defined as:

$$J(\pi) = \mathbb{E}_{s_0 \sim p_0, s_{t+1} \sim p(s_t, \pi(s_t))} \sum_{t=0}^{T} r(s_t, a_t) \tag{3.2}$$

3.3.2 MDP Formulation

In our MDP formulation, we only learn the gait policy in the high-level gait generator, and consolidate the other components, including phase integrator, the convex MPC controller, and the robot dynamics, into the environment. At each step, the gait policy outputs gait parameters and receives a reward, which is based on energy consumption and speed-tracking performance.

State and Action Space Since we aim to optimize the gaits based on the current and desired speed, we only include the desired and actual linear velocity of the base in the state space $\mathbf{s} = [\bar{v}_{\text{base}}, v_{\text{base}}]$. We do not include prioperceptive information, such as joint angles and IMU readings, because the detailed control of balance and locomotion, which consumes these information, is delegated to the low-level controller. The action space is a 5-dimensional vector $\mathbf{a} = [f, p_{\text{swing}}, \theta_2, \theta_3, \theta_4]$, as defined in Section 3.2.2.

Reward Design We design the reward function as a linear combination of survival bonus, speed-tracking penalty, and energy penalty:

$$r = c - w_v \underbrace{\left\| \frac{\bar{v}_{\text{base}} - v_{\text{base}}}{\bar{v}_{\text{base}}} \right\|^2}_{\text{Speed Penalty}} - w_e \underbrace{\frac{\sum_{i=1}^{12} \max(\tau_i \omega_i + \alpha \tau_i^2, 0)}{mg\bar{v}_{\text{base}}}}_{\text{Energy Penalty (Cost of Transport)}} \tag{3.3}$$

The survival bonus c prevents the learning algorithm from falling into the local minima of early termination. The speed penalty is the L2 norm of the relative error between the

desired ($\bar{v}_{\text{base}}$) and actual speed (v_{base}) of the base. The energy penalty estimates the Cost of Transport (CoT), a standard metric for measuring the efficiency of locomotion [62, 63, 64]. The numerator estimates total power consumption in all 12 motors based on the angular velocity (ω_i) and torque (τ_i) of each motor, and the motor parameter ($\alpha = 0.3$). (See Appendix 3.7.2 for details.) The denominator consists of the mass of the robot ($m = 15\text{kg}$), and gravity constant ($g = 9.8\text{m/s}^2$). In each episode, the desired velocity $\bar{v}_{\text{base}}$ starts at 0 m/s, increases linearly to 2.5m/s at 1m/s^2 and stays at 2.5m/s for the rest of the episode. We use the same weights $c = 3, w_v = 1$, $w_e = 0.37$ for all of our experiments.

Early Termination on Infeasible Gaits Despite the robustness of low-level whole body controller, the robot can still lose balance if the gait is infeasible, such as standing with one leg for an extended amount of time. To avoid unnecessary exploration in suboptimal gaits, we terminate an episode early if the robot falls (i.e. orientation deviates significantly from the upright pose, or the robot's height becomes too low).

3.3.3 Policy Representation and Training

We represent our policy as a neural network with one hidden layer of 256 units and tanh nonlinearities. We chose this network architecture because it is sufficiently expressive to learn different gaits, and structurally compact to be efficiently optimized by our learning algorithm. We train our policies using Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [65], a simple, parallelizable evolutionary algorithm that has been successfully applied to locomotion tasks [66, 67, 68]. Compared to other RL algorithms, CMA-ES performs exploration in the policy parameter space and does not require accurate value function estimation at every step [69, 70], which is well-suited for our complex hierarchical system.

3.4 Results

We design experiments to validate that our framework can learn fast and efficient locomotion controllers. Particularly, we aim to answer the following questions:

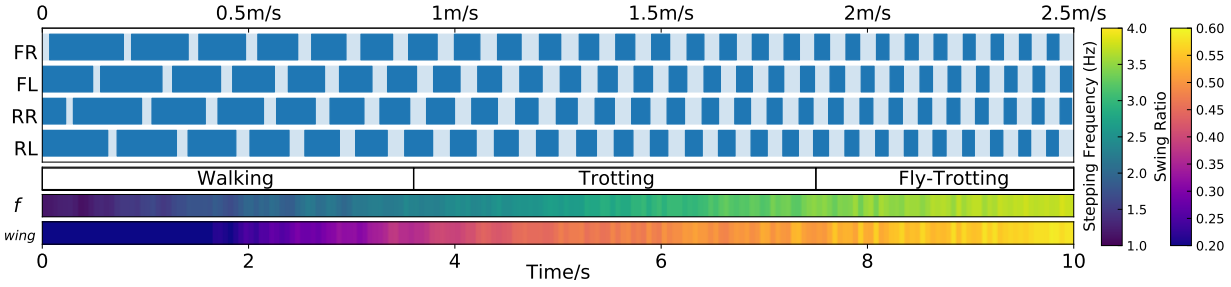


Figure 3.3: Robot gait during acceleration. Dark blue indicates foot contact. The robot switches from low-speed walking, to mid-speed trotting, to high-speed fly-trotting and increases the stepping frequency f and swing ratio p_{swing}.

	Freq (Hz)	Swing Ratio	Phase
Walk	2	0.3	$[0, \pi, 1.5\pi, 0.5\pi]$
Slow Trot	2	0.5	$[0, \pi, \pi, 0]$
Rapid Trot	4	0.5	$[0, \pi, \pi, 0]$
Fly Trot	4	0.6	$[0, \pi, \pi, 0]$

Table 3.1: Parameters of hand-tuned gaits shown in Fig. 3.4. The order of leg phases is [FR, FL, RR, RL].

1. Does our framework enable the robot to learn energy-efficient locomotion controllers?

2. Does the gait switching behavior, which is widely observed in animals, emerge naturally in the learning process?

3. Can the gait policy learned in simulation be deployed on real robots?

4. What are the advantages of our hierarchical framework and what are important design decisions?

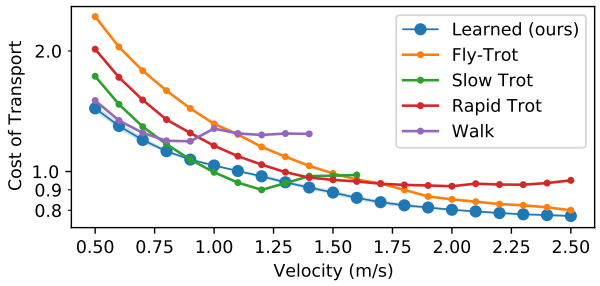


Figure 3.4: Cost of Transport (CoT) of hand-tuned gaits and our learned gait policy at different speeds. The CoT of learned gait is averaged over 5 random seeds. The standard deviation is too small to be visible in the figure.

3.4.1 Experiment Setup

We use the Unitree A1 robot [39], a small-scale, 15kg quadruped robot with 12 degrees of freedom. We use PyBullet [71] to simulate the robot dynamics. We implement the state estimation and high-level policy inference in Python, and the low-level Centroidal Dynamics-based Convex MPC Controller in C++. We use a Mac-Mini with M1 processor as our on-board computer, which runs the high-level gait generator at 20Hz and low-level convex MPC controller at 500Hz.

3.4.2 Emergence of Energy-Efficient Gaits

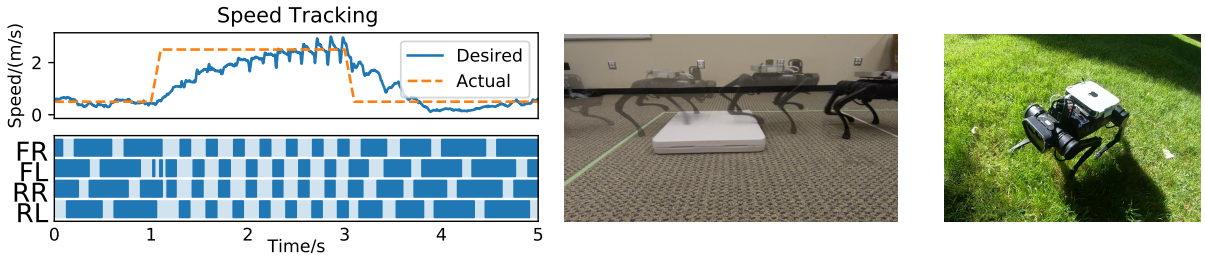
To demonstrate that our framework can learn efficient locomotion controllers, we evaluate the learned gait policy under different speed commands, and compare the Cost of Transport (CoT) with four carefully-designed, animal-inspired gaits: Walk, Slow Trot, Rapid Trot and Fly Trot. To find the parameters of these gaits, we first choose the phase offsets and swing-ratio based on the characteristics of each gait, and then perform grid search on stepping frequency to maximize the reward (Eq. 3.3) at different speed ranges. The parameters of these manually-designed gaits are summarized in Table. 3.1. We plot the CoT of each manually-designed gait and our learned controller (averaged over 5 random seeds) in Fig. 3.4.

Figure 3.5: Real world deployment of the learned policy. The robot moves at different speeds with different gaits.

Clearly, the learned policy consistently achieves the lowest energy consumption at most of the speeds.

A closer look at Fig. 3.4 reveals that walking gait is the most efficient when the speed is below 0.8m/s, the slow and rapid trotting gait are the most efficient between 0.8 and 2m/s, and the fly-trotting gait is the most efficient when the speed is above 2m/s. We execute the learned gait policy on an accelerating speed profile, and are glad to find that the policy switches gaits at similar boundaries (Fig. 3.3). At low speeds (less than 0.9m/s), the policy exhibits a four-beat *walking* gait by moving one leg at a time in the order of [RR, FL, RL, FR]. At intermediate speeds (between 0.9 and 1.8 m/s), the policy synchronizes the diagonal legs and exhibits a *trotting* gait. At the highest speeds (1.8m/s or above), the policy exhibits a *fly-trotting* gait, with noticeable "airborne-phases" when all legs lift from the ground (characterized by $p_{\text{swing}} > 0.5$).

Compared to gaits observed in quadrupedal animals, our gait policy discovered similar behaviors at low and mid-range speeds (walking and trotting). However, at high speeds, many animals would *gallop*. To understand why our method does not automatically learn galloping at higher speeds, we reproduced a galloping gait in our framework by limiting the range of phase offsets. However, we found the learned galloping gait to be actually 30% *less* efficient than the flying trot on our robot, in contrast to animals [54]. This difference between animals and robots has been noted in [63], and it was hypothesized that such difference could result from differences in morphology, kinematics limits and joint actuation.



(a) Gait switching at abrupt speed changes. (Success rate: 5/5)

(b) Walking over step (8cm). (Success rate: 4/5)

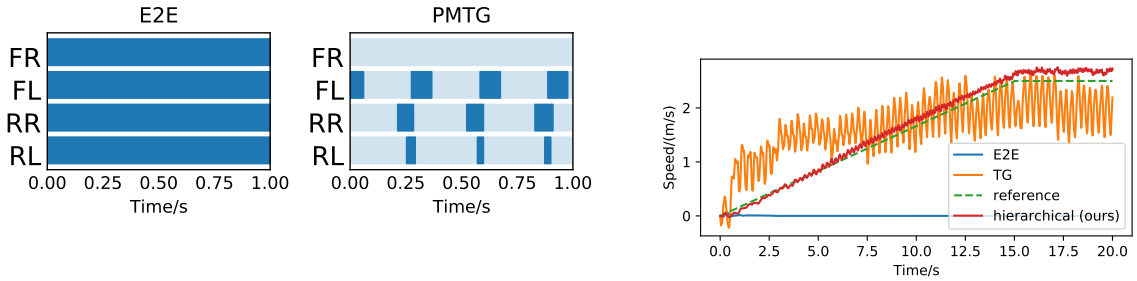
(c) Walking on grass. (Success rate: 4/5)

Figure 3.6: Deployment of gait policy to novel scenarios not encountered during training.

3.4.3 Validation on the Real Robot

We deploy our hierarchical controller, including the learned gait policy and the low-level model-based controllers, to the real robot (Fig. 3.5). Please watch the accompanying video. In contrast to many RL works that focus on sim-to-real transfer [7, 48, 61], our gait policy, learned entirely in simulation with PyBullet, can be directly deployed to the real world without additional data collection or fine-tuning. Similar to the simulation results, as the robot accelerates, it dynamically switches between walking, trotting and fly-trotting gaits, and eventually reached the speed of 2.5m/s, or 5 body lengths per second.

We test the generalization of our learned controller in a number of novel scenarios that were not seen during training. Although the policy is only trained using a slowly accelerating desired speed profile, the robot remains stable with abrupt changes of the desired speed, such as sharp acceleration and braking (Fig. 3.6a). The learned controller also generalizes to new terrains, including walking over an 8-cm step (Fig. 3.6b) and on grass (Fig. 3.6c). This excellent generalization is attribute to our hierarchical setup with a robust low-level convex MPC controller, which has demonstrated proven robustness on a wide variety of robot systems [3, 72, 73].



(a) Gaits Learned by Non-Hierarchical Policies.

(b) Speed Tracking Performance.

Figure 3.7: Gaits and speed-tracking performance of learned non-hierarchical policies that directly output motor commands. **Left**: The E2E policy falls into the local minima and stands in-place without moving. The PMTG policy moves forward using three legs. **Right**: Both policies do not track the desired speed as well as our hierarchical framework.

3.4.4 Comparisons with Non-Hierarchical Policies

To demonstrate the importance of the hierarchical setup, we compare the performance of our hierarchical framework with two non-hierarchical baselines: E2E and PMTG. E2E is a fully-connected end-to-end policy that directly maps from state to motor actions. PMTG implements the Policies Modulating Trajectory Generator [20], which simplifies learning by incorporating cyclic motion priors, and is widely used in the locomotion learning literature [74, 17, 45]. Following the prior work [75], we expand the state space to include IMU readings and motor angles, and modify the action space to be desired motor positions. Additionally, we carefully tuned the reward function in each baseline for optimal performance. We train all policies using the same CMA-ES algorithm.

We find that E2E only learns to stand, while PMTG learns an unnatural gait of using 3 legs, and cannot track the desired speed well (Fig. 3.7). This is likely due to the optimization landscape of the flat policy parameterization being more jaggy and the optimization converges to one of the bad local optima. While additional reward shaping could yield better results, our hierarchical system provides a simple alternative to effectively learn locomotion policies.

3.4.5 Comparisons with Different Learning Algorithms

Using Evolutionary Strategies (ES) to train our hierarchical controller is a critical design decision. To understand its importance, we compare CMA-ES with other state-of-the-art methods, including PPO [43], SAC [58] and ARS [44] in training the hierarchical controller. Since both PPO and SAC can benefit from more comprehensive state information, we also run these algorithms with an extended observation space, which includes IMU readings, motor angles and gait phases, in addition to current and desired velocities.

As shown in Table. 3.2, algorithms based on Evolutionary Strategies (ES), CMA-ES and ARS, significantly outperforms other algorithms. When using the original observation space, both PPO and SAC fail to complete the task, which is likely due to the lack of sufficient information to accurately estimate the value function. With the extended observation space, SAC learns to walk forward, but fails to track the speed closely, and consumes more energy, compared to ES-based algorithms. We hypothesize that the poor performance of these two popular reinforcement learning algorithms is because the low-level convex MPC controller is a black box to the RL agent, which is not fully-observable and make the environment less Markovian.

Algorithm	Success?	CoT	Avg Speed Tracking Error
PPO	No	2.29 ± 0.96	0.080 ± 0.092
PPO (extended obs)	No	3.60 ± 1.05	0.15 ± 0.17
SAC	No	2.22 ± 0.47	0.069 ± 0.025
SAC (extended obs)	Yes	1.19 ± 0.04	0.030 ± 0.025
ARS	Yes	$\mathbf{0.87 \pm 0.0073}$	$\mathbf{0.0082 \pm 0.00034}$
CMA-ES (ours)	Yes	$\mathbf{0.84 \pm 0.017}$	$\mathbf{0.0083 \pm 0.00075}$

Table 3.2: Cost of Transport (CoT) and speed tracking error (Eq. 3.3) for gait policies trained by different algorithms. Error bar shows 1 standard deviation.

3.5 Related Works

Quadrupedal animals demonstrate a wide variety of gaits [76]. Hoyt and Taylor [54] showed empirically that horses minimize their energy consumption when using the preferred gait at each speed. Alexander and Jayes [55] generalized this result by providing a unifying theory of gait transitions for quadrupedal animals. A wide variety of these gaits have been implemented in quadrupedal robots, including walking [77], trotting [3], pacing [56], bounding [78] and galloping [79]. In these works, model-based controllers [3, 80] optimize for motor commands at a high frequency. These controllers usually assume a pre-defined contact sequence to keep the optimization problem tractable, which does not allow gait transitions. Alternatively, contact implicit optimization [21, 81, 82] optimizes contact forces and sequences together, but is not feasible for real-time use due to high computation cost. Using manually designed heuristics, Boussema et al. [83] achieved online gait transition by computing the Feasible Impulse Set for each leg, at a speed up to 0.6m/s, or 1 body length/s. Owaki and Ishiguro [62] also demonstrated online gait transition on a 2kg quadruped robot using foot-force heuristics. Compared to these approaches, our learning-based approach requires less manual tuning, achieves more agile motions (up to 2.5m/s, or 5 body length/s) on a larger robot (15kg).

Recently, reinforcement learning became a popular approach to learn locomotion policies for legged robots [7, 49, 84]. Since policies are often learned in simulation, extra effort is usually required to transfer the learned policies to the real robot, including building more accurate simulation [7, 49], dynamic randomization [7, 60], motion imitation [85, 86] and meta learning [48, 61]. Inspired by the periodicity of locomotion behaviors, several methods have been proposed to make the learned policy more predictable and safer for real robot deployment, such as cyclic trajectory generators [20], phase-functioned neural networks [87] and state machines [88]. In contrast to these previous works, our learned policy achieves zero-shot sim-to-real transfer, and the controller is robust in multiple real-world environments.

Compared to directly learning an end-to-end controller, hierarchical learning [89] can

improve data efficiency, and achieve complex tasks. The low-level controller can be a learned policy [74, 90, 60], or a hand-tuned controller [91, 92, 93]. Li et al. [92] uses a learned policy to modulate objectives of the low-level MPC controller, with a fixed contact sequence. Recently, Da et al. [91] looked into learning hierarchical controllers for gait control in quadrupeds, where the low-level controller is model-based and high-level policy selects from a fixed set of gait primitives. We use a similar hierarchical setup as Da et al. [91]'s, but extend the high-level policy to search a continuous range of gaits with arbitrary gait changes, and achieve significantly faster walking.

3.6 Discussion

We present a hierarchical learning framework that can automatically learn fast and efficient gaits for quadrupedal robots. Our framework combines a high-level gait generator with a low-level convex MPC controller, where a gait policy is trained using evolutionary strategies with a simple reward function. Through learning, distinctive gaits and natural transitions between gaits emerge automatically. More importantly, the policy learned in simulation can be successfully deployed to the real world, thanks to the hierarchical setup. Observations in our robotic experiments agree well with prior bio-mechanical studies, which showed that quadrupedal animals switch gaits at different speeds to lower their energy expenditure. Our hierarchical control framework is general, and can be extended to modulate not only the gait patterns, but also other parts of the low-level controller, such as foot placement positions and desired base pose, to enable more agile and versatile locomotion skills. We plan to further develop this hierarchical framework through the lens of bi-level optimization, and apply it to other robotic platforms.

(a) The stance controller optimizes ground reaction forces $\boldsymbol{f}_{1,\ldots,4}$ to track a desired base trajectory.

(b) The swing controller tracks the leg on a quadratic curve, which is fitted using $(\boldsymbol{p}_{\text{lift-off}}, \boldsymbol{p}_{\text{air}}, \boldsymbol{p}_{\text{land}})$.

Figure 3.8: Our low-level convex MPC controller uses different controllers for stance (**left**) and swing (**right**) legs.

3.7 Appendix

3.7.1 Details of the Low-Level Convex MPC Controller

Stance Leg Controller

The stance leg controller optimizes for the ground reaction forces using Model Predictive Control (MPC) (Fig. 3.8a), where the objective is for the base to track a desired trajectory. The robot is modeled using the simplified centroidal dynamics model. We now describe our setup in detail:

Notation We represent the base pose of the robot in the world frame as $\boldsymbol{x} = [\boldsymbol{\Theta}, \boldsymbol{p}, \boldsymbol{\omega}, \dot{\boldsymbol{p}}] \in \mathbb{R}^{12}$. $\boldsymbol{\Theta} = [\phi, \theta, \psi]$ is the robot's base orientation represented as Z-Y-X Euler angles, where ψ is the yaw, θ is the pitch and ϕ is the roll. $\boldsymbol{p} \in \mathbb{R}^3$ is the Cartesian coordinate of the base position. $\boldsymbol{\omega}$ and $\dot{\boldsymbol{p}}$ are the linear and angular velocity of the base. $\boldsymbol{r}_{\text{foot}} = (\boldsymbol{r}_1, \boldsymbol{r}_2, \boldsymbol{r}_3, \boldsymbol{r}_4) \in \mathbb{R}^{12}$ represents the four foot positions relative to the robot base. MPC optimizes for the ground

reaction force $\boldsymbol{f}_{1,\ldots,4}$ at each foot, which we denote as $\boldsymbol{u} = (\boldsymbol{f}_1, \boldsymbol{f}_2, \boldsymbol{f}_3, \boldsymbol{f}_4) \in \mathbb{R}^{12}$. $\mathbf{I}_n$ denotes the $n \times n$ identity matrix. $[\cdot]_\times$ converts a 3d vector into a skew-symmetric matrix, so that for $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^3$, $\boldsymbol{a} \times \boldsymbol{b} = [\boldsymbol{a}]_\times \boldsymbol{b}$.

Centroidal Dynamics Model Our centroidal dynamics model is based on [3] with a few modifications. We assume massless legs, and simplify the robot base to a rigid body with mass m and inertia $\mathbf{I}_{\text{base}}$ (in the body frame). The rigid body dynamics in world coordinates are given by:

$$\frac{\mathrm{d}}{\mathrm{d}t}(\mathbf{I}_{\text{world}}\boldsymbol{\omega}) = \sum_{i=1}^{4} \boldsymbol{r}_i \times \boldsymbol{f}_i \tag{3.4}$$

$$\ddot{\boldsymbol{p}} = \frac{\sum_{i=1}^{4} \boldsymbol{f}_i}{m} + \boldsymbol{g} \tag{3.5}$$

where $\boldsymbol{g} = [0, 0, -9.8]^T$ is the gravity vector. To simplify Eq.3.4, note that when angular velocity is small, we can omit the Coriolis forces and write the left hand side as:

$$\frac{\mathrm{d}}{\mathrm{d}t}(\mathbf{I}_{\text{world}}\boldsymbol{\omega}) = \mathbf{I}_{\text{world}}\dot{\boldsymbol{\omega}} + \omega \times (\mathbf{I}_{\text{world}}\omega) \approx \mathbf{I}_{\text{world}}\dot{\boldsymbol{\omega}} \tag{3.6}$$

Given the robot the orientation matrix in the world frame $\mathbf{R} \in SO(3)$, the world-frame inertia is:

$$\mathbf{I}_{\text{world}} = \mathbf{R}\mathbf{I}_{\text{base}}\mathbf{R}^T \tag{3.7}$$

When the robot is close to upright ($\theta, \phi \approx 0$), the relationship between the angular velocity and the change rates of Euler angles can be written as:

$$\dot{\boldsymbol{\Theta}} = \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \approx \begin{bmatrix} \cos(\psi) & \sin(\psi) & 0 \\ -\sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \boldsymbol{\omega} = \mathbf{R}_z(\psi)\boldsymbol{\omega} \tag{3.8}$$

With the above simplifications, we get the linear, time-varying dynamics model:

$$\underbrace{\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix}\boldsymbol{\Theta}\\ \boldsymbol{p}\\ \boldsymbol{\omega}\\ \dot{\boldsymbol{p}}\end{bmatrix}}_{\dot{\boldsymbol{x}}_{\text{base}}} = \underbrace{\begin{bmatrix}\mathbf{0}_3 & \mathbf{0}_3 & \mathbf{R}_z(\psi) & \mathbf{0}_3\\ \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{1}_3\\ \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3\\ \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3\end{bmatrix}}_{\mathbf{A}} \underbrace{\begin{bmatrix}\boldsymbol{\Theta}\\ \boldsymbol{p}\\ \boldsymbol{\omega}\\ \dot{\boldsymbol{p}}\end{bmatrix}}_{\boldsymbol{x}_{\text{base}}} + \underbrace{\begin{bmatrix}\mathbf{0}_3 & \dots & \mathbf{0}_3\\ \mathbf{0}_3 & \dots & \mathbf{0}_3\\ \mathbf{I}_{\text{world}}^{-1}[\boldsymbol{r}_1]_\times & \dots & \mathbf{I}_{\text{world}}^{-1}[\boldsymbol{r}_1]_\times\\ \mathbf{I}_3/m & \dots & \mathbf{I}_3/m\end{bmatrix}}_{\mathbf{B}} \underbrace{\begin{bmatrix}\boldsymbol{f}_1\\ \boldsymbol{f}_2\\ \boldsymbol{f}_3\\ \boldsymbol{f}_4\end{bmatrix}}_{\boldsymbol{u}} + \begin{bmatrix}\mathbf{0}\\ \mathbf{0}\\ \mathbf{0}\\ \boldsymbol{g}\end{bmatrix} \tag{3.9}$$

We then discretize the continuous time dynamics equation, which we use in our MPC formulation.

$$\boldsymbol{x}_{t+1} = \mathbf{A}'\boldsymbol{x}_t + \mathbf{B}'\boldsymbol{u}_t + \boldsymbol{g}' \tag{3.10}$$

where $\mathbf{A}'$, $\mathbf{B}'$, $\boldsymbol{g}'$ are the discrete time counterpart of $\mathbf{A}$, $\mathbf{B}$ and $\boldsymbol{g}$ in Eq. (3.9).

Reference Trajectory Generation Given the desired linear velocity of the base $\bar{\boldsymbol{v}}_{\text{base}}$, we compute a desired trajectory $\bar{\boldsymbol{x}}_t$ for the next T timesteps, where T is the MPC planning horizon. In each reference state, we set $\bar{\dot{\boldsymbol{p}}}$ to $\bar{v}_{\text{base}}$ and set $\bar{\boldsymbol{p}}$ to numerically integrate $\bar{v}_{\text{base}}$ for speed-tracking, and set the desired orientation $\bar{\boldsymbol{\Theta}}$ and angular velocity to $\bar{\boldsymbol{\omega}}$ to 0 to ensure stable walking.

MPC Formulation Given the reference trajectory $\bar{\boldsymbol{x}}_{1,\dots,T}$, we solve for the ground reaction forces $\boldsymbol{u}_{1,\dots,T}$ by solving the following Quadratic Program (QP):

$$\min_{\boldsymbol{u}_{1,\dots,T}} \sum_{t=1}^{T} \|\boldsymbol{x}_t - \bar{\boldsymbol{x}}_t\|_{\mathbf{Q}} + \|\boldsymbol{u}_t\|_{\mathbf{R}} \tag{3.11}$$

$$\begin{aligned}
\text{subject to } \boldsymbol{x}_{t+1} &= \mathbf{A}'\boldsymbol{x}_t + \mathbf{B}'\boldsymbol{u}_t + \boldsymbol{g}' && \text{Eq. (3.10)} \\
f^z_{i,t} &= 0 && \text{if leg } i \text{ is a swing leg at } t \\
f_{\min} \le f^z_{i,t} &\le f_{\max} && \text{if leg } i \text{ is a stance leg at } t \\
-\mu f^z_{i,t} \le f^x_{i,t} &\le \mu f^z_{i,t} && \forall i, t \\
-\mu f^z_{i,t} \le f^y_{i,t} &\le \mu f^z_{i,t} && \forall i, t
\end{aligned}$$

where $\mathbf{Q}$, $\mathbf{R}$ are diagonal weight matrices. The constraints include the centroidal dynamics, the contact schedule of each leg and the approximated friction cone conditions. The optimized contact forces are then converted to motor torques using Jacobian transpose: $\boldsymbol{\tau} = \mathbf{J}^T \boldsymbol{f}$.

Swing Leg Control

The swing leg controller calculates the swing foot trajectories and uses Proportional-Derivative (PD) controllers to track these trajectories (Fig. 3.8b). To calculate a leg's swing trajectory, we first find its lift-off, mid-air and landing positions ($\boldsymbol{p}_{\text{lift-off}}, \boldsymbol{p}_{\text{air}}, \boldsymbol{p}_{\text{land}}$) (Fig. 3.8b). The lift-off position $\boldsymbol{p}_{\text{lift-off}}$ is the foot location at the beginning of the swing phase. The mid-air position $\boldsymbol{p}_{\text{air}} = \boldsymbol{p}_{\text{ref}} + (0, 0, z_{\text{des}})$ is a fixed distance above the normal standing position $\boldsymbol{p}_{\text{ref}}$. We use the Raibert Heuristic [27] to estimate the desired foot landing position:

$$\boldsymbol{p}_{\text{land}} = \boldsymbol{p}_{\text{ref}} + \boldsymbol{v}_{\text{CoM}} T_{\text{stance}}/2 \tag{3.12}$$

where $\boldsymbol{v}_{\text{CoM}}$ is the projected robot's CoM velocity onto the $x-y$ plane, and T_{stance} is the expected duration of the next stance phase, which can be calculated using the stepping frequency and swing ratio from the gait policy (Section 3.2.2). Raibert's heuristic ensures that the stance leg will have equal forward and backward movement in the next stance phase, and is commonly used in locomotion controllers [56, 3, 4].

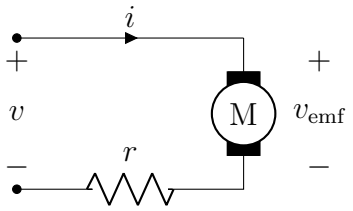


Figure 3.9: Schematic drawing of DC motor model.

Given these three key points, $\boldsymbol{p}_{\text{lift-off}}, \boldsymbol{p}_{\text{air}}$, and $\boldsymbol{p}_{\text{land}}$, we fit a quadratic polynomial, and computes the foot's desired position in the curve based on its progress in the current swing phase. Given the desired foot position, we then compute the desired motor position using inverse kinematics, and track it using a PD controller. We re-compute the desired foot position of the feet at every step (500Hz) based on the latest velocity estimation.

3.7.2 Modeling Motor Power Consumption

DC Motor Model

We model a DC motor circuit as in Fig. 3.9, which includes a motor with internal resistance r and torque constant k. To apply a torque τ_m, the motor controller applies a voltage v to the motor, which generates a current i. As the motor rotates with angular velocity ω_m, it also generates a back-emf voltage v_{emf}. We aim to express the battery power consumption $p = vi$ in terms of the motor velocity ω_m and applied motor torque τ_m. If we ignore motor inductance and only consider steady-state behaviors, the circuit characteristic can be written as:

$$\tau_m = ki \tag{3.13}$$

$$v_{\text{emf}} = k\omega_m \tag{3.14}$$

$$i = \frac{v - v_{\text{emf}}}{r} \tag{3.15}$$

where Eq.3.13 and Eq.3.14 models steady-state motor behavior, and Eq.3.15 is derived

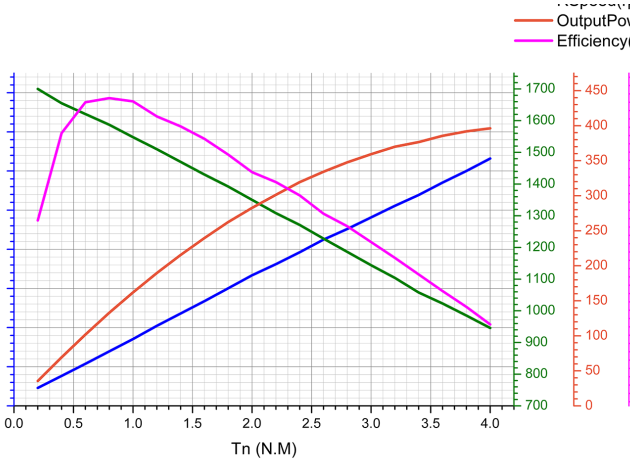


Figure 3.10: Characteristic curve for A1 motors [39] from robot manufacturer. The angular velocity and output torque are measured at the motor level without gear reduction.

from Ohm's law. Solving for v in terms of τ_m, ω_m and motor constants k, r, we get:

$$v = k\omega_m + \frac{\tau_m r}{k} \tag{3.16}$$

The power supplied by the battery can be computed by:

$$p = vi = \left(k\omega_m + \frac{\tau_m r}{k}\right)\frac{\tau_m}{k} = \tau_m\omega_m + \frac{r}{k^2}\tau_m^2 \tag{3.17}$$

Note that the first term is the *mechanical* power delivered by the motor, and the second term is the extra *heat* dissipation in the motor circuit. Since Unitree's battery management system does not support regenerative braking, we lower-bound the power consumption by 0, and get:

$$p_{\text{actual}} = \max\left(\tau_m\omega_m + \frac{r}{k^2}\tau_m^2, 0\right) \tag{3.18}$$

Power Consumption of A1 motors

Based on the motor characteristic curve of A1 (Fig. 3.10), at $\tau_m = 4$Nm and the output power is approximately 400w, with an efficiency of approximately 50%. Therefore, $\tau_m \omega_m \approx \frac{r}{k^2}\tau_m^2 \approx 400$w. We can then deduce that $\frac{r}{k^2} \approx 25$ and express power consumption as:

$$p_{\text{actual}} \approx \max(\tau_m \omega_m + 25\tau_m^2, 0) \tag{3.19}$$

The motor of A1 have a gear reduction ratio of 9.1. Therefore the joint velocity and joint torque (τ, ω) can be expressed in terms of motor velocity and motor torque (τ_m, ω_m) as:

$$\tau = 9.1\tau_m$$
$$\omega = \frac{\omega_m}{9.1}$$

Substituting into Eq. 3.19, we can express the power consumption in terms of joint torque and velocity:

$$p_{\text{actual}} \approx \max\left(\tau\omega + \frac{25}{9.1^2}\tau^2, 0\right) \approx \max(\tau\omega + 0.3\tau^2, 0) \tag{3.20}$$

Chapter 4

VERSATILE JUMPING WITH LEARNED ACTION RESIDUALS

4.1 Introduction

Jumping can greatly extend the capabilities of legged robots. Compared to walking, jumping exhibits a long "air phase", where all legs leave the ground at the same time. This air phase enables the robot to traverse through large areas without making contact, which is essential for difficult terrains with large gaps or abrupt height changes. While recent works have greatly improved the speed [3, 10, 53] and robustness [8, 18, 94] of legged robots, most of them focused on standard walking behaviors with continuous foot contacts. Meanwhile, long-distance jumping is still a difficult task, and usually requires manual trajectory design [95, 5] as well as long periods of offline planning [96, 13].

In this work, we present a hierarchical learning framework for quadrupedal robots to jump continuously, where the jumping direction and distance can be specified online. Continuous jumping has long been a difficult task for legged robots due to complex robot dynamics and frequent, abrupt contact changes. On the one hand, while optimal control based controllers have achieved robust walking in many quadruped platforms [3, 77], they usually assume a simplified dynamics model for computation efficiency [3], and cannot control the robot pose precisely in highly dynamic motions like jumping [96]. On the other hand, despite recent success in learning for locomotion [8, 53, 94], reinforcement learning (RL) based controllers still require careful reward tuning [86] and extended training times to learn jumping motions, due to the non-smooth reward landscape created by abrupt contact changes. Therefore, it can be difficult to use standard control or learning techniques for the jumping task.

Our framework addresses the challenges above, and learns continuous, versatile jumping

motions that can be transferred directly to the real world. The core of our framework is a *stance controller*, which computes desired body pose by summing over the outputs from a manually-designed *acceleration controller* and a learned *residual policy*. Our design of the stance controller has two major benefits. Firstly, warm-starting the policy training with the acceleration controller reduces noises in the reward landscape, so that training process converges smoothly and efficiently. Secondly, with the residual policy trained, the robot performance is no longer limited by the simplified dynamics model used by the acceleration controller, and therefore can better stabilize the robot throughout the entire jumping episode. In addition to the stance controller, we also implemented a low-level *whole-body controller* to convert the body pose command to motor actions. By combining the acceleration controller, the residual policy and the low-level whole-body controller, our framework learns continuous, versatile jumping motions automatically.

We train our framework on a simulated environment of the Go1 quadruped robot from Unitree [32], and test the trained policy directly on the real robot. The trained framework enables versatile jumping motions for the robot, including jumping at different directions and distances (up to 50cm high, 60cm forward), and jump-turning (up to 90 degrees). We then conduct detailed analysis on the behavior of our overall framework, and verify that the combination the acceleration controller and residual policy can learn more stable jumping motions than each individual method. Additionally, we compare our method to end-to-end RL and find that our method is at least 1 order-of-magnitude more data efficient, thanks to the hierarchical setup and the smooth reward landscape from the acceleration controller.

In summary, the contributions of this paper include the following:

1. We propose a hierarchical framework that combines optimal control and reinforcement learning to learn continuous, versatile jumping for quadrupedal robots.

2. The trained framework can be directly transferred to the real robot and achieves continuous jumping motions at substantial height (50cm) and distance (60cm).

3. Our experiments show that the combination of controller and residual policy can learn more stable jumping motions than using either method individually.

4.2 *Learning a Residual Policy for Continuous Versatile Jumping*

4.2.1 Overview

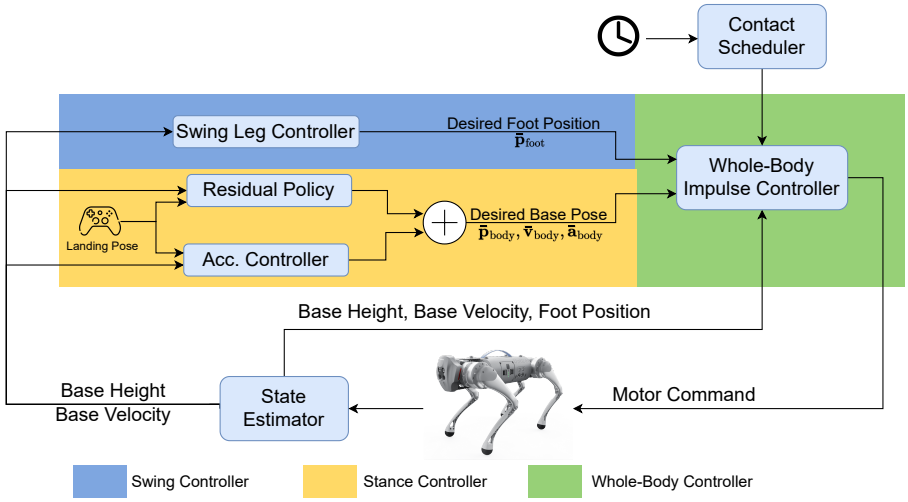


Figure 4.1: Our hierarchical learning framework controls the jumping of the Go1 robot from two different levels.

We design a hierarchical framework to learn continuous robot jumping (Fig. 4.1). The timing of each jump is regulated by an open-loop time-based *contact scheduler*, which subsequently specifies the desired state of each leg (*swing* or *stance*). We adopt the pronking gait, where all legs enter and leave the ground at the same time, and set the duration of each jump to be 1 second, which consists of 0.5s of stance time and 0.5s of swing time. Based on the output from the contact scheduler, we use separate control strategies for swing and stance legs, where the *stance controller* controls the desired body pose and *swing controller* controls the foot positions. At the low level, a whole-body controller converts the pose command from the swing or stance controller to motor commands. We also implemented a Kalman Filter

based state estimator to estimate the position and velocity of the robot. We run the entire pipeline at 500Hz so that the robot can respond quickly to external perturbations.

As a critical component of the entire control process, the stance controller needs to achieve sufficient lift-off speed for each jump, while maintaining body stability throughout the entire jump. To achieve that, we compute the pose command of the stance controller as the sum of a manually designed *acceleration controller* and a learned *residual policy*, where the acceleration controller computes base acceleration to ensure sufficient lift-off velocity based on simplified robot dynamics, and the residual policy fine-tunes the controller's action to ensure stability. Since the robot is underactuated in the air, we use a simple trajectory controller for swing legs, which computes the desired landing position of each feet according to the Raibert Heuristic [27].

4.2.2 Low-level Whole-body Controller

At the low-level, we use a whole-body controller (WBC) to convert the body and foot pose commands into motor actions. Our implementation is based on the work by [10] with a few modifications. We briefly summarize the controller design here. Please refer to the original work for further details.

Interface with Stance Controller In summary, WBC takes in a 18-dimensional vector that specifies the desired pose $\boldsymbol{p}_{\text{body}}$, velocity $\boldsymbol{v}_{\text{body}}$, and acceleration $\boldsymbol{a}_{\text{body}}$ for each of the 6 DoFs of the robot *body*, as well as the foot swing positions $\boldsymbol{p}_{\text{foot}}$. The foot swing positions $\boldsymbol{p}_{\text{foot}}$ is directly specified by the swing controller (Section. 4.2.1). The stance controller (Section. 4.2.4) specifies 4 out of the 6 dimensions for the base accelerations $\boldsymbol{a}_{\text{body}}$ (3 linear accelerations and angular accelerations around the z axis), as well as 2 out of the 6 dimensions of the base pose $\boldsymbol{p}_{\text{body}}$ (roll and pitch). For the remaining pose commands, we set the desired body pose p_{body} to be the current body pose, the desired linear body velocity to the current velocity, the desired angular body velocity to 0, and the desired angular acceleration around the x, y axis to 0.

Computation of Motor Commands WBC computes an impedance command that specifies the desired position $\bar{\boldsymbol{q}}$, velocity $\bar{\dot{\boldsymbol{q}}}$ and torque $\bar{\boldsymbol{\tau}}$ for each *motor*. The applied torque $\boldsymbol{\tau}$ is the sum of the desired torque plus the PD feedback:

$$\boldsymbol{\tau} = k_p(\bar{\boldsymbol{q}} - \boldsymbol{q}) + k_d(\bar{\dot{\boldsymbol{q}}} - \dot{\boldsymbol{q}}) + \bar{\boldsymbol{\tau}} \tag{4.1}$$

where $\boldsymbol{q}, \dot{\boldsymbol{q}}$ is the current position and velocity of each motor, and k_p, k_d are fixed gains. To compute the motor command, WBC first applies an inverse kinematics algorithm, which computes the desired position $\bar{\boldsymbol{q}}$ and velocity $\bar{\dot{\boldsymbol{q}}}$ for each motor, in order to move the robot to the desired body position $\boldsymbol{p}_{\text{body}}$ and velocity $\boldsymbol{v}_{\text{body}}$. After that, WBC computes the additional motor torque $\bar{\boldsymbol{\tau}}$ required to achieve the desired base accelerations $\boldsymbol{a}_{\text{body}}$, based on the full rigid-body dynamics model of the robot.

4.2.3 Manually-designed Acceleration Controller

Due to the discrete contact change, the reward landscape in the jumping environment can be highly non-smooth with local minima, which makes it challenging to learn using standard exploration strategies in RL algorithms. To facilitate learning, we manually design an acceleration controller as the base policy for the environment, and uses reinforcement learning to learn *residual actions* to finetune the policy's performance. The acceleration controller models the robot body as a single point mass, computes the desired lift-off velocity based on contact timing, and tracks this lift-off velocity using simple heuristics.

Computing the Lift-off Velocity The acceleration controller computes the desired lift-off velocity $\boldsymbol{v}_{\text{liftoff}} = (v_x, v_y, v_z, v_{\text{yaw}})$ based on the desired landing displacement p_x, p_y, p_{yaw} and the swing time t_{swing}. The planar velocities, $v_x = \frac{p_x}{t_{\text{swing}}}, v_y = \frac{p_y}{t_{\text{swing}}}, v_{\text{yaw}} = \frac{p_{\text{yaw}}}{t_{\text{swing}}}$, is the average flying speed required for the robot to land at the desired position and orientation. The vertical velocity, $v_z = \frac{1}{2} g t_{\text{swing}}$, is the minimum vertical velocity for the robot to maintain the desired swing time, where g is the gravity constant.

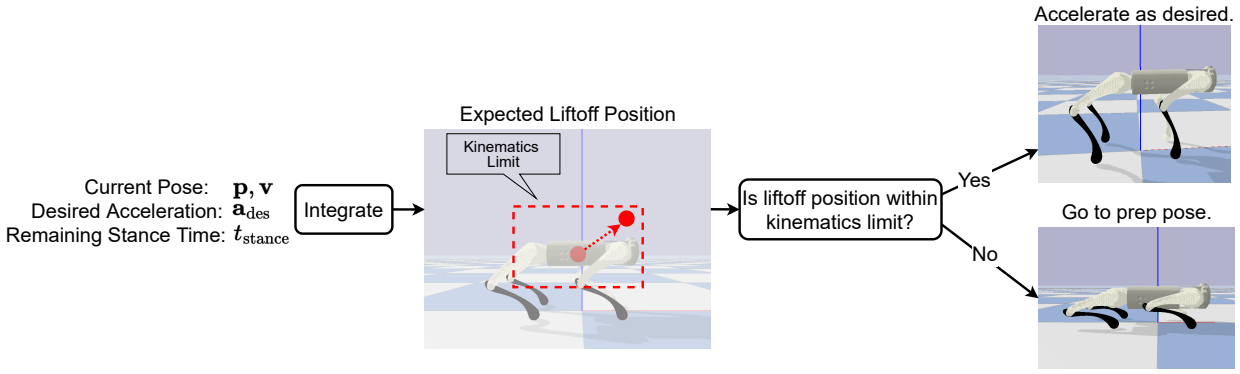


Figure 4.2: The acceleration controller decides whether to execute the desired acceleration command $\boldsymbol{a}_{\text{des}}$ based on the estimated lift-off position, which is computed by numerical integration. If the lift-off position (dark red dot) is within the approximated kinematics limit (dashed square), the controller executes the $\boldsymbol{a}_{\text{des}}$. Otherwise the controller moves the robot to a low-standing preparation pose.

Tracking the Lift-off Velocity To track the desired lift-off velocity $\boldsymbol{v}_{\text{liftoff}}$, the acceleration controller computes the desired acceleration $\boldsymbol{a}_{\text{des}} = \frac{\boldsymbol{v}_{\text{liftoff}} - \boldsymbol{v}}{t}$ based on the remaining time in the stance phase t, the desired liftoff velocity $\boldsymbol{v}_{\text{liftoff}}$ and the current CoM velocity $\boldsymbol{v}$. The acceleration controller then either executes this acceleration $\boldsymbol{a}_{\text{des}}$, or moves the robot to a preparation position, based on an estimation of the lift-off position (Fig. 4.2). More specifically, the acceleration controller assumes the robot as a point-mass, and computes the body's CoM position at lift-off time based on the current pose and desired accelerations. If the lift-off position violates kinematics limits (e.g. if the base is so high that foot contact cannot be maintained), the controller computes an alternative acceleration that moves the robot to a low-standing preparation position. Otherwise, the controller outputs the desired acceleration $\boldsymbol{a}_{\text{des}}$. To simplify computation, we approximate the feasible CoM positions as a bounding box around the robot's current CoM.

4.2.4 Reinforcement Learning for Action Residuals

Environment setup We design our environment so that the robot can jump in several directions within each episode. More specifically, each episode consists of 5 jumps, where the robot jumps in-place, 1m forward, 0.5m backward, 0.2m to the left, and 0.2m to the right, in that order. Before each jump, the environment records the robot's current position and computes the desired landing position based on the jumping directions. This information is then supplied to the residual policy to compute the desired pose commands, which is subsequently sent to the low-level whole-body controller. The details of our environment are as follows:

State Space The state space includes the robot state and task information. More specifically, the robot state includes the position, orientation, linear velocity, and angular velocity of the base, as well as the foot positions. The task information includes the robot's distance to the desired landing position, and the remaining time in the current locomotion cycle.

Action Space Since the low-level whole-body controller is effective for precise tracking of the body pose [10], we directly command the desired body pose in our environment. In the original work by [10], the whole-body controller takes in an 18-dimensional vector specifying the position, velocity, and acceleration for each of the base's 6 DoFs. To reduce the search dimension, we design the action space to specify one command for each DoF of the base, and computes the other two dimensions of each DoF heuristically. More specifically, the action space specifies the desired linear *acceleration* (3-dimensional) and angular *acceleration* around the z axis (1-dimensional), so that the policy can directly control the planar and vertical movements of the body, as well as turning, which can change rapidly within each jump. The action space specifies the desired *position* for the remaining two DoFs, namely the body roll and pitch, to avoid unnecessary body oscillations. The remaining commands for each DoF is specified heuristically. See section. 4.2.2 for more details.

Reward We design the reward function as the linear combination of alive bonus, distance penalty, orientation penalty and contact consistency penalty:

$$r = r_{\text{alive}} + w_p \cdot r_{\text{position}} + w_o \cdot r_{\text{orientation}} + w_c \cdot r_{\text{contact}} \tag{4.2}$$

where the alive bonus, $r_{\text{alive}} = 4$, is a fixed constant that makes the total return positive. The position reward, r_{position} is the squared distance between the robot's current position and the desired landing position, normalized by the total desired jumping distance. The orientation reward, $r_{\text{orientation}} = -(\text{roll}^2 + \text{pitch}^2)$ encourages the robot to stay upright. The contact consistency reward $r_{\text{contact}} = \sum_{i=1}^{4} \mathbb{1}(c_i \neq \hat{c}_i)$ is the sum of 4 indicator functions about the contact situation of each leg, where $\hat{c}_i \in \{0, 1\}$ is the desired contact of foot i according to contact scheduler (Section. 4.2.1) and $c_i \in \{0, 1\}$ is the actual contact of foot i. Intuitively, the contact consistency reward ensures that the robot jumps according to the desired schedule without significant mismatch in lift-off and touch-downs. We use $w_p = 1, w_o = 5, w_c = 0.4$ in all our experiments.

Early Termination To prevent the learning algorithm from unnecessary explorations in suboptimal regions, we terminate an episode early if the robot falls (i.e. when the base height is lower than 8cm, when the cosine distance between body's upright vector and the gravity direction is less than 0.6, or when any body parts came in contact with the ground).

Policy Training To improve the performance of the acceleration controller, we train a policy to add an action residual to the acceleration controller's outputs. We represent our policy as a neural network with 1 hidden layer of 256 units and tanh activation. We train our policy using Augmented Random Search (ARS) [44], a simple, parallelizable evolutionary algorithm that has been successfully applied to locomotion tasks [20, 66, 67, 68]. We chose ARS because it explores in the policy parameter space and does not directly inject noise in action outputs. Moreover, ARS evaluates policies require accurate estimation of the value function, which can be challenging for hierarchical tasks [40] due to its non-Markovian nature.

4.3 Results

4.3.1 Experiment Setup

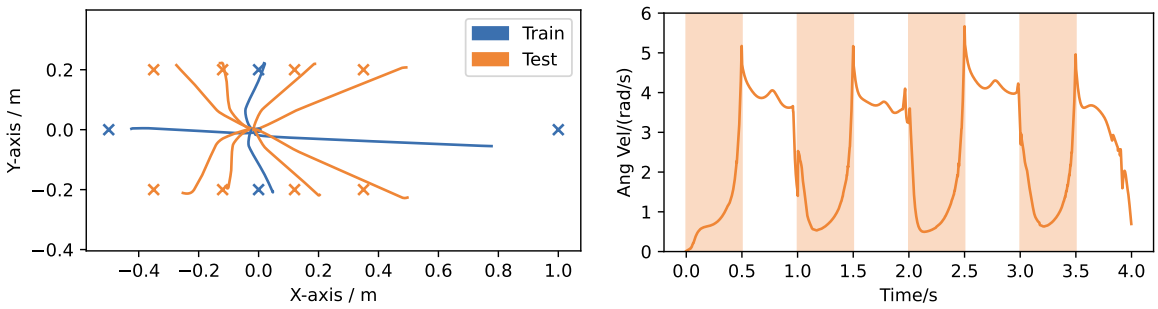
We test our framework on a Go1 robot from Unitree [32], which is a small-scale, 15kg quadrupedal robot with 12 degrees of freedom. We build the simulation environment of Go1 using Pybullet [71], and implement the entire control pipeline, including the the state estimator, low-level WBC, the jump controller and the residual policy in Python. We train the residual policy on a desktop computer with a 16-core CPU, where the training takes around 3 hours to complete.

4.3.2 Continuous, Versatile Jumping

We test the performance of our learned framework on a series of jumping tasks, where each task specifies either a different desired jumping direction, or a desired turning rate (Fig. 4.3). Although we train the framework in only 4 jumping directions, the resulting controller interpolates between them and jumps in intermediate directions (fig. 4.3a). The framework also enables the robot to jump and turn *simultaneously*, and achieves an average turning rate of about 3.5 rad/s (fig. 4.3b). For omni-directional jumping, we also notice that the controller tends to overshoot for forward jumps, and undershoot for backward jumps. This is likely due to the asymmetric leg designs of the robot platform, which generates higher accelerations in the forward direction than in the backward direction.

4.3.3 Transfer to the real robot

We deploy the learned residual policy directly to the real world without additional finetuning. Thanks to the robustness of the low-level WBC, our framework can complete several high jumps in the real world, including jumping in multiple directions and turning (Fig. 4.4). Please visit the website for videos. The robot achieves a maximum jumping height of around 50cm, a forward jumping distance of around 60cm, and a maximum turning rate of 90 degrees per jump. Note that this jumping performance in the real world is slightly lower than the



(a) Omni-directional jumping: CoM trajectory (b) Jump-turn: z-component of angular velocity

Figure 4.3: Different jumping skills learned by our framework. **Left**: Omni-directional jumping. Lines show birds-eye view of CoM trajectory, where the robot starts facing positive x direction. Crosses show desired landing positions. Colors show whether the direction is seen during training. **Right**: Continuous jump-turn. Plot shows the z-component of angular velocity (approximately the change rate of yaw angle). Shaded area indicates foot contact.

robot's performance in simulation. We hypothesize this as a result of unmodeled motor saturation, and plan to investigate further in future works.

4.3.4 Comparison with baseline policies

To further validate our design choices, we conduct an ablation study by removing either the residual policy or the acceleration controller from our pipeline. We also compare our framework with an end-to-end RL policy that directly outputs motor position commands. The result is summarized in figure. 4.5.

Acceleration Controller Only We test the performance of the manually designed acceleration controller without learned residual policy on the same jumping task. While the controller completes all the jumps without falling over, it achieves a lower reward without the residual policy (Fig. 4.5). We also find that the residual policy increases the overall success

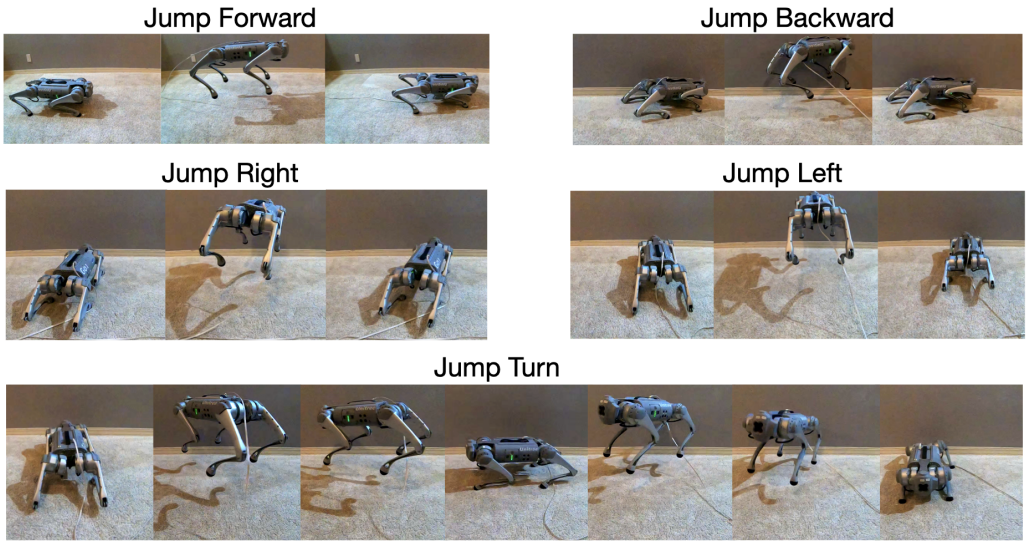


Figure 4.4: Footages of the Go1 robot completing several jumps in the real world.

rate of the robot in the real world (Table. 4.1). To further understand the effectiveness of the residual policy, we perform a backward jump with and without the residual policy, and plot the base pitch angle in Fig. 4.6. While the acceleration controller maintains the body pitch closer to reference in the stance phase, it causes significantly larger pitch angle deviations in the swing phase. This is because the acceleration controller approximates the robot as a point-mass, and cannot account for changes in body orientations. In contrast, the residual policy learns to correct this pitch angle deviation by slightly shifting the body pitch in stance phase, which results in lower overall pitch deviations throughout the entire jump.

Policy Only Without the acceleration controller, the residual policy gets stuck in a local minima, and achieves a low reward (Fig. 4.5). The resulting policy does not achieve sufficient height for each jump, and keeps legs in contact most of the times (Fig. 4.7a). We hypothesize this as a result of the frequent contact changes in the environment, which can create a noisy reward landscape for the algorithm to optimize. In contrast, the policy trained with the controller achieves consistent, high flight times for each jump (Fig. 4.7a).

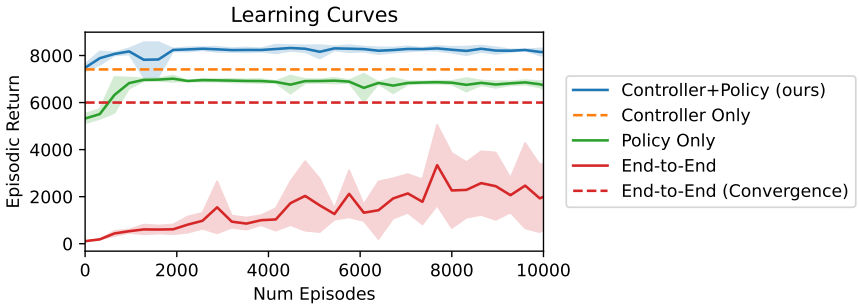


Figure 4.5: Learning curves of our framework compared with baseline methods. Error bar shows 1 standard deviation.

End-to-End RL Lastly, we compare our method to a baseline, where we train an end-to-end reinforcement learning policy that directly outputs motor position commands. Similar to previous works [8, 94, 7], we reduce the control frequency to 50Hz to avoid high-frequency motor oscillations. We use the same state space and reward function in the environment design, and train the policy using the same ARS algorithm. We find that the E2E policy learns significantly slower, and requires around 10 times more training episodes (Fig. 4.5). The policy also achieves the lowest overall reward compared to the other policies. While additional efforts in reward shaping and imitation learning [86] can improve the performance of the E2E policy, our method creates a simple, data-efficient alternative that does not require pre-recorded demonstration trajectories.

4.4 Related Works

Learning Agile Locomotion Recently, researchers have made significant progress in applying reinforcement learning (RL) for quadrupedal locomotion. Using reinforcement learning, the legged robots can adapt to the environment [8, 94] and learn diverse skills such as self-righting [49, 97], high-speed walking [53], goal-keeping [98]. However, for successful real-robot deployment, RL-based controllers usually require significant effort in imitation learning[86], reward shaping [8] and sim-to-real [7, 49, 47], especially for more agile tasks such as jumping. Compared to the end-to-end RL approaches, we re-design the RL task to

Task	Controller + Policy	Controller Only
Forward	100%	20%
Backward	80%	0%
Left	100%	80%
Right	100%	60%
Jump-Turn	100%	0%

Table 4.1: Success rates (over 5 trials) with or without the residual policy. A jump is successful if it does not trigger the early termination condition (Sec. 4.2.4).

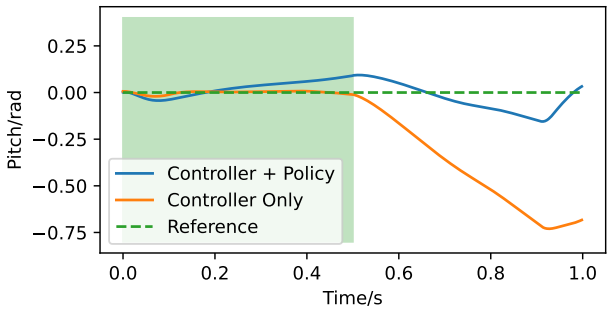
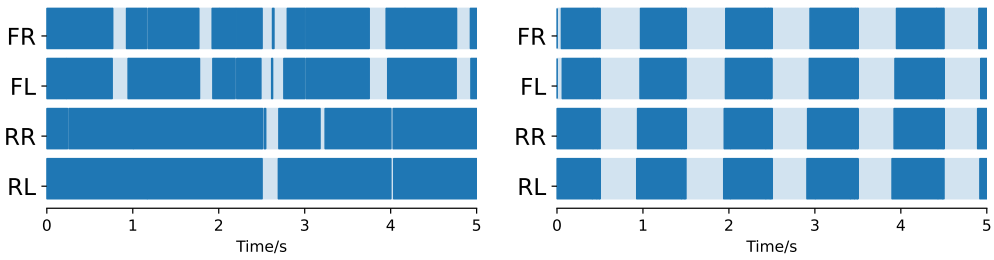


Figure 4.6: Robot pitch angle during a backward jump using different controllers. Shaded area shows stance phase.

learn the residual action [99] that adds to a manually-designed acceleration controller. As a result, the training process consumes significantly fewer data and does not require complex reward specification. More over, the learned policy can be deployed directly to the real world without additional fine-tuning.

Optimal Control for Locomotion With recent advancements in actuator design and numerical optimization, optimal-control based controllers have enabled high-speed and robust locomotion [3, 10, 77] for legged robots. For computation efficiency, these controllers usually simplify the robot dynamics model, such as the single rigid body model with massless legs [3, 95, 100], so that the optimization can run in real-time. However, these simplified models usually cannot capture the robot's orientation dynamics accurately, especially when the robot is in the air. Therefore, it can be difficult to use them for jumping tasks with long periods of air time. To optimize for more precise jumps, controllers usually need to precompute the entire jumping trajectory offline using higher-fidelity models [96, 13]. Compared to these approaches, our method does not require offline optimization, jumps continuously,



(a) Policy Only. (b) Controller + Policy (ours).

Figure 4.7: Foot contacts for controllers trained without and with the jump controller. Dark area indicates foot contact.

and can respond to different landing positions and orientations.

Combining control and learning for legged locomotion Recently, a number of works have proposed to use reinforcement learning and optimal control jointly for robust and versatile locomotion. A common approach is to set up the controller hierarchically, where a high-level policy outputs intermediate commands, which are converted by a low-level controller into motor actions. While such hierarchical approaches have enabled the robot to walk more efficiently [40, 91] and conquer difficult terrains [101], they are not yet demonstrated on more agile tasks such as jumping. We use a similar hierarchical setup, but uses a different whole-body controller in the low-level for precise tracking of body acceleration, and achieves continuous, versatile jumping on the real robot. Another common approach is to use RL to finetune the controller's outputs. Recently, [102] demonstrates that deep RL can improve the quality of a single jump in simulation. Our work extends their result by using RL to optimize for the controller's performance in the real world, and achieves continuous, omni-directional jumping on the real robot, as well as jump turns.

4.5 Discussion

In this work, we present a hierarchical framework to for continuous quadruped jumping, which consists of a manually designed acceleration controller, a learned residual policy, and a low-level whole-body controller. The trained framework can be transferred directly to the real world, and is capable of continuous jumping at arbitrary angle and distances according to user specification, as well as jump-turning. One limitation of our work is that it currently only supports the pronking gait, where all 4 legs leave or touch the ground at the same time. Supporting more versatile gaits such as bounding or galloping can potentially increase the height and distance of the jump, which we plan to investigate in future work. Another future direction is to integrate perception into our framework, so that the robot can use its jumping skills to traverse through difficult terrains autonomously.

Chapter 5

CONTINUOUS ADAPTIVE JUMPING WITH MASSIVELY-PARALLEL RL

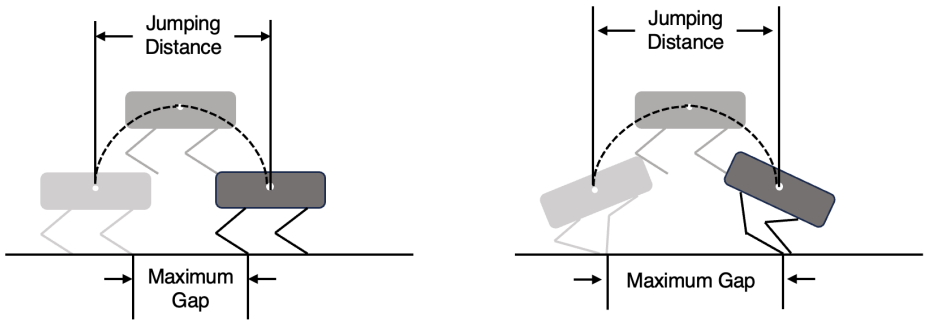


Figure 5.1: Compared to the pronking gait (left), the bounding gait can jump over wider terrain discontinuities under the same body movement.

While the work proposed in the previous section enables continuous jumping in multiple directions, the robot's behavior is limited to the *pronking* gait with synchronized foot contacts. This contact pattern can be inefficient for long-distance traversal, as the length of the robot body needs to be subtracted from the robot's total jumping distance (Fig. 5.1). In this work, we redesign the high-level policy to learn *general* jumping gaits, including bounding, so that the robot can jump continuously over long distances.

5.1 Introduction

Legged robots possess a unique capability to navigate some of the earth's most challenging terrains. By strategically adjusting their foot placement and base pose, legged robots can negotiate steep slopes [103, 104, 105], traverse uneven surfaces [17, 106], and crawl through

tight spaces [107]. However, for terrains with scarce contact choices, such as gaps or stepping stones, the capability of legged robots remains somewhat limited. This limitation primarily stems from the fact that most legged robots rely heavily on walking gaits with continuous foot contacts. As such, options for foot placement are confined to within one body length from the robot's current location. Jumping offers a compelling solution to this problem. By enabling "air phases", a jumping robot can traverse through long distances without terrain contacts. Such a capability could markedly enhance a legged robot's versatility when dealing with challenging terrains. In addition, a robot capable of *continuous*, *adaptive* and *long-distance* jumps could further boost its speed and efficiency during terrain traversal.

Compared with standard walking, jumping is a significantly more challenging control task for both optimization-based [72, 3, 10, 108, 109, 107, 14, 110, 13] and learning-based controllers [7, 106, 19, 111, 112]. Optimization-based controllers, despite proving robust in challenging terrains, face computational limitations that prevent them from planning for long jumping trajectories in real time. Typically, these controllers circumvent this issue by first solving an intricate trajectory optimization problem offline, then utilizing simplified model predictive control (MPC) to track this predetermined *fixed* trajectory online. Consequently, existing works tend to be restricted to non-adaptive, single jumps [107, 14, 110, 13]. On the other hand, RL controllers have the potential to learn more adaptive and versatile locomotion skills, but they require substantial effort in reward design and sim-to-real transfer [113, 7, 8, 17], particularly for dynamic and underactuated tasks such as jumping. Therefore, achieving continuous jumping over long distances can be a significant challenge for existing methods.

In this paper, we present CAJun (Continuous Adaptive Jumping with a Learned Centroidal Policy), which achieves continuous long-distance jumpings with adaptive distances on the real robot. Our framework seamlessly combines optimization-based control and RL in a hierarchical manner. Specifically, a high-level RL-based *centroidal policy* specifies the desired gait, target base velocity, and swing foot positions to the *leg controller*, and a low-level *leg controller* solves the optimal motor commands given the centroidal policy's action. Our framework effectively integrates the benefits of both control and learning. First, the

RL-based *centroidal policy* is able to learn versatile, adaptive jumping behaviors without heavy computational burden. Second, the low-level quadratic-programming-based (QP) *leg controller* optimizes torque commands at high frequency (500Hz), which ensures reactive feedback to environmental perturbations and significantly reduces the sim-to-real gap. Finally, to resolve the common training speed bottleneck in hierarchical methods [101, 40, 114], we reformulated the QP problem in the *leg controller* to a least-squares problem with clipping so that the entire stack is 10 times faster and can be executed in massive parallel [19].

Within 20 mins of training in simulation, we deploy CAJun directly to a Unitree Go1 robot [32]. Without any fine-tuning, CAJun achieves continuous, long-distance jumping, and adapts its jumping distance based on user command. Moreover, using the alternating contact pattern in a bounding gait, the robot is capable of crossing a gap of 70cm, which is at least 40% larger than existing methods (Fig. 5.5 and Table 5.2). To the best of our knowledge, CAJun is the first framework that achieves continuous, adaptive jumping with such gap-crossing capability on a commercially available quadrupedal robot. We further conduct ablation studies to validate essential design choices. In summary, our contribution with CAJun are the following:

- We present CAJun, a hierarchical learning and control framework for continuous, adaptive, long-distance jumpings on legged robots.
- We demonstrate that jumping policies trained with CAJun can be directly transferred to the real world with a gap-crossing capability of 70cm.
- We show that CAJun can be trained efficiently in less than 20 minutes using a single GPU.

5.2 Method Overview

In order to learn continuous, long-distance, and adaptive jumping behaviors, we design CAJun as a hierarchical framework consisting of a high-level centroidal policy and a low-level leg controller (Fig. 5.2). To specify a jump, The *centroidal policy* outputs three key

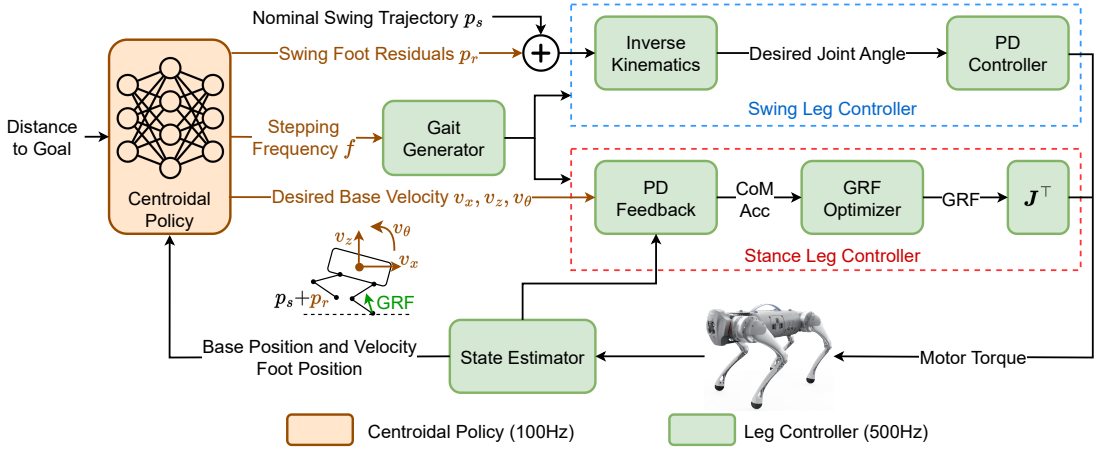


Figure 5.2: Overview of the hierarchical framework of CAJun.

actions to the low-level controller, namely, the stepping frequency, the swing foot residual, and the desired base velocity. The modules in the *leg controller* then convert these actions into motor commands. Similar to previous works [3, 10, 91, 40, 101], the *leg controller* adopts separate control strategy for swing and stance legs, where the desired contact state of each leg is determined by the *gait generator*. We design the gait generator to follow a pre-determined contact sequence with timings adjustable by the high-level centroidal policy. For swing legs, we first find its desired position based on a heuristically-determined reference trajectory and learned residuals, and converts that to joint position commands using inverse kinematics. For stance legs, we first determine the desired base acceleration from the policy commands, and then solves an optimization problem to find the corresponding Ground Reaction Forces (GRFs) to reach this acceleration. We run the low-level controller at 500Hz for fast, reactive torque control, and the high-level controller at 100Hz to ensure stable policy training.

5.3 *Reparameterized Low-level Controller for Massively-Parallel GPU Simulation*

The gait generator determines the desired contact state of each leg (swing or stance) based on a pre-defined contact sequence and the timing information from the centroidal policy.

Figure 5.3: The contact sequence and default timing of the pronking (**left**) and bounding (**right**) gait.

To capture the cyclic nature of locomotion, we adopt a phase-based gait representation, similar to prior works [40, 20]. The gait is modulated by a phase variable ϕ, which increases monotonically from 0 to 2π in each locomotion cycle, and wraps back to 0 to start the next cycle. The propagation of ϕ is controlled by the *stepping frequency* f, which is commanded by the centroidal policy:

$$\phi_{t+1} = \phi_t + 2\pi f \Delta t \tag{5.1}$$

where Δt is the control timestep. The mapping from ϕ to the desired contact state is pre-defined. We adopt two types of jumping gaits in this work, namely, *bounding* and *pronking*, where bounding alternates between the front and rear leg contacts, and pronking lands and lifts all legs at the same time (Fig. 5.3). Note that while the *sequence* of contacts is fixed, the centroidal policy can flexibly adjust the *timing* of contacts to based on the state of the robot.

5.3.1 Stance Leg Control

The stance leg controller computes the desired joint torque given the velocity command from the centroidal policy. Since jumping is mostly restricted to the sagittal plane, the policy specifies the velocity in the forward and upward axis (v_x, v_z), as well as the rotational velocity v_θ, and the velocity for the 3 remaining DoF is set to 0. We compute the desired torque following a 3-step procedure. First, we compute the desired CoM acceleration $\ddot{\boldsymbol{q}}_f^{\text{ref}} \in \mathbb{R}^6$ using a PD controller . Next, we optimize for the GRF $\boldsymbol{f} = [\boldsymbol{f}_1, \boldsymbol{f}_2, \boldsymbol{f}_3, \boldsymbol{f}_4] \in \mathbb{R}^{12}$ to track this desired acceleration, where $\boldsymbol{f}_i$ is the foot force vector of leg i. Lastly, we compute the

motor torque command using $\boldsymbol{\tau} = \boldsymbol{J}^\top \boldsymbol{f}$, where $\boldsymbol{J}$ is the foot Jacobian. When training a hierarchical controller with a low-level optimization-based controller, a major computation bottleneck lies in the GRF optimization [101, 40, 114]. As such, we re-design this optimization procedure to significantly speed up the training process.

QP-based GRF Optimization To optimize for GRF, prior works typically solve the following quadratic program (QP):

$$\min_{\boldsymbol{f}} \ \|\ddot{\boldsymbol{q}} - \ddot{\boldsymbol{q}}_f^{\text{ref}}\|_{\mathbf{U}} + \|\boldsymbol{f}\|_{\mathbf{V}} \tag{5.2}$$

$$\text{subject to: } \ddot{\boldsymbol{q}} = \mathbf{A}\boldsymbol{f} + \boldsymbol{g} \tag{5.3}$$

$$f_{i,z} = 0 \qquad \text{if } i \text{ is a swing leg} \tag{5.4}$$

$$f_{\min} \le f_{i,z} \le f_{\max} \qquad \text{if } i \text{ is a stance leg} \tag{5.5}$$

$$-\mu f_{i,z} \le f_{i,x} \le \mu f_{i,z}, \quad -\mu f_{i,z} \le f_{i,y} \le \mu f_{i,z} \qquad i = 1, \dots, 4 \tag{5.6}$$

Eq. (5.3) represents the centroidal dynamics model [3], where $\mathbf{A}$ is the generalized time-variant inverse inertia matrix, and $\boldsymbol{g}$ is the gravity vector. Eq. (5.5), (5.4) specifies the contact schedule, as computed by the gait generator. Eq. (5.6) specifies the approximated friction cone constraints, where μ is the friction coefficient. $\mathbf{U}, \mathbf{V} \succ 0$ are positive definite weight matrices.

Unconstrained GRF Optimization with Clipping QP-based GRF optimization would require an iterative procedure (e.g., active set method or interior point method), which can be computationally expensive and difficult to scale up in parallel in GPU. Instead of using the QP formulation, CAJun relaxes this optimization problem by solving the unconstrained GRF first and clipping the resulting GRF to be within the friction cone. Since Eq. (5.3) is linear in $\boldsymbol{f}$, if we ignore the constraints in Eq. (5.6) and (5.5) and eliminate the variables for non-contact legs, the optimal $\boldsymbol{f}$ can be solved *in closed-form*:

$$\widehat{\boldsymbol{f}} = (\mathbf{A}^\top \mathbf{U} \mathbf{A} + \mathbf{V})^{-1} \mathbf{A}^\top \mathbf{U} (\ddot{\boldsymbol{q}}_f^{\text{ref}} - \boldsymbol{g}) \tag{5.7}$$

Next, we project the solved ground reaction forces into the friction cone, where we first clip the normal force within actuator limits, and then clip the tangential forces based on the clipped normal force:

$$f_{i,z} = \text{clip}(\widehat{f}_{i,z}, f_{\min}, f_{\max}), \quad (f_{i,x}, f_{i,y}) = (\widehat{f}_{i,x}, \widehat{f}_{i,y}) \cdot \min(1, \mu f_{i,z} / \sqrt{\widehat{f}_{i,x}^2 + \widehat{f}_{i,y}^2}) \tag{5.8}$$

We design this projection to minimize the force disruption in the gravitational direction, so that the low-level controller can track height commands accurately.

Note that our unconstrained formulation not only reduces computational complexity, but also makes the solving procedure highly parallelizable. Therefore, when paired with GPU-accelerated simulator like Isaac Gym [19], CAJun can be trained efficiently in massive parallel, which significantly reduced the turn-around time. Additionally, while our unconstrained formulation may yield sub-optimal solutions when the least-squares solution (Eq. (5.7)) finds a GRF outside the friction cone, the high-level *centroidal policy* would observe this sub-optimality during training, and thereby compensating for it by adjusting the desired CoM velocity commands. In practice, we find the policies trained using the constrained and unconstrained optimization to perform similarly (see Sec. 5.5.6 for details).

5.3.2 Swing Leg Control

We use position control for swing legs, where the desired position is the sum of a heuristically constructed reference trajectory [3, 27] and a residual output from the centroidal policy. Similar to prior works [40, 101], we generate the reference trajectory by interpolating between key points in the swing phase. On top of the heuristic trajectory ($\boldsymbol{p}_s$ in Fig. 5.2), the centroidal policy adjusts the swing foot trajectory for higher foot clearance and optimal foot placement by outputting a residual in foot position ($\boldsymbol{p}_r$ in Fig. 5.2). Once the foot position is determined, we convert it to desired motor angles using inverse kinematics and execute it using joint PD commands.

5.4 Learning A Centroidal Policy for Jumping

Environment Overview For maximum expressiveness, we design the environment such that the centroidal policy directly specifies the contact schedule, base velocity and swing foot position for the low-level controller. To focus on continuous jumps, we design each episode to contain exactly 10 jumping cycles, where termination is determined by the gait generator (Section. 5.3). Additionally, we normalize the reward so that total reward within each jumping cycle is agnostic to its duration. In order to learn distance-adaptive jumping, we sample different jumping distances uniformly in [0.3m, 1m] before each jump, and compute the desired landing position, which is included in the state space.

State and Action Space We design the state space to include the robot's proprioceptive state, as well as related information about the current jump. The proprioceptive information includes the current position and velocity of the robot base, as well as the foot positions in the base frame. The task information includes the current phase of the jump ϕ (Sec. 5.3) and the location of the target landing position in egocentric frame. The action space includes the desired stepping frequency f, the desired base velocity in sagittal plane v_x, v_z, v_θ, as well as the desired swing foot residuals, which are specified to different modules in the low-level controller.

Reward Function We design a reward function with 9 terms. At a high level, the reward function ensures that the robot maintains an upright pose, follows the desired contact schedule, and lands close to goal. We provide the detail about each term and its corresponding weight below:

1. **Upright (0.02)** is the projection of a unit vector in the z-axis of the robot frame onto the z-axis of the world frame, and rewards the robot for keeping an upright pose.

2. **Base Height (0.01)** is the height of the robot's CoM in meters, and rewards the robot for jumping higher.

3. **Contact Consistency (0.008)** is the sum of 4 indicator variables: $\sum_{i=1}^{4} \mathbb{1}(c_i = \hat{c}_i)$, where c_i is the actual contact state of leg i, and $\hat{c}_i$ is the desired contact state of leg i specified by the gait generator. It rewards the robot for following the desired contact schedule.

4. **Foot Slipping (0.032)** is the sum of the world-frame velocity for contact-legs: $\sum_{i=1}^{4} \hat{c}_i \sqrt{v_{i,x}^2 + v_{i,y}^2}$, where $\hat{c}_i \in \{0, 1\}$ is the desired contact state of leg i, and $v_{i,x}, v_{i,y}$ is the *world-frame* velocity of leg i. This term rewards the robot for keeping contact legs static on the ground.

5. **Foot Clearance (0.008)** is the sum of foot height (clipped at 2cm) for non-contact legs. This term rewards the robot to keep non-contact legs high on the ground.

6. **Knee Contact (0.064)** is the sum of knee contact variables $\sum_{i=1}^{4} kc_i$, where $kc_i \in \{0, 1\}$ is the indicator variable for knee contact of the ith leg.

7. **Stepping Frequency (0.008)** is a constant plus the negated frequency $1.5 - \text{clip}(f, 1.5, 4)$, which encourages the robot to jump at large steps using a low stepping frequency.

8. **Distance to goal (0.016)** is the Cartesian distance from the robot's current location to the desired landing position, and encouarges the robot to jump close to the goal.

9. **Out-of-bound-action (0.01)** is the normalized amount of excess when the policy computes an action that is outside the action space. We design this term so that PPO would not excessively explore out-of-bound actions.

Early Termination To speed up training and avoid unnecessary exploration in sub-optimal states, we terminate an episode early if the robot's base height is less than 15cm, or the base orientation deviates significantly from the upright pose.

Policy Representation and Training We represent policy and value functions using separate neural networks. Each network includes 3 hidden layers of $[512, 256, 128]$ units respectively with ELU activations [115]. We train our policy using Proximal Policy Optimization (PPO) [43]. Please see Table. 5.1 for the detailed configuration.

Parameter	Value
Learning rate	0.001, adaptive
# env steps per update	98,304
Batch size	24,576
# epochs per update	5
Discount factor	0.99
GAE λ	0.95
Clip range	0.2

Table 5.1: Hyperparameters used for PPO.

5.5 Results and Analysis

5.5.1 Experiment Setup

We use the Go1 quadrupedal robot from Unitree [32], and build the simulation in IsaacGym [19, 46]. To match the GPU-accelerated simulation environment, we implement the entire control stack, including the centroidal policy and the leg controller, in a vectorized form in PyTorch [116]. We adopt the PPO implementation from `rsl_rl` [19]. We train CAJun on a standard desktop with an Nvidia RTX 2080Ti GPU, which takes less than 20 minutes to complete.

5.5.2 Continuous and Adaptive Jumping

To verify that CAJun can learn continuous, dynamic jumping with adaptive jumping distances on the real robot, we deploy the trained *pronking* and *bounding* controllers to the real robot. For each gait, we run it continuously for at least 6 jumps, where the desired jumping distance alternates between 0.3 and 1 meter. We put LEDs on the base and feet of the robot and capture the robot's trajectory using long-exposure photography (Fig. 5.4).

We find that both the pronking and the bounding controller can be deployed successfully

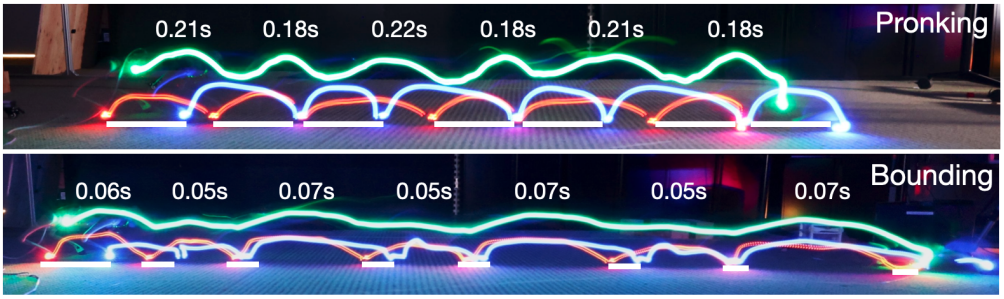


Figure 5.4: Long-exposure photos visualizing base (green), front foot (blue) and rear foot (red) trajectories of the robot when jumping with alternating distance commands. White lines show the foot positions during each landing (contact phase for pronking, mid-air phase for bounding). Time shows the duration of "air phase" (Fig. 5.3) in each jump when all legs are in the air.

Figure 5.5: Using the bounding gait, the robot can jump over a 60cm-wide yoga mat without making foot contact.

to the real robot, and achieve continuous jumping with long jumping distances. Both the base and the foot trajectories exhibit clear periodicity, which demonstrates the long-term stability of the jumping controller. Moreover, the policy responds to jumping distance commands well, and results in alternating patterns of further and closer jumps. A closer look at the duration of each air phase shows that in both the bounding and pronking gait, the centroidal policy reduces the air time by approximately 20% when switching from longer to shorter jumps. This is achieved by the stepping frequency output (Section. 5.3) of the centroidal policy. As demonstrated in previous works [40, 117], such gait adjustments can potentially save energy and extend the robot's operation time.

Method	Jumping Style	Widest Gap Crossed
TWiRL [113]	Single	0.2m
Barkour [105]	Single	0.5m
Margolis et al. [118]	Continuous	0.26m
Walk-These-Ways [111]	Single w/ Acceleration	0.6m
CAJun (ours)	Continuous w/ Adaptive Jumping Distance	**0.7m**

Table 5.2: Comparison of gap-crossing capability on controllers deployed to similar-sized robots.

5.5.3 Jumping over Wide Gaps

While both the pronking and bounding gait can jump with at least 70cm of base movement in each step, we find that the bounding gait offers a unique advantage in traversing through gaps. As seen in the foot trajectories in Fig. 5.4, the alternating contact pattern in bounding enables the front and rear of the robot to land closely in the world frame, so that the robot can utilize *the entire jumping distance* of 70cm for gaps. To further validate this, we place a yoga mat with a width of 60cm in the course of the robot, and find that the robot can jump over it with additional buffer space before and after the jump (Fig. 5.5). To the best of our knowledge, CAJun is the first framework that achieves continuous jumping with such gap-crossing capability on a commercially-available quadrupedal robot (Table. 5.2).

5.5.4 Validation on Robustness

We design two experiments to further validate the robustness of CAJun. In the first experiment, we add a leash to the back of the robot and actively pulled the leash during jumping (Fig. 5.6). While both the pronking and bounding gait experienced a significant drop in forward velocity during the pull, they recovered from the pull and regained momentum for subsequent jumps. In the second experiment, we test the robot outdoors, where the robot needs to jump from asphalt to grass (Fig. 5.7). The uneven and slippery surface of the grass perturbed the robot and broke the periodic pattern in pitch angles. However, both policies recovered from the initial perturbation, and resume stable, periodic jumps after around 2

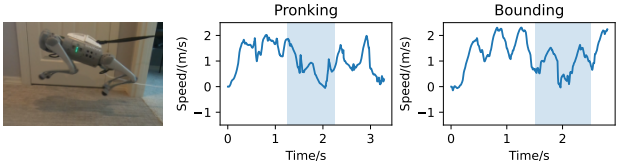


Figure 5.6: Forward velocity of the robot jumping under leash pulling (shaded area shows active pulling).

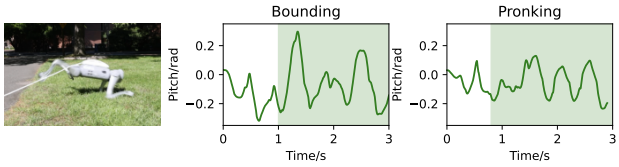


Figure 5.7: Pitch angle of the robot jumping from asphalt to grass (shaded area indicates grass)

	Pronking		**Bounding**	
	Sim	Real	Sim	Real
E2E	4.17±0.01	1.67±0.18	**4.61±0.03**	3.47±0.15
CAJun (ours)	**4.98±0.02**	**4.76±0.11**	4.27±0.05	**4.34± 0.17**

Table 5.3: Total distance after 6 jumps achieved by end-to-end RL and CAJun.

jumping cycles. The robustness of CAJun can be likely attributed to the high control frequency of the low-level leg controller, which enables the robot to react swiftly to unexpected perturbations, and the online adjustment of the learned centroidal policy.

5.5.5 Comparison with End-to-End RL

To demonstrate the effectiveness of CAJun's hierarchical setup, we compare it to an end-to-end RL baseline, where the policy directly outputs motor position commands. We use a similar MDP setup as CAJun (section. 5.4) for the end-to-end RL baseline. More specifically, we use the same gait generator as CAJun to generate reference foot contacts, and include stepping frequency as part of the action space so that the policy can modify the gait schedule. However, unlike CAJun, this reference gait is only used for reward computation, and does not directly affect leg controllers. For reward, we keep the same reward terms and weights. However, since the initial exploration phase of end-to-end RL can lead to a lot of robot failures with negative rewards, we add an additional alive bonus of 0.02 to ensure that the reward stays positive.

In both simulation and the real world, we run each policy for 6 jumps with a desired

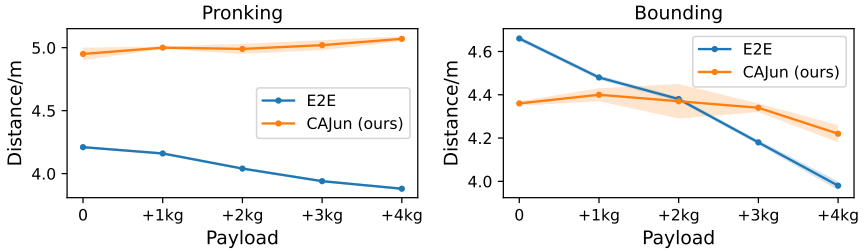


Figure 5.8: Comparison of total jumping distance under increased payload.

distance of 1 meter per jump, and report the total CoM displacement in Table. 5.3. While CAJun and end-to-end RL achieves comparable performance in simulation, CAJun faces a significantly smaller sim-to-real gap and outperforms e2e baseline for both gaits in the real world (25% further in bounding, 185% in pronking). While additional efforts such as domain randomization [7], system identification [16] or teacher-student training [8] could improve the robustness and reduce the sim-to-real gap for E2E methods, the hierarchical framework of CAJun offers a simple and efficient alternative that can be deployed zero-shot to the real world.

Sim-to-Sim Transfer To better understand the robustness of CAJun and end-to-end RL (E2E) under different dynamics, we conduct a sim-to-sim transfer experiment, where we test the performance of CAJun and E2E under increased body payloads. The result is summarized in Fig. 5.8. While the distance of E2E drops quickly with increased payload, CAJun maintains a near-constant distance even with a 4kg payload, thanks to the robustness of the low-level centroidal controller.

5.5.6 Ablation Study

We design a set of ablation studies to validate the design choices of CAJun. We summarize the results here. For each baseline, we report its total reward and CoM displacement over 6 consecutive jumps with a desired distance of 1m per jump (Fig. 5.9a). We train each baseline

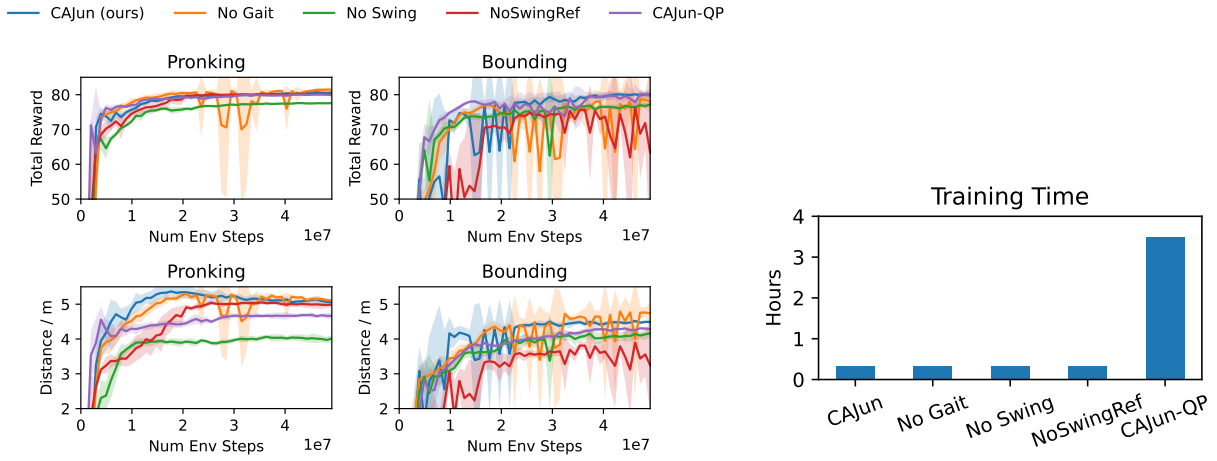
using 5 random seeds and report the average and standard deviations. We also report the wall-clock training time in Fig. 5.9b.

No Gait Modulation The stepping frequency from the centroidal policy is essential for the stability of the robot. In *no-gait*, we disable the stepping frequency output and adopt a fixed stepping frequency of 1.66Hz for both the pronking and bounding gait, which is the average stepping frequency output from CAJun. While the baseline can achieve a similar reward, the learning process is noisy with frequent failures. Since the heuristically-designed gait might not match the capability of the robot, it is important for the policy to adjust the gait timing to stabilize each jump.

No Swing Leg Residual The swing residuals play a critical role in achieving long-distance jumps. To validate that, we design a baseline, *no-swing*, where we disable the swing residuals so that swing legs completely follow the heuristically-designed trajectory from the swing controller. We find that the baseline policy cannot jump as far as CAJun, and achieves a lower reward for both gaits.

No Swing Leg Reference The reference swing leg trajectory improves the overall jumping performance. In *NoSwingRef*, we train a version of CAJun where the centroidal policy directly specify swing foot position without reference trajectory. While *NoSwingRef* performs similarly to CAJun for the pronking gait, it jumps significantly shorter and achieves a lower reward for the bounding gait, because the bounding gait requires more intricate coordination of swing legs.

CAJun-QP The clipped QP in GRF optimization significantly reduced training time without noticeable performance drops. To validate this design choice, we compare the training time and policy performance of CAJun with a variant, CAJun-QP, where we solve for GRFs using the complete QP setup, where the approximated friction cone is imposed as constraints. We adopt the QP-solver from `qpth` [119], an efficient interior-point-method-based solver that



(a) Reward Curves

(b) Training Time

Figure 5.9: Reward curve and training time of CAJun compared to the ablated methods.

supports GPU acceleration. For both the pronking and bounding gait, we find that CAJun achieves a similar reward compared to CAJun-QP. However, because CAJun-QP needs to iteratively optimize GRF at every control step, its training time is almost 10 times longer, which is consistent with prior observations [101].

5.5.7 Extension to Other Gaits

While we focus on jumping gaits in this work, CAJun is a versatile locomotion framework that is capable of learning a wide range of locomotion gaits. By adopting a different contact sequence for the gait generator (Fig.5.3), CAJun can learn a wide variety of other locomotion gaits such as crawling, pacing, trotting and fly trotting. With GPU-parallelization, all these gaits can be trained in less than 20 minutes. Please check our website for videos.

5.6 Related Works

Optimization-based Control for Jumping Using optimization-based controllers, researchers have achieved a large variety of jumping behaviors, from continuous pronking and bounding [72, 3, 10, 108, 109] to large single-step jumps [107, 14, 110, 13]. By optimizing for control inputs at a high frequency, these controllers can execute robust motions even under severe perturbations [3, 10]. However, due to the high computation cost, they cannot plan ahead for a long horizon during online execution. Therefore, they primarily focus on high-frequency jumps with a short CoM displacement per jump [10, 108, 109]. One way to overcome this computation limit is to pre-compute a reference trajectory offline using trajectory optimization (TO) [107, 14, 110, 13], which can greatly extend the height [14] and distance [110] of each jump. However, it can be challenging to generalize beyond the reference trajectories towards continuous, adaptive jumping [41, 5, 42]. Notably, using a multi-level planner, Park et al. [5] achieved continuous bounding with fixed gait and adaptive height to jump over hurdles. Compared to these approaches, our framework adopts a more general formulation, where the policy can adjust the gait timing, base pose, and swing foot position simultaneously.

Learning-based Control for Jumping In recent years, learning-based controllers have significantly improved the capability of legged robots, from rapid running [53] to traversing over challenging terrains [106]. While standard walking gaits can be learned from scratch using reinforcement learning (RL), more dynamic behaviors such as jumping usually require additional setup in the learning process, such as motion imitation [113, 112, 111], curriculum learning [19] and multi-stage training [105, 120]. Another challenge for learning-based controllers is sim-to-real, especially for dynamic underactuated behaviors like jumping [118]. To overcome the sim-to-real gap, researchers have developed a suite of tools such as domain randomization [7], system identification [16] and motor adaptation [8]. Recently, Smith et al. [113] used motion imitation and transfer learning to jump over a gap of 20cm (0.4 body

length) on a Unitree A1 robot, and Caluwaerts et al. [105] used multi-stage training with policy synthesis to jump over a gap of 50cm (1 body length) on a custom-built quadrupedal robot. Compared to these works, CAJun's hierarchical setup can jump over *wider* gaps (70cm / 1.4 body length) *continuously*, and can adapt its landing position based on user command.

Hierarchical RL for Legged Robots Recently, there has been increasing interest in combining RL with optimization-based control for legged robots [40, 91, 101, 114, 121, 102]. These frameworks typically follow a hierarchical structure, where a high-level RL-trained policy outputs intermediate commands to a low-level leg controller. The RL policy can give several forms of instructions to the low-level controller, such as gait timing [40, 91], CoM trajectory [101, 102, 118, 121] and foot landing positions [114, 122, 123, 124, 125]. Our approach uses a similar hierarchical setup but adopts a general action space design where the policy specifies the gait, CoM velocity and swing foot locations *simultaneously*. One bottleneck of the hierarchical approaches is the slow training time because every environment step involves solving the optimization problem in the low-level controller. We overcome this bottleneck by relaxing the constraints in foot force optimization [126, 127, 128], so that foot force can be solved efficiently in closed form. Compared to existing frameworks which can take hours or even days to train, CAJun can be trained in 20 minutes using GPU-accelerated simulation [19].

5.7 Discussion

In this work, we present CAJun, a hierarchical learning framework for legged robots that consists of a high-level centroidal policy and a low-level leg controller. CAJun can be trained efficiently using GPU-accelerated simulation and can achieve continuous jumps with adaptive jumping distances of up to 70cm. One limitation of CAJun is that, while it can adapt to changes in jumping distances, it can not land accurately at the desired location yet. This inaccuracy might be due to a number of factors such as unmodeled dynamics and state

estimation drifts. Another limitation of CAJun is that it does not make use of perception, and only adjusts its jumping distances based on ad-hoc user commands. In future work, we plan to extend CAJun to incorporate perception and achieve more accurate jumps, so that the robot can demonstrate extended agility and autonomy in challenging terrains.

Chapter 6

HIGH-SPEED CONTINUOUS JUMPING OVER DISCONTINUOUS TERRAINS

6.1 Introduction

Jumping can greatly extend the capabilities of legged robots. While the standard walking gait is sufficient for relatively smooth terrains, a carefully coordinated jumping gait with a long air-phase can enable a robot to cross terrains with large discontinuities, such as wide gaps and high steps, as demonstrated in recent works [52, 51, 107, 102, 41, 129, 105, 14, 130]. However, for more complex terrains with repeated discontinuities, such as stairs and stepping stones, traversing these terrains requires a sequence of carefully coordinated jumps, with precise planning in contact timing, body motion and foot placement, as well as rapid adaptation to newly perceived terrain information [41]. Because of these reasons, achieving continuous jumping over these challenging terrains with repeated discontinuities remains a central challenge in legged robots.

In this work, we present a hierarchical learning-control framework that achieves high-speed continuous jumping on real-world terrains with challenging discontinuities. Compared with single-step jumping, continuous, perceptive jumping on these challenging terrains introduces several unique challenges. Firstly, while the continuous jumping task requires precise body and foot motions, the highly dynamic nature of the task leads to an amplified sim-to-real gap [7, 49], where even small dynamics mismatches can lead to significant discrepancies in resulting trajectories. Therefore, the controller needs to be robust to handle these discrepancies effectively. Moreover, the robot is constantly underactuated throughout the entire jumping process [13, 15], making it difficult to control all aspects of body movements simultaneously. As a result, standard reinforcement learning (RL) algorithms that are unaware of

Figure 6.1: Our framework enables a quadrupedal robot to jump continuously over real-world stairs and steps.

this underactuation may learn to exploit this underactuation and learn behaviors that cannot be transferred to the real world. Lastly, continuous jumping requires rapid and accurate hand-eye coordination [51, 52, 129], where the complexity of real-time perception and motion planning can be overwhelming to learn with a single end-to-end network. Due to these challenges, most existing works focus on single-step jumps [51, 14, 102, 105] or continuous jumps on flat terrains [130, 41].

Our framework addresses all the challenges mentioned above, achieving continuous jumping on challenging terrains with repeated discontinuities for the first time. Building on top of our prior work [130], our framework consists of a heightmap predictor to process depth images, a motion policy to coordinate body and foot motions, and a low-level leg controller for motor control. We adopt a two-staged training approach [8, 18, 106, 51], where we first train the motion policy using reinforcement learning with ground-truth terrain information, and then train the heightmap predictor using supervised learing to estimate the terrain information. We redesigned key components of our framework to achieve the task of high-speed, perceptive jumping. To reduce the sim-to-real gap, we augmented the low-level feedforward optimal controller with a velocity feedback [10, 72, 11]. This increases the controller's robustness against dynamic shifts while maintaining effective long-horizon trajectory tracking. To account for the underactuated nature of the robot, we included the final cost of the low-

level optimal control problem in the reward function for policy training, and encourages the policy to learn feasible actions. To reduce the complexity of perception and control, we learn a separate heightmap predictor and use it together with the RL-trained motion policy. This modular approach allows us to inspect each component individually, from carefully identifying camera latencies to evaluating the tracking performance of the motion policy, ensuring the high performance of the resulting system.

We deploy our trained framework on a Unitree Go1 robot [32], and test it on a variety of challenging terrains in the real world (Fig. 6.1). With the continuous jumping gait, our robot jumps over a human-sized staircase with 14 steps in less than 4.6 seconds, traversing 2 stair steps in each jumping cycle. To the best of our knowledge, this is the first time any quadrupedal robot has achieved such high-speed traversal on stair cases. In addition to stair jumping, our framework also learns to jump continuously on horizontal stepping stones, and crosses one gap in each jump. Our framework also achieves state-of-the-art performance in jumping over single horizontal discontinuities (up to 80cm horizontally, 60cm vertically) [51, 52]. Thanks to the RL training, the motion policy learns versatile jumping behaviors through environment interaction, and can plan accurate jumping trajectories based on terrain perception. We further conduct an ablation study to validate important design choices.

In summary, our contributions in this paper are as follows:

1. We extended the hierarchical learning-control framework [130] for continuous, perceptive jumping on challenging discontinuous terrains.

2. We deploy the framework for real-world jumping tasks, and achieve continuous, high-speed jumping on stairs and stepping stones for the first time.

3. We validate that our framework can learn diverse jumping skills with high-performance single-step jumps.

6.2 Method Overview

6.2.1 The Hierarchical Learning-Control Framework

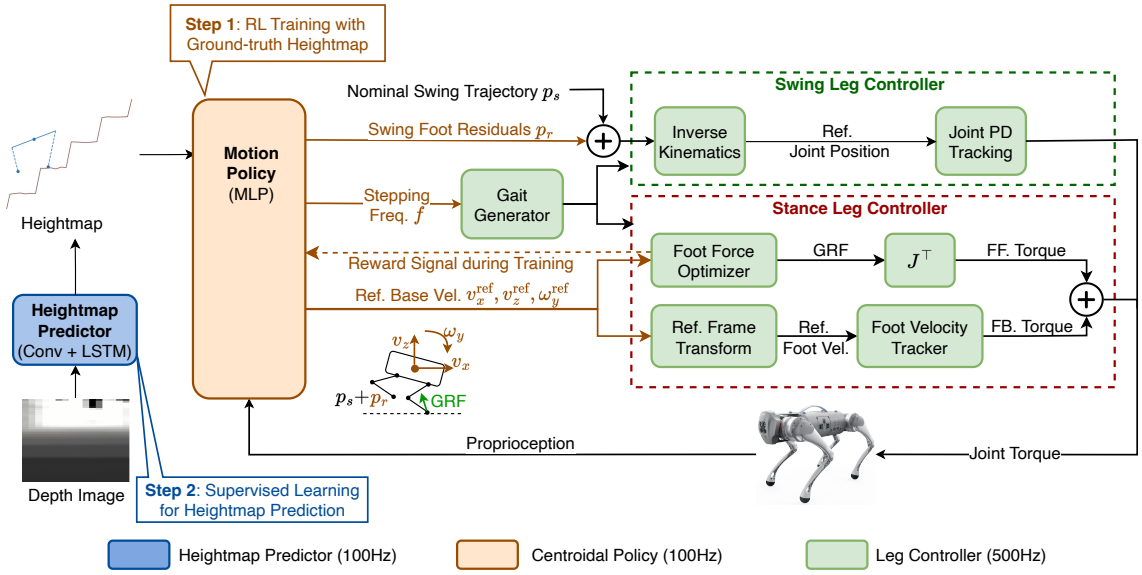


Figure 6.2: Block Diagram

In order to learn high-speed, continuous jumping motions over challenging discontinuous terrains, we design a hierarchical learning-control framework (Fig. 6.2), which includes a *Heightmap Predictor* to process perceptive information, a *Motion Policy* for high-level motion planning, and a *Leg Controller* for low-level motor control. At the perception level, the *Heightmap Predictor* takes in consecutive depth images and estimates a 1D heightmap in the front-back axis of the robot. The *Motion Policy* then uses this estimated heightmap, as well as robot proprioceptive information, to coordinate the jumping motions. This motion policy outputs three key actions to the Leg Controller, which includes the stepping frequency f, the swing foot residuals p_r, and the desired base velocity v_x, v_z, v_θ. The Leg Controller then converts these actions into motor commands, with separate control strategies for swing and stance legs [40, 130, 3, 10]. The switch between swing and stance legs is managed by a

gait generator, which maintains a fixed contact sequence but allows flexible contact timing based on the stepping frequency. f The swing leg follows a reference trajectory, which is the sum of a nominal trajectory $\boldsymbol{p}_s$ determined by Raibert Heuristics [27] and a swing residual $\boldsymbol{p}_r$ from the policy output. We design the stance leg controller to include both a feed-forward foot-force optimizer and a feed-back velocity tracker, which tracks the trajectories accurately and prepares the robot for unexpected perturbations. We run the leg controller at 500Hz for responsive torque control, and the motion policy and heightmap predictor at 100Hz for smooth policy output.

6.2.2 Two-staged Policy Training with Explicit Heightmap

Similar to prior works [8, 52, 51], we train the Motion Policy and Heightmap Predictor separately in two stages. In the first stage, we train the *Motion Policy* through reinforcement learning, and feed the policy with ground-truth heightmaps. We parallelize the entire control loop, including heightmap generation and the low-level controllers, on GPU to reduce training time. In the second stage, we train the Heightmap Predictor through imitation learning [131], where we alternate between data collection and model fitting to iteratively improve the predictor's accuracy.

6.3 Low-level Leg Controller with Feedback

Similar to prior hierarchical frameworks [130, 40, 101], the low-level *leg controller* of our framework adopts separate control strategies for swing and stance legs, where the contact state of each leg is determined by the *gait scheduler*. Each module in the leg controller is directly modulated by the actions from the *centroidal policy*.

6.3.1 Gait Generation

While multiple jumping gaits have been studied for quadrupedal robots [121, 113, 105], we focus on the *bounding gait* in our framework (Fig. 6.3), a gait that has been known for

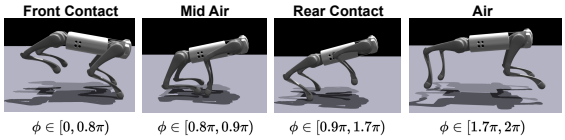


Figure 6.3: The contact sequence and default timing of the bounding gait.

long-distance jumps [113, 105]. Each bounding cycle consists of 4 contact modes, namely, the *front-contact*, *mid-air*, *rear-contact*, and *air*. The robot prepares for the jump in the first three modes, and performs long horizontal or vertical jumps in the air phase. We represent the robot's progress between contact modes with a phase variable $\phi \in [0, 2\pi)$. ϕ increases monotonically within each cycle and wraps back to 0 as the new cycle begins. The gait generator modulates the progress of each cycle by advancing the phase ϕ at each timestep:

$$\phi \leftarrow \phi + 2\pi f \Delta t \tag{6.1}$$

where f is the stepping frequency from the centroidal policy, and Δt is the control timestep.

6.3.2 Stance Leg Controller with Feedforward and Feedback

For fast and reactive response in dynamic jumping tasks, the stance foot controller needs to effectively prepare for future trajectories, while reacting dynamically to unexpected perturbations. Therefore, unlike the prior work that only includes the *feedforward* optimal controller for stance foot control, we introduce an additional *feedback* controller on top of the feedforward controller for improved robustness. Given the reference CoM velocity $v_x^{\text{ref}}, v_z^{\text{ref}}, \omega_y^{\text{ref}}$ from the centroidal policy, the stance foot controller sums over the feedforward torque τ_{ff} and the feedback torque τ_{fb} as the final joint torque command.

Feedforward Optimal Control The feedforward optimal controller first computes a reference CoM acceleration $\boldsymbol{a}^{\text{ref}}$ based on the reference CoM velocity v_x^{ref}, v_z^{ref}, v_θ^{ref}, and then optimize for ground reaction forces (GRF) to track this reference acceleration. We compute

the CoM acceleration using CoM PD Control:

$$\boldsymbol{a}^{\text{ref}} = \boldsymbol{k_p}(\boldsymbol{p}^{\text{ref}} - \boldsymbol{p}) + \boldsymbol{k_d}(\boldsymbol{v}^{\text{ref}} - \boldsymbol{v}) \tag{6.2}$$

where $\boldsymbol{p} = [p_x, p_y, p_z, \phi, \theta, \psi]$ is the body pose with Euler angle orientation representation, $\boldsymbol{v} = [v_x, v_y, v_z, \omega_x, \omega_y, \omega_z]$ is the body velocity. At the position level, we only track a reference roll angle of 0, and set the gain K_p to be 0 in other dimensions. At the velocity level, we set $\boldsymbol{v}^{\text{ref}} = [v_x^{\text{ref}}, 0, v_z^{\text{ref}}, 0, v_\theta^{\text{ref}}, 0]$ to track the commands from the motion policy.

Given the reference CoM acceleration $\boldsymbol{a}^{\text{ref}}$, we then optimize for ground reaction forces (GRFs) by solving a quadratic program (QP):

$$\min_{\boldsymbol{f}} \ \|\boldsymbol{a} - \boldsymbol{a}^{\text{ref}}\|_{\mathbf{U}} + \|\boldsymbol{f}\|_{\mathbf{V}} \tag{6.3}$$

$$\text{s.t. } \boldsymbol{a} = \mathbf{M}\boldsymbol{f} + \boldsymbol{g} \tag{6.4}$$

$$f_{i,z} = 0 \qquad \text{if } i \text{ is a swing leg} \tag{6.5}$$

$$f_{\min} \le f_{i,z} \le f_{\max} \qquad \text{if } i \text{ is a stance leg} \tag{6.6}$$

$$-\mu f_{i,z} \le f_{i,x}, f_{i,y} \le \mu f_{i,z} \qquad i = 1, \ldots, 4 \tag{6.7}$$

Eq. (6.4) represents the centroidal dynamics model [3, 130, 40], where $\mathbf{M}$ is the generalized time-variant inverse inertia matrix, and $\boldsymbol{g}$ is the gravity vector. Eq. (6.6), (6.5) specifies the contact schedule, as computed by the gait generator. Eq. (6.7) specifies the approximated friction cone constraints, where μ is the friction coefficient. $\mathbf{U}, \mathbf{V} \succ 0$ are positive definite weight matrices. To reduce training time, we compute a closed-form approximate solution of the QP during training [130]. During deployment, we compute the exact solution for improved robustness. Finally, we convert the GRFs into motor torques using $\boldsymbol{\tau}_{\text{ff}} = \mathbf{J}^\top \boldsymbol{f}$, where $\mathbf{J}$ is the foot jacobian.

Feedback Velocity Tracking Control While the feedforward (FF) controller considers robot dynamics and is effective for long-horizon trajectory tracking, it may struggle with unexpected dynamics shifts and perturbations. To ensure robust and precise motion control, we introduce an additional feedback (FB) controller to provide real-time corrections [10, 11]

to the feedforward controller. The feedback controller tracks the desired foot *velocities* based on the base reference velocities v_x^{ref}, v_z^{ref}, v_θ^{ref}.

Assuming static foot contacts, we can compute the reference foot velocities from these reference CoM velocities:

$$\boldsymbol{v}_{\text{foot}}^{\text{ref}} = -\boldsymbol{v}_{\text{CoM}}^{\text{ref}} - \boldsymbol{\omega}_{\text{CoM}}^{\text{ref}} \times \boldsymbol{p}_{\text{foot}} \tag{6.8}$$

where $\boldsymbol{v}_{\text{CoM}}^{\text{ref}}, \boldsymbol{\omega}_{\text{CoM}}^{\text{ref}}$ is the reference CoM velocity, and and $\boldsymbol{p}_{\text{foot}}$ is the position of the foot in body frame.

We can then convert the reference foot velocities to the reference joint velocities via Jacobian inverse:

$$\dot{\boldsymbol{q}}^{\text{ref}} = \mathbf{J}^{-1}\boldsymbol{v}_{\text{foot}}^{\text{ref}} \tag{6.9}$$

Finally, we compute the feedback torques based on the refernce joint velocities:

$$\boldsymbol{\tau}_{\text{fb}} = \boldsymbol{k_d}(\dot{\boldsymbol{q}}^{\text{ref}} - \dot{\boldsymbol{q}}) \tag{6.10}$$

We set $\boldsymbol{k}_d = 1$ to provide effective feedback while not causing significant interference with the feedforward controller.

6.3.3 Swing Foot Control

Similar to prior work [130], we use position control for swing legs, where the desired foot position is the sum of a reference position $\boldsymbol{p}_s$ computed by the Raibert Heuristic [27] and a residual $\boldsymbol{p}_r$ from the Centroidal Policy. Given the reference position, we then convert this task-space reference position into joint space using inverse kinematics (IK), and tracks the desired joint positions using joint-level PD control.

6.4 Two-staged Training of Perception and Motion Planning

To speed up training and improve the interpretability of our framework, we separate the high-level policy into a heightmap predictor for heightmap estimation and a motion policy for motion coordinaton. We first train the motion policy using reinforcement learning (RL), and

Component	Dimensions
Gait phase	2
Base Height (p_z, ϕ, θ)	3
Base Velocity $(v_x, v_y, v_z, \omega_x, \omega_y)$	5
Foot Position	12
1D Heightmap (-0.4m-0.8m)	30

Table 6.1: The observation space of Centroidal Policy.

then train the motion policy using supervised learning, where the trajectories are collected using the trained motion policy.

6.4.1 Low-level Aware Reinforcement Learning

We train the motion policy with reinforcement learning (RL). The RL problem is represented as a Markov Decision Process (MDP), which includes the state space $\mathcal{S}$, action space $\mathcal{A}$, transition probability $p(s_{t+1}|s_t, a_t)$, reward function $r : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$, and initial state distribution $p_0(s_0)$. We aim to learn a policy $\pi : \mathcal{S} \mapsto \mathcal{A}$ that maximizes the expected cumulative reward over an episode of length T, which is defined as $J(\pi) = \mathbb{E}_{s_0 \sim p_0(\cdot), s_{t+1} \sim p(\cdot|s_t, \pi(s_t))} \sum_{t=0}^{T} r(s_t, a_t)$.

Environment Overview Similar to prior work [130], we design the environment to learn continuous, terrain-adaptive jumps. Each episode starts with the robot in the front-contact mode of the bounding cycle (Fig. 6.3), and terminates when the robot has finished exactly 10 cycles, or when the robot has fallen. Since each episode can be of variable length depending on the stepping frequency, we additionally normalize the reward by the stepping frequency to reduce the bias on low-frequency gaits.

State and Action Space As detailed in Table. 6.1, the observation space of the centroidal policy includes the phase of the current gait cycle, robot proprioceptive information, and an 1-D heightmap. The 1-D heightmap samples the terrain uniformly along the x

Component	Dimensions
stepping frequency (f)	1
desired CoM velocity ($v_x^{\text{ref}}, v_z^{\text{ref}}, \omega_y^{\text{ref}}$)	3
swing residuals ($\boldsymbol{p}_r$)	6

Table 6.2: The action space of the centroidal policy.

(forward-backward) axis of the robot body, and spans from -0.4m behind the robot to 0.8m ahead of the robot. The action space consists of the stepping frequency f, the desired CoM velocity $v_x^{\text{ref}}, v_z^{\text{ref}}, \omega_y^{\text{ref}}$, and the swing residuals $\boldsymbol{p}_r$, which modulates components of the low-level controller (Table. 6.2). Since the bounding gait primarily involves movements in the sagittal plane with symmetrical leg movements, the centroidal policy only specifies swing residuals for the left legs, and mirrors these residuals for the right legs.

Low-level Aware Reward Design Due to the alternating landing between front and rear legs, the robot is consistently under-actuated throughout the bounding gait and cannot track arbitrary body velocities at all times. For instance, when the front leg is in contact, achieving a large v_z inevitably induces a large torque in the y direction, making it difficult to maintain a small ω_y However, since this information is *not* directly available to the motion policy, the RL algorithm might explore a lot of infeasible actions during training, which causes unstable body tracking in the low-level controller with large tracking errors. To alleviate this issue and make the centroidal policy aware of this physical constraint, we introduce the cost of the final QP (Eq.(6.3)) as part of the reward function during policy training, which encourages the policy to output feasible velocities. The rest of the reward terms were adopted from the reward function in the prior work [130], which encourages the robot to follow the contact schedule, remain stable, and jump over long distances.

Terrain Design and Curriculum For the robot to gradually master complex jumping skills, we design the four different terrains in the with horizontal and vertical discontinu-

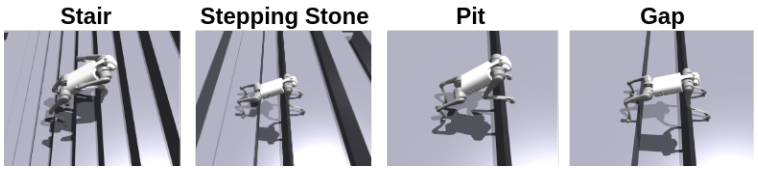


Figure 6.4: The environment includes four different terrain types for the robot to learn versatile jumping skills.

Terrain	Curriculum
Stairs	Step Height: 0.05m $\rightarrow$ 0.3m
Stepping Stones	Gap Width: 0m $\rightarrow$ 0.6m
Pit	Height: 0m$\rightarrow$0.7m
Gap	Width: 0m $\rightarrow$ 1m

Table 6.3: The terrain curriculum introduces a set of horizontal and vertical jumping tasks with increasing difficulty.

ities (Fig. 6.4), which encourages the robot to learn continuous, long-distance jumps with precise foot placements. Each terrain type comes with a curriculum of increasing difficulty (Table. 6.3). Similar to prior work [19, 51, 52], the robot starts in the initial curriculum level at the start of training, and advances to the next level when it can jump sufficiently far in the current level. Within each level, the robot starts in a random position in a randomly-selected terrain type. To simulate the diversity of environments in the real world, we further randomize the step width of stairs and stepping stones within each curriculum level.

Policy Representation and Training Since the motion policy receives the ground-truth heightmap of the surrounding environment, its observation space has sufficient information for decision making. Therefore, we represent the centroidal policy as a MLP without memory, which contains 3 layers of $[512, 256, 128]$ units each with ELU activation [115]. We train our policy using Proximal Policy Optimization (PPO) [43] for 8000 gradient steps,

where each gradient step samples 24 timesteps from 4096 parallelized environments.

6.4.2 Supervised Learning for Heightmap Prediction

Once the motion policy is trained, the next step is to convert this policy with privileged ground-truth heightmap into a policy that uses inputs from on-board sensors. To achieve that, prior works have proposed to train a student policy using imitation learning, where the goal is to imitates either the action or a learned embedding of the teacher policy. We adopt a similar two-staged approach to train our framework. However, unlike these approaches, we train decouple the task of perception from motion planning, and train the "student policy" to *explicitly estimate the heightmap* in the second stage. This modular design improves the interpretability and generalizability of our framework, and leads to more robust terrain estimation under real-world noises.

To effectively fuse past depth information, we design the heightmap predictor as a small 3-layer convolutional network for depthmap processing followed by a 1-layer GRU for memory-based terrain reconstruction. To ensure accurate heightmap estimation around the relevant trajectories, we train the heightmap predictor iteratively using supervised learning, similar to DAGGER [131]. The training loop alternates between rolling out the trajectories using the latest heightmap predictor and the centroidal policy, and training the heightmap predictor on the trajectories collected.

6.5 Results and Analysis

We design our experiments to validate the capability of our framework in jumping over challenging terrains. More specifically, we aim to answer the following questions:

1. Can our framework traverse through challenging discontinuous terrains in the real world?

2. What is the maximum jumping performance of our framework and how does it compare with existing works?

3. Can our framework plan body and foot trajectories proactively in response to perceived terrains?

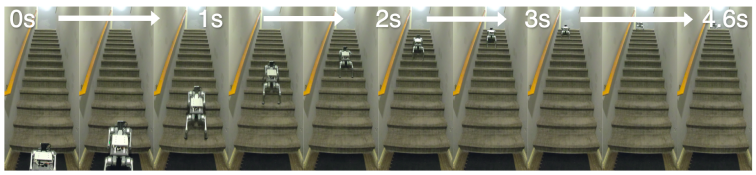
4. What are the important design choices to facilitate successful sim-to-real in our framework?

6.5.1 Experimental Setup

We test our hierarchical learning-control framework on the Unitree Go1 robot [32], where we mounted a Intel Realsense D435i camera to capture depth images, and stream the images to a Mac Mini computer for policy inference. For GPU-accelerated training, we build the simulation environment in IsaacGym [46], and implemented both the high-level centroidal policy and the low-level leg controller in PyTorch [116], so that the entire training loop can be executed on GPU. We train the centroidal policy for 8000 gradient steps using Proximal Policy Optimization (PPO) [43], which takes about 7 hours on a computer with Nvidia RTX 4090 GPU. We train the heightmap predictor for 30 DAgger steps where each step collects 5000 state-action pairs from the environment, which takes about 1 hour to complete on the same computer.

6.5.2 Continuous Jumping over Discontinuous Terrains

To test the performance of our framework in continuous, terrain-aware jumping, we deploy the learned framework in two real-world test terrains (Fig. 6.5). The robot carefully coordinates its body and foot motion and traverses through both terrains at high speeds. In the first case, the robot completes the 14-step staircase in 4.6 seconds, with an average horizontal speed of 0.76m/s and vertical speed of 0.61m/s (Fig. 6.5a). Our framework coordinates the motion of the jumping gait with the stair edges, and crosses the entire 14-step stair cases in less than 8 jumps, with 2 stair steps per jump most of the time. In the second case, we test the robot on a stepping stone environment with 4 stepping stones. The robot jumps across one horizontal gap each time, and traverses the entire terrain in less than 2 seconds.



(a) Stairs (20cm high, 25cm deep)

(b) Stepping Stones (40cm wide, 30-40cm apart).

Figure 6.5: Our robot traverses real-world discontinuous terrains with continuous jumping gaits. The selected frames illustrate the "air phase" of each jumping cycle.

We compare the terrain traversal performance of our framework with a few baselines in perceptive locomotion (Table. 6.4):

Compared to Perceptive Walking Jumping plays a critical role in the performance of our robot. While prior works [132, 106] have demonstrated similar stair-climbing performance on similar-sized robots, these works adopt the walking gait with continuous foot contact, and can climb up to 1 stair step per foot swing due to kinematics limit. Similarly, robots with walking gaits cannot cross large gaps between stepping stones, and are mostly limited to short-distance traversals.

Compared to Single-Step Jumping Another important component of our framework is the coordination of continuous jumping cycles with terrain adaptation. While prior works [52, 51] have demonstrated high-performance single-step jumps, these behaviors are not sufficient for long-distance traversals on stairs and stepping stones, which requires continuous, well-planned jumping cycles to cross repeated discontinuities.

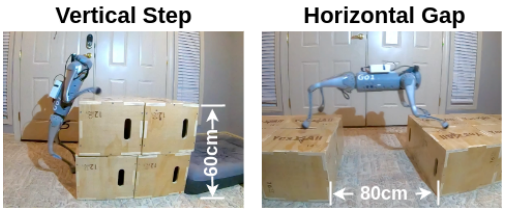
	Step	Gap	Stairs	Stepping Stones
Egocentric-Vision	0.26m	0.17m	0.17m	0.15m
Robot Parkour [51]	0.4m	0.6m	-	-
Extreme Parkour [52]	0.5m	**0.8m**	-	-
Ours	**0.6m**	**0.8m**	**0.2m**	**0.35m**

Table 6.4: Comparison of jumping performance between our framework and baselines. Number shows the maximum step height and gap width achieved by each jump. Note that the baselines [52, 51] were tested on the A1 robot [39], whose motors are 50% stronger compared to our test platform.

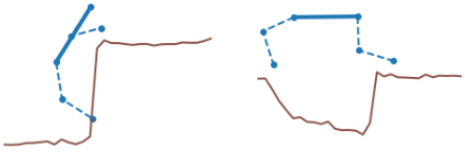
6.5.3 High-performance Jumping over Single Discontinuity

To further validate the capability of our framework, we test our framework in manually constructed terrains with a single discontinuity, such as a vertical step or a horizontal gap (Fig. 6.6a). Our framework handles these terrains well, and can cross horizontal gaps up to 80cm and climb vertical steps up to 60cm. We further visualize the estimated heightmap at the photo frames in Fig. 6.6b, where we plot the estimated heightmap together with the robot pose. Using only proprioception and ego-centric depth image, the heightmap predictor accurately estimates the terrain shape with little penetration or missed contacts. This heightmap is then utilized by the motion policy to coordinate body and leg movements for long-distance jumps.

We compare this performance with prior works on similar jumping tasks [52, 51]. Note that these works were deployed on the A1 robot [39], which is of similar size and weight compared to our platform, but features stronger motors with up to 150% torque output. Despite this hardware limitation, our framework makes [full] use of the motor capabilities, and matches or even exceeds the performance of prior works (Table. 6.4).



(a) Photo of Jump.



(b) Estimated Heightmap.

Figure 6.6: Our robot can jump over terrains with significant vertical and horizontal discontinuities. **Top**: Key frames of robot jumping. **Bottom**: The motion policy uses robot proprioception (blue) and heightmap (brown) estimated from depth images to coordinate the jumping motion.

6.5.4 Emergent Foot and Body Motion Planning

As the core component of our hierarchical framework, the high-level motion policy learns versatile behaviors in body and foot motion coordination. We highlight a few examples here:

Footstep Planning We test the robot's performance in jumping over horizontal gaps of different sizes (Fig. 6.7). While the robot prefers the standard one-step jumping *over* gaps for smaller gap, it chooses to perform a *two-step* for wider gaps with an intermediate landing underneath the gap.

Gait and Body Motion Planning We record the robot's body pitch trajectory on vertical jumps of different heights (Fig. 6.8). The body pitch trajectories look qualitatively similar for different jumps, where the robot first stands up on its rear foot with a negative pitch angle, jumps up, and slowly increases the pitch angle as it lands on the step. Moreover,

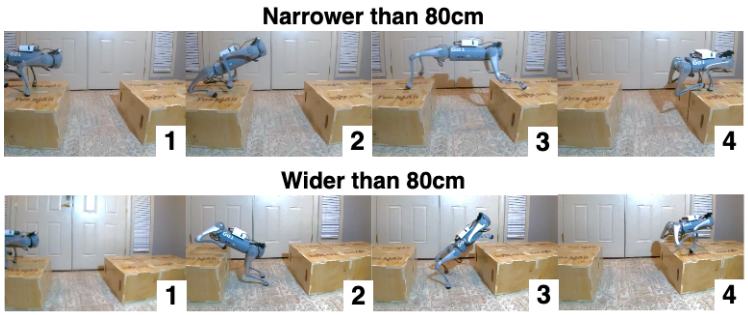


Figure 6.7: The motion policy plans different foot placements for different jumping widths. **Top**: For gaps narrower than 80cm, the robot jumps over the gap directly. **Bottom**: for wider gaps, the robot lands on the ground and jumps over the gap again.

the motion policy carefully plans the timing and body pitch trajectory for different steps, with a larger body pitch variations and longer jumping time for higher jumps.

6.5.5 Jumping over Diverse Staircases in the Real World

To test the generalizability of our framework, we deploy our robot in 5 real-world stairs of different materials and geometries (top row in Fig. 6.1). We run the robot 5 times on each staircase and report the average completion percentage, which is defined as the number of steps traversed by the robot divided by the total number of steps. The result is summarized in Table. 6.5. While our framework performs well on standard-shaped stairs of different dimensions (Stair 1 and 2), the performance degrades when the stair's material (Stair 4) or geometry (Stair 3 and 5) falls outside the training distribution. More specifically, stair 3 contains protruding steps of up to 5cm, where the horizontal part of each step extends outward beyond the vertical part. Stair 5 consists of wooden planks stacked together with noticable gaps between steps. These irregular designs were not seen during training, and can sometimes confuse the heightmap prediction or trap the swing leg. While stair 4 is of standard shape, the marble material is significantly more slippery than a concrete stair, which makes it difficult for the robot to gain sufficient traction for continuous, upwards

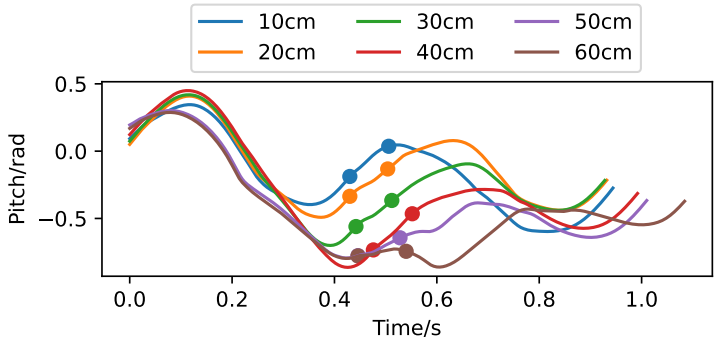


Figure 6.8: The motion policy plans different CoM pitch trajectories for different jump heights. The markers in each trajectory indicates the air duration, from the departure of the rear leg below the step, to the landing of the front leg on top of the step.

jumps. We hypothesize that a more diverse stair representation in simulation can improve the robot's success rate in these terrains, and plan to inspect into this in future work.

6.5.6 Ablation Study

To validate important design choices, we perform an ablation study and compare the performance of our framework with a number of baseline policies. For each policy, we report the *curriculum completion rate* achieved on each terrain. The completion rate is defined as a number between 0 and 1 that corresponds to the the robot's progression through the terrain curriculum (Table. 6.3). For example, achieving a difficulty level of 0.7 on the step terrain means the policy successfully completed 70% of the curriculums on the gap terrain, and can jump over a maximum step of 0.49m. To evaluate the robustness of the policies under dynamics shifts, we test each policy under two scenarios, the *original* and *shifted* environment. We set the original environment to be identical to the training environment, and shift the robot's center-of-mass (CoM) backwards by 4.6cm in the shifted environment. We evaluate all policies in simulation for reproducibility and safety concerns. The result is summarized

	Stair 1	Stair 2	Stair 3	Stair 4	Stair 5
Num Steps	13	6	14	15	10
Material	Concrete	Concrete	Carpet	**Marble**	Wood
Stair Geometry	Standard	Standard	**Protruding Steps**	Standard	**Planked Steps**
Step Width (m)	0.33	**0.42**	0.25	0.33	0.33
Step Height (m)	0.15	0.1	0.2	0.15	0.15
Mean Completion	100%	100%	75%	61.3%	48%

Table 6.5: Performance of our framework on jumping at different staircases in the real world. Bold font indicates profile outside the training distribution.

	Step		**Gap**		**Stairs**		**Stepping Stones**	
	Original	Shifted	Origial	Shifted	Original	Shifted	Original	Shifted
No Feedback	0.6	0.6	0.7	0.4	0.7	0.3	**1**	0.6
No LL Reward	**0.9**	0.7	**1.**	0.8	0.8	0.6	**1.**	**1.**
IL	0.5	0.4	0.2	0.1	0.8	0.6	0.4	0.3
Ours	**0.9**	**0.9**	**1.**	**1.**	**0.9**	**0.7**	**1.**	**1.**

Table 6.6: We compare the curriculum completion rate of our framework with a number of baselines.

in Table. 6.6.

We compare our framework with the following baselines:

No Feedback We remove the velocity-based foot feedback control from the low-level controller, and only use the feedforward optimal control in the low level. We find that this baseline cannot achieve a comparable performance on most terrains, and suffers from larger performance degrades under dynamics shift. This highlights the functionality of the low-level feedback controller in stabilizing the robot under highly dynamic tasks.

No LL Reward We remove the optimal control cost from the reward function of the

centroidal policy, and use the remaining reward terms to train the centroidal policy. In this case, the policy is not directly aware of the limitations of the leg controller, and can sometimes overfit to this limitation with un-transferrable actions. Therefore, while this policy achieves similar performance on the original dynamics in most terrains, its performance drops significantly under dynamics shift, especially for harder tasks that require jumping over large discontinuities.

Imitation Learning (IL) We keep the same RL-trained motion policy. However, instead of learning the heightmap predictor, we choose to directly learn a *student policy* that imitates the actions of the motion policy with consecutive depth image inputs. While this student policy performs well on stairs, it fails to achieve a good performance on other terrains, where the student policy cannot possess the same level of ground-truth terrain information as the teacher policy due to camera occlusions. Our framework alleviates this problem by explicitly specifying the terrain heightmap as the intermediate representation, which coordinates well between the perception and the motion planning modules.

6.6 Related Works

6.6.1 Optimal Control for Dynamic Legged Locomotion

Researchers have had a long history building optimal-control based controllers for high-speed dynamic quadrupedal locomotion [3, 10, 4, 11, 124]. By optimizing for motor control inputs at high frequency, these controllers can accurately track the reference body and foot trajectory, even for high-speed motions like running [3], galloping [10] or jumping [10, 108, 109]. However, due to the constraint for high-frequency real-time control, these frameworks are typically confined to a short plan horizon, and require additional effort in offline trajectory optimization [107, 14, 110, 13], dynamics model simplification [41, 133, 12], and optimal control relaxation [12] to plan for long-horizon jumping motions. Another bottleneck of these frameworks is the difficulty to operate in terrains with vertical and horizontal discontinuities [6, 132, 5]. These discontinuities not only requires additional effort in perception and terrain

segmentation [6], but also increases the complexity of the optimal control problem, making it difficult to solve in real-time [77, 132]. Using manually-designed terrain perception and motion adaptation, Park et al. [5] achieved bounding over hurdles with fixed shapes on an MIT Cheetah 2 Robot. More recently, by classifying step feasibility and formulating it into foot placement optimization, Grandia et al. [6] achieved low-speed walking on uneven stepping stones in the Anymal robot. Compared to these works, our framework uses reinforcement learning for perception and motion planning, and achieves continuous, dynamic locomotion on terrains with large, repeated discontinuities.

6.6.2 Learning Perceptive Locomotion

Recently, learning-based approaches have emerged as a promising alternative to achieve dynamic, terrain-adaptive quadrupedal locomotion [53, 111, 8, 18, 51, 52, 113]. The core idea is to construct an end-to-end policy that directly maps from perceptual and proprioceptive inputs to motor actions, and train the policy using reinforcement learning, usually in simulated environments [46]. These approaches have enabled legged robots to learn end-to-end locomotion policies in challenging terrains, such as mountain trails [9] or stepping stones [134]. More recent works have also shown promising results in agile behaviors such as horizontal or vertical jumping[51, 52, 129, 105]. However, due to the high complexity in end-to-end training, these frameworks usually cannot achieve the same precision as optimal control frameworks, and face difficulty in more complex tasks such as continuous jumping. In contrast, we decompose the control pipeline into perception, motion planning, and low-level motor control, and train each module independently, and achieve continuous, terrain-adaptive jumping for the first time in any quadrupedal robot.

6.6.3 Hierarchical Frameworks

One promising direction to combine the benefit of learning-based and optimal-control-based controllers is to combine them hierarchically, with a learned policy for high-level perception and motion planning, and an optimal controller for low-level motion control [114, 91, 101, 40,

130, 121, 102]. While these frameworks can learn generalizable locomotion behaviors with versatile gait selection [91, 40], foot placement [101, 130, 114] and body motions [101, 130], it can be computationally expensive to train these frameworks, especially with perceptual input. Our prior work [130] speeds up the training of such hierarchical frameworks by relaxing the low-level optimal control problem for GPU-parallelized solving. Building on top of this framework, we incorporated perception into the high-level policy and re-designed key components in reward function and low-level controller for continuous, terrain-adaptive jumping.

6.7 Discussion

In this work, we present a hierarchical learning-control framework for continuous, terrain-adaptive jumping in discontinuous terrains with stairs and stepping stones. Using our framework, a quadrupedal robot achieved animal-like jumping on continuous terrains like stairs and stepping stones, and can traverse through these terrains at significantly higher speed than existing methods. One limitation of our framework is that the robot motion is currently limited to the sagittal plane, and do not support sideways or turning motions. Another limitation is that our framework only follows a fixed forward velocity command, and does not strategically plan its forward path to reach a desired destination. In future work, we plan to address these limitations by expanding the scope of the current training pipeline, and achieve long-horizon agile locomotion in highly complex terrains.

Part II

EXTENDING THE SCOPE OF AGILITY FOR LEGGED ROBOTS

In the first part of the thesis, we present a hierarchical learning-control framework for agile locomotion. While this framework demonstrates high-speed and precise motions, the results are mostly limited to indoor lab environments with rigid objects. In addition, we primarily focused on improving the mobility of the robot in challenging environment. In the second part of the thesis, we extend the scope of agility of legged robots in two important directions. First, we extend the domain of legged robots from indoor, lab environments with rigid objects to *offroad* terrains with complex, deformable terrains. In these terrains, we find that the terrain *semantics* is crucial in guiding robot behavior, and build a imitation learning framework to learn semantics-aware locomotion skills from human demonstratins. In the next work, we extend our focus from *locomotion* to *loco-manipulation*, where we build lightweight custom hardware so that a quadrupedal robot can use its *leg* for versatile manipulation tasks. We use a similar unified hierarchical framework to control this robot in versatile locomotion and loco-manipulation tasks.

Chapter 7

LEARNING SEMANTICS-AWARE LOCOMOTION SKILLS FROM DEMONSTRATIONS

7.1 Introduction

To operate in complex offroad environments, it is crucial for quadrupedal robots to adapt their motion based on the perception of the terrain ahead. When encountering new terrains, the robot needs to identify changes in key terrain properties, such as friction and deformability, and respond with the appropriate locomotion strategy to maintain a reasonable forward speed without incurring failures. In many cases, information about such terrain properties is more easily inferred from a terrain's *semantic* class (e.g. grass, mud, asphalt, etc.) instead of its *geometric* shape (e.g. slope angle, smoothness) [135, 136]. However, recent works in perceptive locomotion [114, 122, 123, 124, 125, 9, 50] mostly focus on the *geometric* aspect of the terrain, and do not make use of such *semantic* information.

In this work, we present a framework for quadrupedal robots to adapt locomotion behaviors based on perceived terrain semantics. The central challenge in learning such a semantic-aware locomotion controller is the high cost of data collection. On the one hand, while simulation has become an effective data source for many robot learning tasks, modeling the complex contact dynamics accurately and rendering photorealistic offroad terrains is not yet possible in simulation. On the other hand, data collection in the real world is time-consuming and requires significant human labor. Moreover, the robot needs to remain safe during the data collection process, as any robot failure can cause significant damage to the hardware and surrounding environment. Therefore, it is difficult to use standard reinforcement learning methods for this task.

Our framework addresses all concerns above, and learns semantics-aware locomotion skills

directly in the real world. To reduce the amount of data required, we pre-train a semantic segmentation network on an off-road driving dataset and extract a semantic embedding from the model for further fine-tuning. To avoid policy exploration in real-world environments, we collect speed choices from human demonstrations and train the policy using imitation learning [137]. Additionally, inspired by previous results on the relationship between speed and gait in animals [54] and legged robots [40, 117], we pair the speed policy with a gait selector to further improve the robot's stability. With the pre-trained image embedding, the imitation learning setup, and the gait selector, our framework learns semantics-aware locomotion skills directly in the real world safely and efficiently.

We deploy our framework on an A1 quadrupedal robot from Unitree [39]. Using only 40 minutes of human demonstration data, our framework learns semantics-aware locomotion skills that can be directly deployed for offroad missions. The learned skill policy inspects the environment and selects a fast and robust locomotion skill for each terrain, from slow and cautious stepping on heavy pebbles to fast and active running on flat asphalts. The learned framework generalizes well and operates without failure on a number of trails not seen during training (over 6km in total). Moreover, our framework outperforms the manufacturer's default controller in terms of speed and safety. We further conduct ablation studies to justify the important design choices.

The technical contributions of this paper include the following:

1. We develop a hierarchical framework that adapts locomotion skills from terrain semantics.

2. We propose a safe and data-efficient method to train our framework directly in the real world, which only requires 40 minutes of human demonstration data.

3. We evaluate the trained framework on multiple trails spanning 6km with different terrain types, where the robot reached high speed and walked without failures.

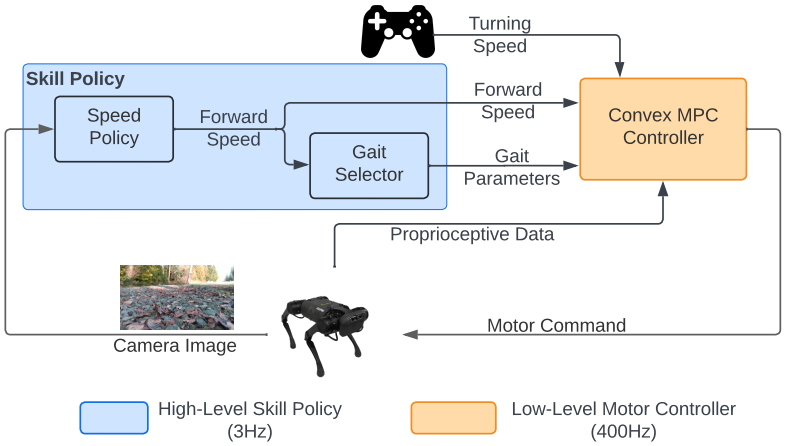


Figure 7.1: Our framework consists of a high-level skill policy and a low-level motor controller. The skill policy selects locomotion skills (gait and speed) based on camera images. The low-level controller computes motor commands for robot control.

7.2 Learning Semantics-Aware Locomotion Skills from Human Demonstrations

7.2.1 Overview

Our hierarchical framework (Fig. 7.1) consists of a high-level skill policy and a low-level motor controller. At the high level, the skill policy receives the RGB image stream from the onboard camera and determines the corresponding locomotion skill. Each skill consists of a desired forward speed and a corresponding locomotion gait, which are computed by the speed policy and gait selector, respectively. We train the speed policy using imitation learning from human demonstrations and manually design the gait selector to find the appropriate gait for each forward speed. At the low level, a convex MPC controller [3] receives the skill command from the skill policy and computes motor commands for robot control. In addition, the convex MPC controller can optionally receive a steering command from an external teleoperator, which specifies the desired turning rate.

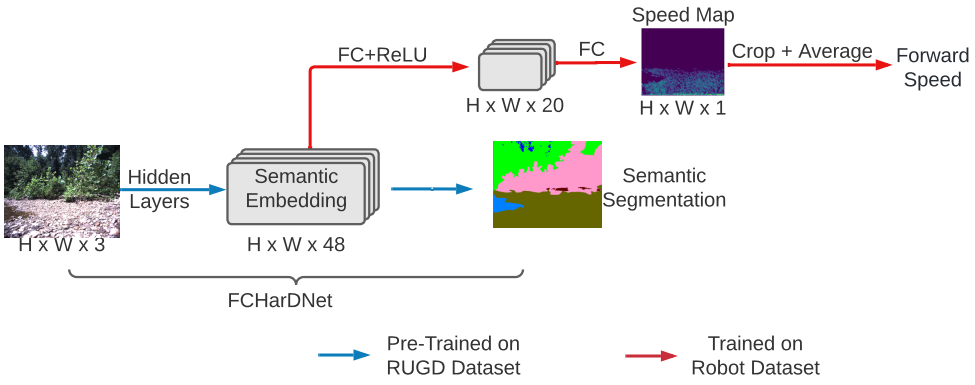


Figure 7.2: Architecture of our perception model. We extract a semantic embedding from a pre-trained semantic segmentation network and use it to learn and predict forward speeds.

7.2.2 Learning Speed Policies

In unstructured offroad terrains, it is crucial for a robot to adjust its speed in response to terrain changes so that it can traverse through different terrains efficiently and without failure. To achieve that, we design a speed policy, which computes the desired forward speed of the robot based on camera images. We train the speed policy using a two-staged procedure: First, we pre-train a semantic embedding from an offline dataset. After that, we collect human demonstrations and train the speed policy using imitation learning.

Pre-trained Semantic Embedding To reduce the amount of real-world data required to train the speed policy, we pre-train a semantic segmentation model and extract a semantic embedding for subsequent finetuning. We implement the model based on FCHarDNet-70 [138], which is a compact fully-convolutional encoder-decoder architecture with good real-time performance. We pre-train the model on the RUGD dataset [139], an off-road driving dataset with pixel-wise semantic labels (grass, dirt, rock, etc.). We choose RUGD because of its similarity to the images collected by the robot camera.

The next step is to extract an embedding from the pre-trained FCHarDNet [138] model for finetuning on robot data. Although the pre-trained model performs well on the RUGD dataset, its predicted segmentation becomes less accurate on robot images due to distribu-

tion shift. Meanwhile, the output of the hidden layers still provides a continuous semantic description for each pixel. Therefore, we extract a *semantic embedding* from the output of the last hidden layer in FCHarDNet, which assigns a 48-dimensional embedding vector to each pixel in the input image (Fig. 7.2). We then compute a speed map by feeding the embedding of each pixel through a fully-connected layer and compute the forward speed by averaging over a fixed region at the bottom of the speed map, which roughly corresponds to a rectangular area 1m long, 0.3m wide in front of the robot. The speed map provides a straightforward and intuitive way to understand the model's predictions and can be used in navigational tasks such as path planning.

7.2.3 Speed-based Gait Selector

In addition to speed, the *gait* of a legged robot, such as its foot swing height, can greatly affect its traversability, especially on uneven terrains. While the perception policy can output speed and gait parameters jointly, training such a policy using imitation learning can be challenging, as it is difficult for the human operator to demonstrate speed and gait choices at the same time. Meanwhile, previous studies in animal [54] and robot [40, 117] locomotion have revealed a close connection between speed and gait choices. Inspired by this discovery, we simplify the demonstration and learning process by designing a heuristic-based *gait selector*, which computes the appropriate gait parameters based on desired forward speed.

Gait Parameterization In our design, each gait is parameterized by three parameters, stepping frequency (SF), swing foot height (SH), and base height (BH). The **stepping frequency (SF)** determines the number of locomotion cycles each second. Similar to [40], we adopt a phase-based parameterization for gait cycles, where each leg alternates between swing and stance. In addition, we assume a trotting pattern for leg coordination, where diagonal legs move together and are 180° out-of-phase with the other diagonal. The trotting pattern is known for its stability, thereby being the default gait choice in most quadrupedal robots [3, 39]. The **swing foot height (SH)** determines the leg's maximum ground clearance

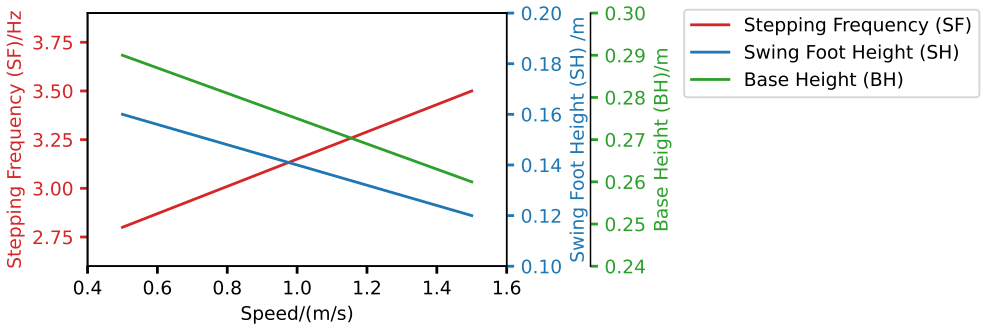


Figure 7.3: The gait selector selects gait parameters (SF, SH and BH) based on desired forward speed. For example, when the desired speed is 0.5m/s, the speed selector would choose a stepping frequency of 2.8Hz, a swing foot height of 0.16m, and a base height of 0.29m.

in each swing phase. While a higher swing height improves stability on uneven terrains by preventing unexpected contacts, a lower swing height is usually necessary for high-speed running. The **base height (BH)** specifies the height of the robot's center-of-mass. While a low base height gives better stability at high speeds, a higher base height can be beneficial when traversing through unknown obstacles.

Speed-Based Gait Selection We use empirical evidence to design the speed-based gait selector, which finds a gait with high traversability for each speed. More specifically, for the boundary speeds (0.5m/s and 1.5m/s), we first try different SFs with a nominal SH (0.12m) and BH (0.26m), and find the lowest SF that would still ensure base stability (2.8Hz and 3.5Hz). After that, we sweep over different values of SH and BH to find the highest value of both that would allow the robot to walk robustly without falling. Lastly, we linearly interpolate the parameter values between the boundary speeds to find the gait for intermediate speeds. See Fig. 7.3 for details.

7.2.4 Low-level Convex MPC Controller

The low-level convex MPC controller computes and applies torques to each actuated degree of freedom, given the locomotion skills from the skill policy. Our low-level convex MPC

controller is based on Di Carlo et al. [3] with two important modifications. Firstly, due to the robot's small form factor, it needs to constantly re-orient its body on uneven terrains, such as bumps and potholes. Therefore, we implemented a state estimator to estimate the ground orientation and adjust the robot pose to fit the ground, similar to Gehring et al. [140]. Secondly, to reduce foot slipping, we implement an impedance controller for stance legs [11]. In addition to the motor torque command computed by MPC, the impedance controller adds a small feedback torque to track the leg in its desired position. We found both techniques to improve locomotion quality significantly.

7.3 Results and Analysis

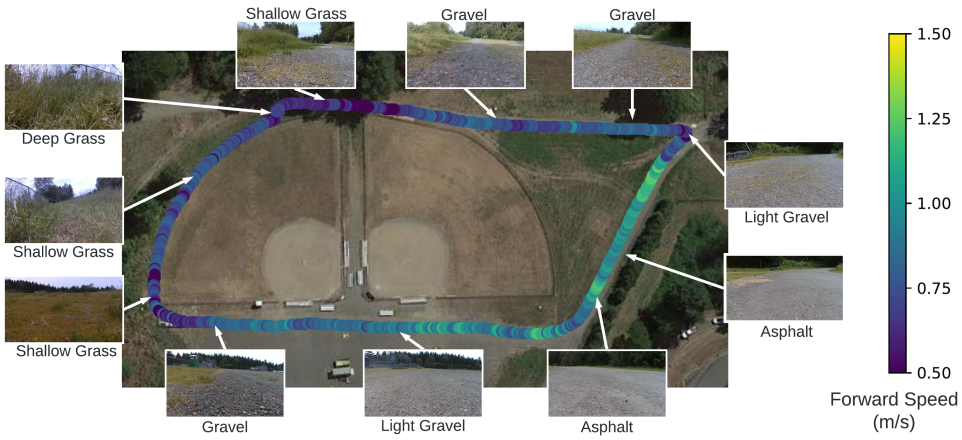


Figure 7.4: The 450m-long test trail consists of multiple terrain types such as deep grass, shallow grass, gravel, and asphalt. The learned skill policy adjusts the speed and gait based on terrain semantics and walks faster on easier terrains.

7.3.1 Experiment Setup

We implement our framework on an A1 quadrupedal robot from Unitree [39]. We equip the robot with an Intel Realsense D435i camera to capture RGB images and a GPS receiver to track its real-time location. We implement the entire control stack in the Robot Operating System (ROS) framework [141] and deploy it on a Mac Mini with M1 chip, which is mounted

on the robot. The convex MPC controller runs at 400Hz, and the speed policy and gait selector run at 3Hz.

To train the speed policy, we collected 7239 frames of data on a variety of terrains, which corresponds to 40 minutes of robot operation. The entire process, including robot setup, data collection, and battery swaps, took less than an hour. We use the FCHarDNet-70 architecture [138], which is obtained from the paper's open-sourced code base. For pre-training on RUGD dataset [139], we train the model for 100 epochs with a batch size of 10 using the Adam optimizer and a learning rate of 0.001. For more robust training, we augment the images from RUGD with random crops and color adjustments. For fine-tuning on demonstration data, we train the model for 60 epochs with a batch size of 32, using the Adam optimizer with the same learning rate of 0.001. Both the pre-training and fine-tuning are conducted on a desktop computer with an Nvidia 2080Ti GPU, where pre-training takes around 6 hours and fine-tuning takes around 20 minutes.

7.3.2 Fast and Failure-Free Walking on Multiple Terrains

To evaluate the adaptation capability of our framework, we test our framework on an outdoor trail with multiple terrain types, including deep grass, shallow grass, gravel, and asphalt (Fig. 7.4). Our controller switches between a wide range of skills as it traverses through the trail, from slow and careful stepping to fast and active walking, and completes the 450m-long trail in 9.6 minutes, comparable to the performance of human demonstrations (10 minutes).

We compared our learned framework with the following baselines on the same test trail (Fig. 7.4), including Unitree's built-in controllers and variants of our controller with no or limited adaptation. The result is summarized in Table. 7.1. For each policy tested in the ablation study, we plot its GPS tracking and failure locations in Fig. 7.5.

Unitree's built-in controllers We tested two modes of the built-in controller, a *normal* mode (Unitree-Normal) that walks up to 1m/s, and a *sports* (Unitree-Sport) mode that walks up to 1.5m/s. Both controllers do not include perception and assume a fixed gait at all times.

Policy Type	**Speed** (m/s)	**Gait Params** (SF, SH, BH)	**Traversal Time** (min)	**Num. Failures**
Fixed-Slow	0.5	[2.8, 0.16, 0.29]	15	**0**
Fixed-Mean	0.8	[3.0, 0.15, 0.28]	∞	3
Fixed-Medium	1	[3.1, 0.14, 0.28]	∞	4
Fixed-Fast	1.5	[3.5, 0.12, 0.27]	∞	10+
Speed-Only	Adaptive	[3.1, 0.14, 0.28]	∞	9
Gait-Only	0.8	Adaptive	∞	2
Unitree-Normal	Tele-operated	N/A	11±0.4	**0**
Unitree-Sport	Tele-operated	N/A	∞	2
Fully-Adaptive (ours)	Adaptive	Adaptive	**9.6±0.2**	**0**

Table 7.1: Performance of different policies on the test trail (450m). Compared to other policies, our framework completes the entire trail without failure in the shortest time. We repeat the Unitree-Normal and Fully-Adaptive policies 3 times and report the mean and standard deviation of the traversal time. We do not repeat the other policies due to excessive robot damage.

Although normal mode completed the entire trail without failure, it walked slower than our learned framework, especially on asphalts, due to limitations on the maximum speed. On the other hand, the sports mode controller failed to complete the course and got stuck in deep grass twice due to insufficient swing foot clearance.

Fixed Skill with No Adaptation For these baselines, we disabled the perception module and operated the robot with a fixed locomotion skill. We tested four skills, namely slow, mean, medium, and fast, operating at 0.5m/s, 0.8m/s, 1m/s, and 1.5m/s, respectively, with the corresponding gait selected according to Fig. 7.3. The slow, medium and fast skills cover the range of possible speeds achievable by our low-level controller, and the mean skill walks at speed similar to the average speed achieved by our adaptive policy (0.78m/s). The mean, medium, and fast skills failed to complete the trail and incurred failures. While the slow

Figure 7.5: GPS tracks (yellow) and failure locations (red cross) for the policies tested in Table. 7.1

skill completed the trail without failure, its traversal time is 50% longer than our learned framework.

Adapt Speed or Gait Only In our framework, we design a robot skill to be a combination of gait and forward speed. To justify this design, we design two policies, where the robot adapts the gait or the forward speed only. For the speed-only policy, we fix the gait parameters as if the forward speed is 1.0m/s in Fig. 7.3 and adapt the speed using our framework. For the gait-only policy, we fix the base speed to be 0.8m/s, similar to the average speed attained by our learned policy, and adapt the gait using our framework. Both policies failed to complete the trail. For the speed-only policy, we found the fixed gait to only work well when the base speed was close to 1m/s and frequently failed at either higher or lower speeds. For the gait-only policy, the robot managed to walk through most of the trail but slipped twice on rocky terrains.

	Trail 1	Trail 2	Trail 3	Trail 4
Trail Length (km)	0.45	0.41	0.2	0.51
Terrain Type	Dirt	Mixed	Asphalt	Mud
Average Speed (m/s)	0.59	0.74	0.94	0.59

Table 7.2: Summary of test trails. Our framework selects different locomotion skills (speed and gait) based on terrain type.

Figure 7.6: Our framework generalizes to unseen terrain types, such as mud, moss, mulch and dirt.

7.3.3 Generalization to Unseen Terrain Instances

We test the performance of our framework in a number of trails not seen during training. Please see Table 7.2 for some examples. These trails include a number of terrain types that are not seen during training, such as mud, moss, mulch, and dirt (Fig. 7.6). Our framework generalizes well to these terrains and enables the robot to traverse through them quickly and safely.

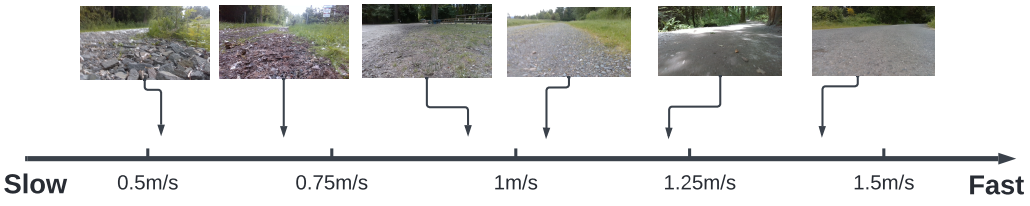
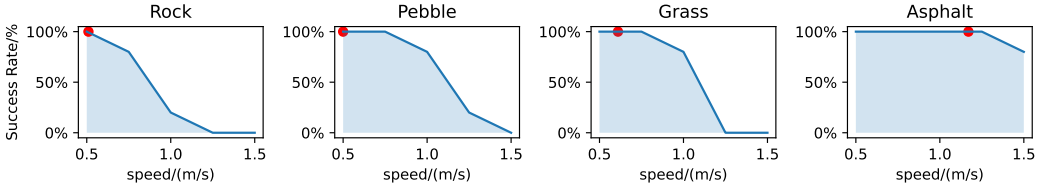


Figure 7.7: Desired speed computed by the skill policy. The policy prefers faster skills for rigid and flat terrains and prefers slower skills for deformable or uneven terrains.

To further demonstrate the skill choices of our framework, we select a few key frames from the camera images and plot the corresponding speed in Fig. 7.7. Generally, the skill policy selects a faster skill on rigid and flat terrains and a slower speed on deformable or uneven terrains. At the time of writing, the robot has traversed through over 6km of outdoor trails without failure.

7.3.4 Analysis on Speed and Safety

(a) We test our robot on different terrain types.



(b) Success rate of fixed locomotion skills (blue) and the learned policy (red).

Figure 7.8: Our framework learns fast and safe locomotion skills. *Top:* We deployed our skill policy to 4 different terrains. *Bottom:* Our policy finds a high speed in the safe region of each terrain.

To test the performance of the learned skill policy in terms of speed and stability, we deploy the learned skill policy on four different terrains, including rock, pebble, grass, and pebble (Fig. 7.8a). We compare our semantics-aware skill policy with 5 fixed skills, where the speed linearly interpolates between 0.5m/s and 1.5m/s. For each speed, the corresponding gait is selected according to Fig. 7.3. For each terrain and skill combination, we repeat the experiment 5 times and report the success rate, where a trial is considered successful if the robot does not fall over during the traversal (Fig. 7.8b). By comparing the success rate at

different speeds, we obtained an approximation of the safe speed range for each terrain. We then test the performance of our framework by comparing the average speed obtained by our learned skill policy on each terrain against these safe speed ranges.

The maximum safe speed varies significantly on different terrains. For example, while the robot can walk up to 1.25m/s without failure on asphalt, it can only walk up to 0.5m/s on rock, due to unexpected bumps and foot slips on the surface. Although not directly optimized for speed or robot safety, our learned policy finds a close-to-maximum speed in the safe region of each terrain after learning from human demonstrations.

We also noted that on pebble and grass, there is a slightly larger gap between the maximum safe speed and the speed selected by the skill policy. One reason for this is that the speed demonstrated by the human operator can be more conservative than the maximum safe speed.

7.3.5 Ablation Study on Perception Module

We compare our way of training the perception-enabled speed policy with a few baselines, which either train the policy from scratch without pre-training or extract the pre-trained embedding directly from the predicted semantic classes. We find that our policy, which is fine-tuned from the output of the hidden layer, achieves the smallest error on the validation set and predicts the speed map with high precision.

Baselines As discussed in Section. 7.2.2, we train the speed policy by finetuning on the pixel-wise semantic embedding, which is extracted from the output of the last hidden layer. To justify this design choice, we compare our way of training the speed policy with two baselines. For the model *finetuned on class labels*, we extract the embedding of each pixel from the one-hot encoding of the model's predicted semantic class. For the model *trained from scratch*, we train the FCHarDNet from scratch on the demonstration data without pre-training.

	Finetuned on Hidden Layers	Finetuned on Class Label	Trained from Scratch
Validation Loss	$\mathbf{0.061 \pm 0.002}$	0.075 ± 0.003	0.088 ± 0.013

Table 7.3: Comparison of performance on different ways of training the speed policy.

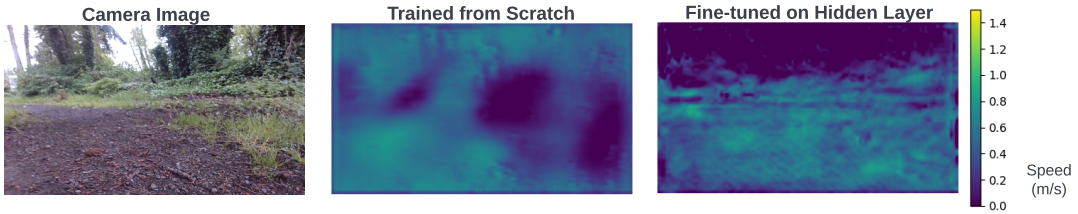


Figure 7.9: Comparison of different ways to predict the speed map. **Left**: the camera image contains multiple semantic classes, including mulch, grass, and trees. **Middle**: the speed model trained from scratch has a low resolution and cannot identify different semantic classes. **Right**: the speed model fine-tuned from semantic embedding accurately identifies different terrain types and computes the desired speed for each terrain.

Results We train our method and the baselines on the demonstration data and test the model's performance on a small validation set, where the data is collected on a different trail. For each model, we repeat the experiment 5 times with different random seeds and report the mean and standard deviation of the loss function (mean-squared loss). The result is summarized in Table. 7.3. Our method, which is finetuned on the output from the hidden layer, achieves the lowest error on the validation set. The model fine-tuned on class label achieves a big loss on both datasets. This is likely due to noisy label prediction, which results from the distribution shift between the model's training data (RUGD) and testing data (robot images). Since the model trained from scratch tunes the entire FCHarDNet on the small set of demonstration data, it overfits to the training data and does not generalize well to the validation set. Moreover, a closer look at the models' predictions shows that the model trained from scratch predicts a blurry speed map with incorrect speed predictions for

several regions, compared to our fine-tuned model (Fig. 7.9). This is likely due to the lack of granularity in demonstration data, which only labels the desired average speed over a fixed region.

7.4 Related Works

Perception for Legged Robots Creating a perceptive locomotion controller is a critical step to enable legged robots to walk in offroad, unstructured environments. Most importantly, it allows robots to detect and react to terrain changes proactively before contact. Many prior works have focused on understanding terrain *geometry* from perceptive sensors [142, 143, 114, 125, 124]. However, such information can be insufficient as it does not reveal important terrain properties such as deformability or contact friction [9, 50, 144, 145]. To ameliorate this, recent works proposed to update this geometric understanding of terrain with proprioceptive information [9, 50, 144]. However, these methods sacrifice proactivity, as the update cannot happen until *after* the robot has stepped on the terrain.

Another approach is to infer the terrain properties from its *semantics* [146, 147, 148, 149] so that the robot can detect changes in terrain property *before* contact, and select its locomotion strategies proactively. Recently, Suryamurthy et al. [150] trained models to predict terrain class and roughness and used the prediction to modulate the height and navigational path of a wheel-leg hybrid robot in an indoor environment. Our framework uses a similar semantics-based approach in the perception module and extends the result to off-road environments with a wide variety of terrains by adapting both the speed and gait of the robot.

Terrain Traversability Estimation The goal of our perception module is to assess the traversability of the terrain ahead of the robot. Researchers have proposed a number of approaches to estimate traversability from perception data, including manually designed [151], learned from self-exploration [143, 152], or learned from human demonstration [153]. While learning-based approaches provide more flexibility, they usually require large amounts

of data, which is difficult to collect in the real world. As a result, most approaches rely on simulation [154] as a source for training data. However, simulation is not feasible for our task, as it is currently difficult to accurately model the complex contact dynamics and create photorealistic renderings of off-road environments. Unlike previous approaches, our framework can be trained directly in the real world and requires only 40 minutes of human demonstration data.

Motion Controller Design for Perceptive Locomotion Another important question in perceptive locomotion is the design of a motion controller that effectively makes use of the perceptive information. A common strategy is to create a low-level motion controller that plans precise foothold placements based on the perceived terrain [114, 122, 123, 124, 125]. While these methods have shown good results in highly uneven terrains, the high computational cost required for terrain understanding and rapid planning makes it infeasible for complex offroad environments. In this work, we devise a novel way to interface between perception and low-level motion controllers for legged robots, where the high-level perception model outputs the desired locomotion skills, including forward speed and robot gait, to a low-level motor controller. With our framework, the robot can select a safe and fast walking strategy for different terrains, which is crucial for offroad traversal.

7.5 Discussion

In this work, we present a hierarchical framework to learn semantic-aware locomotion gaits from human demonstrations. Our framework learns to adapt locomotion skills for a variety of terrains using 40 minutes of human demonstration and enables a robot to traverse over 6km of outdoor terrains without failure. One limitation of our framework is that, while our robot walks robustly on a variety of off-road terrains, its performance is limited by the low dimensionality of human demonstrations. For more difficult terrains such as steps or gaps, the robot will need more agile behaviors such as jumping, which requires a deeper integration between the perception system and low-level motor controller and learning more skills than

speed or gait demonstrations. Another limitation is that the perception system assumes that there is no non-traversable obstacles ahead of the robot and therefore does not adjust the heading of the robot. In future work, we plan to increase the agility of our controller and integrate path planning into our framework so that the robot can operate fully autonomously in challenging off-road environments.

Chapter 8

LOCOMAN: CONNECTING AGILE LOCOMOTION WITH DEXTEROUS MANIPULATION

8.1 Introduction: Current State of Loco-Manipulation

Recent advances in the capability of quadrupedal robots have enabled them to traverse through highly complex terrains [155, 156, 17, 157, 158, 159], perform acrobatic skills [130, 160, 51, 52, 129], and interact with humans [161]. While most of these works focus on improving the *mobility* of quadrupedal robots, the *manipulation* capability of quadrupedal robots still remains limited and often requires special designs to achieve tasks such as button pressing [162] and ball kicking [163, 164]. To be effectively deployed in daily life, quadrupedal robots are required to possess versatile manipulation skills with enhanced dexterity, in addition to their inherent locomotion capability.

Integrating manipulation into quadrupedal robots proves to be a difficult task. As illustrated in Table. 8.1, without external hardware modifications, quadrupedal robots have to utilize their legs, head or torso to move or transport objects [162, 164, 165, 166], which limits their ability to control the 6D poses of the object and perform high-precision tasks. A popular solution to address this challenge is to use a top-mounted robot arm [167]. However, this solution often comes at the cost of increased payload and decreased agility, and prevents the robot from reaching narrow spaces or performing more dexterous tasks such as bimanual manipulation. While foot-mounted grippers offer a lightweight alternative to full robot arms, existing works adopt 1-DoF grippers with limited functionality [168]. Therefore, it can be challenging to achieve versatile dexterous loco-manipulation using existing manipulator solutions for legged robots.

In this work, we present LocoMan, a comprehensive solution for versatile and skilled

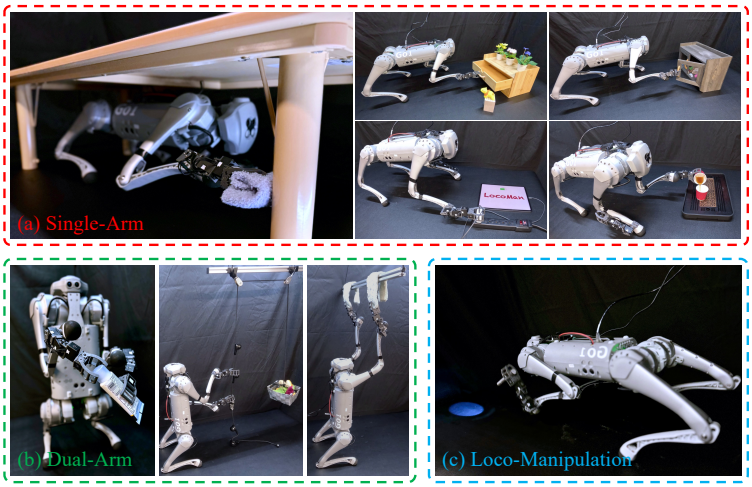


Figure 8.1: Equipped with loco-manipulators, LocoMan is proficient in handling versatile manipulation tasks. (a) With a single loco-manipulator, LocoMan not only can perform manipulation tasks that require precision and stability, but also excels at operating in extremely narrow space in its compact form. (b) With both loco-manipulators installed on the two front legs, LocoMan is able to perform bimanual manipulation tasks when standing upright. (c) LocoMan is also capable of loco-manipulation, e.g. carrying objects with its loco-manipulators while walking.

manipulation with legged robots. To reduce the payload and increase dexterity, LocoMan mounts two custom-designed, lightweight 3-DoF manipulators to the front *calves* of the robot, and uses the existing leg joints in addition to the manipulators for manipulation, as illustrated in Fig. 8.1. We refer to our designed manipulator as "loco-manipulator", which draws conceptual inspiration from the front limbs of animals like great apes and bears, which are adept at both locomotion and skilled manipulation.

For precise and robust control of LocoMan to perform loco-manipulation tasks, we design a unified whole-body control framework, which tracks the command kinematically and dynamically through joint impedance commands. The integration of the loco-manipulator's lightweight design with precise whole-body control enables LocoMan to operate in diverse modes, from reaching an object from narrow-space with single-arm, bimanual manipulation while sitting upright, to carrying an object while walking. We further design a Finite State

Machine (FSM) to manage the transitions of these operation modes.

We evaluate the design of loco-manipulator on a Unitree Go1 robot and find that the added loco-manipulators can significantly increase the robot's workspace (by 99.01% for single-arm manipulation and 118.28% for bimanual manipulation) with negligible weight increase (<2.5%). In addition, the whole-body controller enables precise trajectory tracking with a mean pose error of 1.89 mm and 0.047 rad. We further validate the capabilities of LocoMan in a number of real-life manipulation tasks under user teleoperation. In addition to standard tasks such as cabinet opening and drink pouring, LocoMan can reach narrow spaces with limited vertical clearance (0.25 m for locomotion and 0.09 m for manipulation). Thanks to the lightweight design of loco-manipulator, LocoMan can stand on its rear legs in an upright pose and perform bimanual manipulation tasks such as grasping a pair of socks simultaneously and hoisting a basket with a rope.

In summary, the contribution of this paper is as follows:

1. We present LocoMan, leg-mounted with two lightweight loco-manipulators to improve manipulation capability of quadrupedal robots.

2. We design a unified whole-body controller for precise, simultaneous 6D pose tracking for both the end effectors and the torso.

3. We evaluate that LocoMan is capable of precise trajectory tracking in a large workspace.

4. We demonstrate that LocoMan can perform versatile real-life manipulation tasks, including cabinet opening, charger inserting, drink pouring, grasping in narrow space, and basket hoisting.

8.2 LocoMan: Augmenting Robot Leg for Dexeterous Manipulation

To accomplish versatile manipulation tasks, the robot must be able to reach desired 6D poses in its workspace. While quadrupedal robots can utilize torso movements to aid manipulation

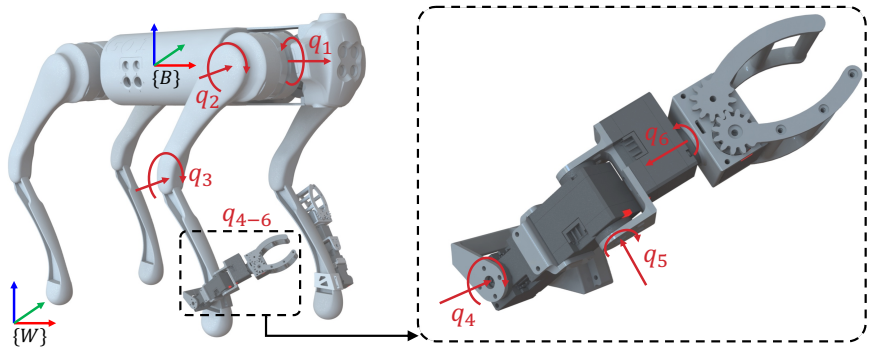
Table 8.1: Comparison of the capabilities between robots with different morphology.

Methods	[162, 164, 169]	[167, 170]	[168]	**Ours**
Additional Manipulator	None	6DoF Top-mounted	1DoF Leg-mounted	3DoF Leg-mounted
6D Operational Space	✗	✓	✗	✓
Narrow Space Manipulation	✗	✗	✓	✓
Bi-Manual Manipulation	✗	✗	✓	✓
Loco-Manipulation	✓	✓	✗	✓

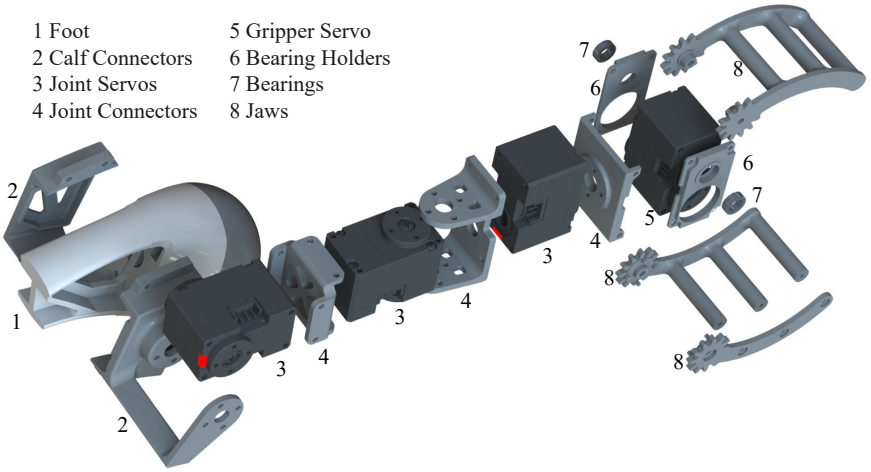
[169, 168], enabling 6D pose reaching directly from the manipulator can significantly improve the precision and range of manipulation tasks. Combining the loco-manipulator DoFs with the existing joint DoFs, each front limb of LocoMan has 6 DoFs in total, and can reach arbitrary 6D space poses *without* body movement.

The design of LocoMan is inspired by the anatomy of human arms. As illustrated in Fig. 8.2(a), we attach custom-designed loco-manipulators to the calf of the front legs of a quadrupedal robot, and utilize the existing leg joints (q_{1-3}) together with the gripper joints (q_{4-6}) for 6DoF space manipulation. The original leg joints, q_{1-3}, similar to human shoulders and elbows, are primarily used for movement and positional tracking. The added joints of loco-manipulator, (q_{4-6}), similar to human wrist, are primarily used for orientation tracking. Integrated together, each front limb of LocoMan can reach a large variety of space poses effectively.

We carefully orient the joints of loco-manipulator to seamlessly integrate with the existing structure of the original robot. Specifically, we design the first joint (q_4) to be aligned with the calf joint of the front leg, design the last joint (q_6) to point towards the gripper, and design the second joint (q_5) to be perpendicular to the other two joints. In this way, the gripper can reach out to far spaces during manipulation (front-right foot of Fig. 8.2(a)),



(a) Joint Configuration of the Legged-Arm



(b) Exploded View of a Right Loco-Manipulator

Figure 8.2: Design of our loco-manipulator. (a) The joint configuration of the legged-arm. The legged-arm has six joints including three from the leg and three from loco-manipulator. (b) The components of a right loco-manipulator shown in an exploded view.

while stay tucked to the calf during standing and walking (front-left foot of Fig. 8.2(a)) with minimum interference to the locomotion capabilities.

The components of loco-manipulator are lightweight, low-cost and easy to fabricate. As illustrated in Fig. 8.2(b), for the actuators of a loco-manipulator, we select four Dynamixel servos, which are compact, highly dynamic, and capable of providing position, velocity, and torque feedback. The loco-manipulator is mounted on the robot's calf via two calf connectors, which are meticulously designed so that their internal structures seamlessly interlock with the

DoFs	3 (joints) + 1 (gripper)
Dimension $[mm^3]$	$179 \times 61 \times 42$
Weight $[g]$	147
Cost[$]	370
4 Servos	XC330-T288-T
2 Bearings $[d \times D \times h]$	$5\ mm \times 8\ mm \times 2.5\ mm$
3D Printer	Formlabs® Form 3+

Table 8.2: Specifications of a loco-manipulator.

skeletal framework of the robot's calf. To make loco-manipulator more compact, we designed the gripper as two pairs of rotating jaws with gear engagement, enabling symmetric opening and closing of the gripper with a single servo. One pair of jaws is connected to the servo horns, while the other pair is mounted with two bearings. Except for the servos (label 3 and label 5 in Fig. 8.2(b)) and the bearings (label 7), the other components of loco-manipulator, including the calf connectors, joint connectors, bearing holders, and jaws, can be 3D printed using a standard 3D printer. Integrated together, each loco-manipulator weighs $147g$ (1.23% of the robot's weight), and can be easily fabricated with a material cost of $370 (Table 8.2).

8.3 A Unified Control Framework for Whole-Body Loco-Manipulation

8.3.1 Notation

We denote the state of LocoMan as $\boldsymbol{q} = (\boldsymbol{x}^{\text{torso}}, \boldsymbol{q}_j) \in \mathbb{R}^{24}$, where $\boldsymbol{x}^{\text{torso}} \in \mathbb{R}^6$ denotes the position and orientation of the floating torso base, and $\boldsymbol{q}_j \in \mathbb{R}^{18}$ denotes the joint angles of LocoMan, including the legs' (12 DoF) and the loco-manipulators' (6 DoF). We denote $\boldsymbol{x}^{\text{eef}} = (\boldsymbol{x}^{\text{eef}}_{\text{left}}, \boldsymbol{x}^{\text{eef}}_{\text{right}}) \in \mathbb{R}^{12}$ and $\boldsymbol{x}^{\text{foot}} \in \mathbb{R}^{24}$ as the cartesian positions and orientations of two grippers and four feet, which can be computed from the state vector using forward kinematics. We use the bar notation $(\bar{\cdot})$ to denote all desired states.

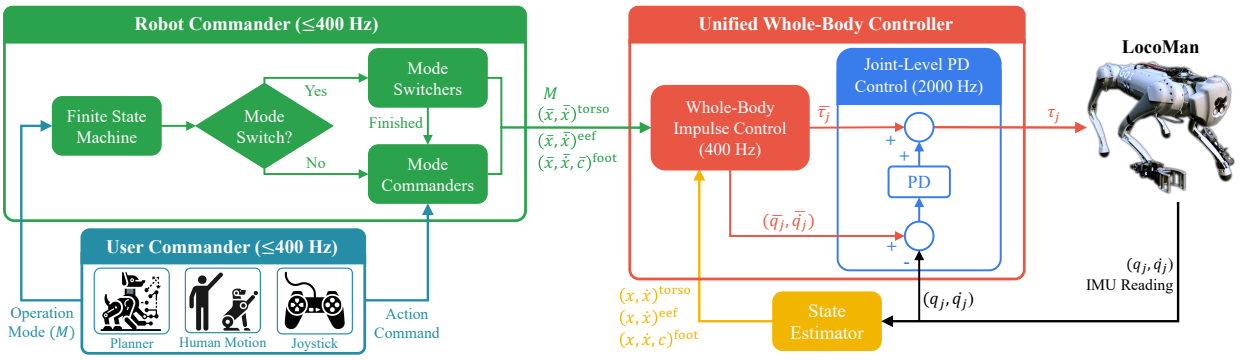


Figure 8.3: The unified framework for whole-body loco-manipulation. The robot commander converts the desired action command from user into the standardized robot command based on a specific operation mode (M). The unified whole-body controller, including whole-body impulse control and joint-level PD control, computes torques for each joint of LocoMan to track the desired robot command,

8.3.2 Overview

Effective loco-manipulation requires the robot to complete a large variety of tasks accurately and robustly, from in-place picking to locomotion while carrying objects. In light of this requirement, we design a unified control framework for LocoMan (Fig. 8.3), where a *robot commander* specifies the desired state of LocoMan based on the action command from user, and an *unified whole-body controller* (WBC) tracks the desired state based on LocoMan's kinematics and dynamics model. The robot commander supports a variety of operation modes (Table 8.3), such as locomotion and in-place manipulation, where the mode switch is governed by a finite state machine (FSM). Within each mode, the FSM launches a mode-specific *mode commander* to convert user specified target (e.g. walking speed or gripper pose) into desired states of the torso $(\overline{\boldsymbol{x}}, \overline{\dot{\boldsymbol{x}}})^{\text{torso}}$, end effector $(\overline{\boldsymbol{x}}, \overline{\dot{\boldsymbol{x}}})^{\text{eef}}$, and foot $(\overline{\boldsymbol{x}}, \overline{\dot{\boldsymbol{x}}}, \overline{\boldsymbol{c}})^{\text{foot}}$. We further design a set of *mode switchers* to smoothly transition between different operation modes. For robust operation and accurate tracking of the desired state, we extend a *whole-body controller* to solve the desired position $\overline{\boldsymbol{q}_j}$, velocity $\overline{\dot{\boldsymbol{q}}_j}$ and torque $\overline{\boldsymbol{\tau}_j}$ of each joint

Operation Mode	Task Tracking Hierarchy	State Estimation
Single Foot Manipulation	Torso Position	Position-Based
	Torso Orientation	
	Foot_m Position	
Single Gripper Manipulation	Torso Position	Position-Based
	Torso Orientation	
	Gripper_m Position	
	Gripper_m Orientation	
Bimanual Manipulation	$\text{Gripper}_{l,r}$ Positions	Position-Based
	$\text{Gripper}_{l,r}$ Orientations	
Locomotion	Torso Velocity	Velocity-Based
	Torso Orientation	
	Foot_s Positions	
Loco-Manipulation	Torso Velocity	Velocity-Based
	Torso Orientation	
	Foot_s Positions	
	$\text{Gripper}_{l,r}$ Orientations	

Table 8.3: Hierarchical tracking objectives and state estimators for the five operation modes. The subscripts $\{m\}$, $\{s\}$, $\{l, r\}$ indicate the end effector in manipulation (m), the feet in swing (s), and the left (l) and right (r) grippers respectively.

based on the complete kinematics and dynamics model of the robot. Then a joint-level PD controller computes the final torque $\boldsymbol{\tau_j}$ for each joint at a higher frequency. In addition, we implement two state estimators to accurately estimate robot state based on sensor feedback. For operation modes with constant foot contact, such as single and bi-manual manipulation, we use a *position-based* state estimator to estimate the 6D body pose from IMU readings and joint angles. For operation modes with contact changes, such as locomotion and loco-

manipulation, we use a *velocity-based* position that estimates the body *velocity* in the forward and side directions.

8.3.3 Robot Commander

The robot commander supports five different operation modes (Table 8.3), namely locomotion, locomanipulation, single-foot manipulation, single-gripper manipulation, and bimanual manipulation. Each mode takes in different user commands from either predefined planners, human motions, or joysticks, which are converted to desired states by a corresponding *Task Commander*.

For single-arm manipulation, the user specifies the desired torso 6D pose, and the desired gripper 6D pose or foot 3D position. For bimanual manipulation, the user commands the desired 6D poses for each gripper. The stance feet remain fixed to the ground during the three manipulation modes. For locomotion and loco-manipulation, the user specifies the desired velocity, roll, pitch, and height of the robot torso, together with the desired gripper orientation for the loco-manipulation mode.

To ensure smooth transition between operation modes, the FSM launches a specific *mode switcher* during mode transitions. For example, from locomotion mode to single-arm manipulation mode, LocoMan is commanded to zero velocity until all feet are on the ground, and then moves the torso to the rear opposite the end effector being manipulated.

8.3.4 Unified Whole-Body Controller

Given the desired states of the torso $(\overline{\boldsymbol{x}}, \overline{\dot{\boldsymbol{x}}})^{\text{torso}}$, end effector $(\overline{\boldsymbol{x}}, \overline{\dot{\boldsymbol{x}}})^{\text{eef}}$, and foot $(\overline{\boldsymbol{x}}, \overline{\dot{\boldsymbol{x}}})^{\text{foot}}$ and foot contact force $\overline{\boldsymbol{c}}^{\text{foot}}$, the WBC computes the desired position $\overline{\boldsymbol{q}_j}$, velocity $\overline{\dot{\boldsymbol{q}}_j}$ and torque $\overline{\boldsymbol{\tau}_j}$ of each joint.

Our WBC implementation is based on the two-step implementation from Kim et al. [10] with additional modeling of the loco-manipulators. In the first step, WBC tracks the desired states *kinematically* by computing the desired position $\overline{\boldsymbol{q}_j}$ and velocity $\overline{\dot{\boldsymbol{q}}_j}$ of each joint using Inverse Kinematics (IK). Since the robot may not have sufficient degrees of freedom to track

the entire desired state, we assign a *objective priority* to each component of the desired states (Table 8.3), and solve objectives in descending priority by projecting low-priority objectives into the null-space of high-priority ones. In the second step, WBC tracks the desired state *dynamically* by optimizing for joint torque commands $\overline{\tau_j}$, where the objective is to track the desired torso acceleration, and the constraints include rigid body dynamics and friction cone limits. Please refer to the original work [10] for additional details.

8.4 Results and Analysis

8.4.1 Experiment Setup

As illustrated in Fig. 8.1, we install two loco-manipulators on the front legs of Unitree Go1, a quadrupedal robot. The servos of the manipulators are powered by the robot via an additional voltage converter (from 24V to 12V). The algorithms of the motion command and control framework run on a desktop, and the control signals for the quadrupedal robot's motors and the loco-manipulators' servos are sent via cables.

8.4.2 Workspace Analysis

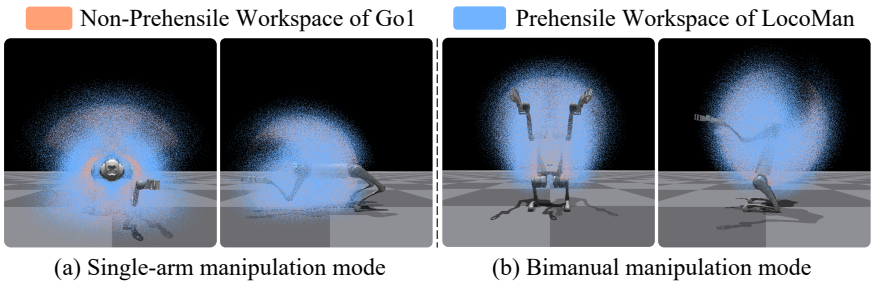


Figure 8.4: Our loco-manipulator turns the original non-prehensile (in orange) workspace of Go1 to a prehensile workspace (in blue) and expand the reachable area by more than 80%.

We compare the workspace of LocoMan with the workspace of the unmodified Unitree Go1 robot as illustrated in Fig. 8.4. The workspace is approximated by uniformly sampling

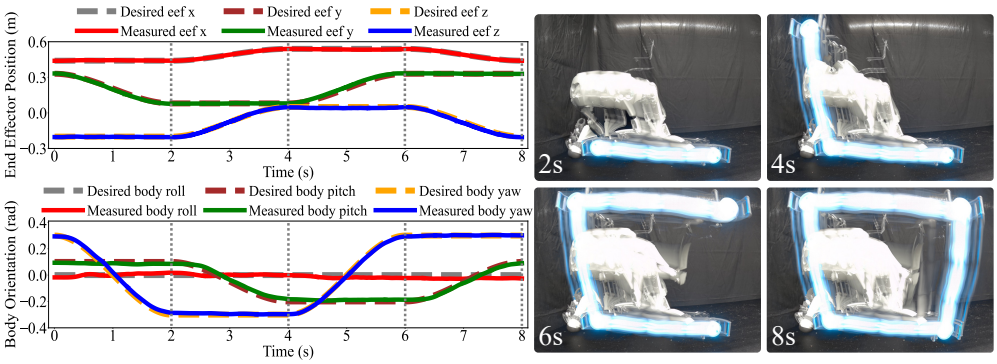


Figure 8.5: The left figure shows trajectory tracking of the end effector position and the torso orientation when drawing a spatial square. The right figure depicts the corresponding 3D trajectory tracked by an LED light attached to the gripper.

multiple angles for each joint within its joint limit, recording the joint configurations without self-collision, and plotting the end effector positions (center of the gripper for LocoMan and foot for the original robot). With additional loco-manipulators, LocoMan extends the workspace of the end effector much further from the original foot toes to the prehensile gripper. Quantitatively, the inclusion of the loco-manipulator expands the workspace volume by at least 80%: from $0.34m^3$ to $0.68m^3$ in the single-arm manipulation mode, from $0.38m^3$ to $0.83m^3$ in the bimanual mode, and from $0.63m^3$ to $1.14m^3$ in the combined volume.

8.4.3 Trajectory Tracking

We design a set of experiments to evaluate the trajectory tracking performance of LocoMan. First, we manually design trajectories of the gripper and torso, where the gripper follows a desired rectangular trace with a spatial bounding box of $0.25m \times 0.1m \times 0.25m$, and the body moves towards the direction of the gripper. We evaluate the quality of the tracking using the Mean Average Error (MAE) of the commanded desired poses and the estimated poses from the kinematics-based state estimator for the manipulator and the torso. As illustrated in Fig. 8.5, the whole-body controller can enable the robot to accurately track the desired

Trajectories	e^t_{eef}	e^R_{eef}	e^t_{torso}	e^R_{torso}
Gripper-M w/o Torso Motion	1.67	0.045	1.92	0.0075
Gripper-M w Torso Motion	1.89	0.047	2.38	0.016
Foot-M w/o Torso Motion	0.93	-	2.02	0.0078
Foot-M w Torso Motion	1.10	-	2.26	0.017
Standing w Torso Motion	-	-	1.05	0.014

Table 8.4: Translational and rotational MAEs of end effector and torso in trajectory tracking. Unit is mm for lengths and rad for angles.

trajectory with tracking MAE of $1.06mm$ in end effector position and $0.023rad$ in the torso orientation.

To further evaluate the tracking performance of the whole-body controller, we define five tasks using end effector with and without torso movement while standing. For each task, we define four different intermediate poses for the end effector and the torso. We then generate an 8-second trajectory with total 3200 target 6D poses using cubic Bezier interpolation from each start pose to its end pose. As illustrated in Table 8.4, the translational and rotational MAEs of the gripper are only $1.89mm$ and $0.047rad$ respectively even with torso movement, which are much more accurate than learning-based method for foot manipulation [169] with translational error of $57mm$. On the other hand, we notice that the translational errors of the torso during manipulation are much larger than those during standing, which may be due to the uneven deformation of the three stance feet. However, the translational errors of the manipulator are smaller than the torso translational errors, benefiting from the task hierarchy of the whole-body controller.

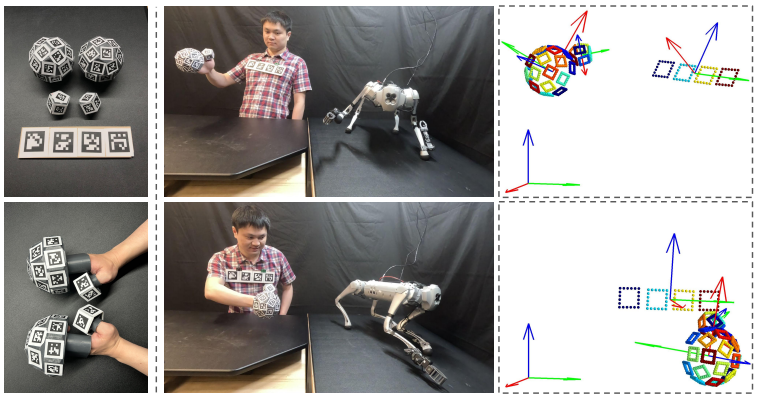


Figure 8.6: Low-cost teleoperation platform. (a) The rigid shells worn by palms, thumbs and torso created by 3D printing. (b) Simultaneous control of the 6D pose of the torso and the end effector of the quadrupedal robot. (c) Visualization of the estimated corresponding 6D poses of the user's right palm, right thumb and the torso using 3D point clouds of AprilTag edges.

8.4.4 Low-cost Vision-based Teleoperation Platform

In LocoMan, the robot torso and end effectors need to be commanded simultaneously, which would be difficult to teleoperate using a single joystick, keyboard or sensor glove. Therefore, we develop a low-cost vision-based teleoperation platform to teleoperate LocoMan using motions of human hands and torso. As illustrated in Fig. 8.6 (a), the core of our teleoperation platform is a pair of 3D-printed rigid shells worn by palms and thumbs of both hands as wells as human torso. The shells are densely covered with AprilTags [171] whose corners and IDs can be robustly detected. Using a method similar to [172, 173], we set up calibrated high-resolution ZED cameras at multiple views and the 6D pose of the rigid shells can be obtained by minimizing the re-projection errors of AprilTag corners in the multiple-view images. The 6D pose of the torso and palm shells are directly used to command the 6D pose of the torso and end effector of quadrupedal robot, while the relative angles between the 6D poses of the thumb and palm shells are used to command the opening and closing of the robot gripper. In this way, the teleoperation users can move their torso and both of

their hands and fingers to naturally control all motion of the quadrupedal robot. With a few cameras and mounts, our teleoperation platform is especially low-cost in hardware compared to previous works, while allowing the users to simultaneously command torso pose, two end effector poses and two gripper openings of the quadrupedal robot.

8.4.5 Versatile Manipulation in Real-World Tasks

Using our teleoperation platform, we validate LocoMan's capabilities listed in Table 8.1 by commanding LocoMan to complete the following real-world robotic tasks using teleoperation:

Manipulation with 6DoF End Effector To validate that LocoMan is capable of performing manipulations in the 6D operational space, we evaluate LocoMan on two challenging tasks: opening a sliding drawer and the swing door of a cabinet. These two tasks demand precise control over the robot end effector pose throughout the interaction. As illustrated in Fig. 8.7 (a), when opening the sliding drawer, LocoMan successfully controls the end effector to move linearly along the sliding direction of the drawer while maintaining a stable orientation axis parallel to the movement path. Fig. 8.7(b) demonstrates LocoMan's capability in opening a cabinet swing door, where it maneuvers the end effector in an arc trajectory and adjusts the gripper orientation to stay perpendicular to the door plane to avoid collision with the door handle.

Furthermore, we teleoperate LocoMan to perform two additional tasks that require precise motion control: inserting a charger into a power socket and pouring liquid from one cup to another. As illustrated in Fig. 8.8 (a), LocoMan can accurately align the charger with the socket, demonstrating its ability to precisely adjust its end effector position. In Fig. 8.8 (b), LocoMan pours liquid from one cup to another, which highlights the robot's ability to stably modulate the end effector orientation.

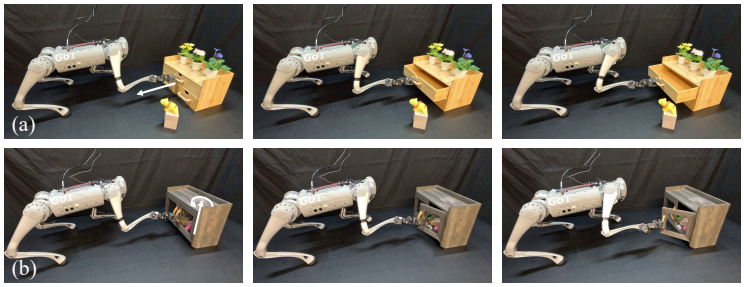


Figure 8.7: LocoMan shows its capability of manipulation in 6D operational space in the tasks of opening two typical types of cabinets under vision-based human motion teleoperation. (a) LocoMan skillfully opens a sliding drawer. (b) LocoMan adeptly opens a swing door cabinet.

Figure 8.8: LocoMan demonstrates its precision and stability in completing two tasks that demand fine control under joystick teleoperation. (a) Insert a charger with meticulous alignment. (b) Pour liquid from a cup to another, carefully modulating the pour angle and rate.

Manipulation in Narrow Space

Fetching objects from extremely narrow space is a challenge for quadrupedal robots due to the limited space for adjusting both the leg and manipulator joint angles. As illustrated in Fig. 8.9 (a), the task requires LocoMan to walk below a bed to grasp an object located under a cabinet and then carry it out. This confined environment only offers a 0.25 m clearance below the bed and an even more constricted 0.09 m height under the cabinet. As illustrated in Fig. 8.9 (b), LocoMan can walk stably below the vertically restricted space, avoiding any contact with the bed. This is only possible without additional robot arm

installed on the top such as the ones in [167, 170]. Thanks to the compact design of loco-manipulator, LocoMan can deftly maneuver its end effector to extend under the cabinet and adjust its pose to successfully grasp the object. LocoMan then retrieves its loco-manipulator while holding the object in the gripper and exits the narrow space.

The successful completion of this challenging task demonstrates LocoMan's competency in loco-manipulation within narrow and complex environments. This showcases its potential for deployment in scenarios such as search and rescue missions, which often necessitate both agile locomotion and manipulation in cramped spaces.

Bimanual Manipulation

Enabling quadrupedal robots to execute bimanual manipulation significantly broadens the spectrum of tasks they can undertake. With dual lightweight loco-manipulators installed on LocoMan's two front legs, it becomes practical for LocoMan to stand upright and engage in complex bimanual tasks.

Hoisting objects with a rope is a task that demands precise bimanual coordination to ensure the rope does not slip back. As illustrated in Fig. 8.10 (a), while maintaining an upright stance, LocoMan adeptly adjusts the 6D pose of an end effector to securely grasp the rope and applies a downward pull. Upon reaching the limits of its workspace, the pulling end effector maintains its grip on the rope, steadfastly holding it in place until the alternate end effector takes over to continue pulling the rope. The seamless transition and successful execution of the hoisting operation vividly illustrate LocoMan's advanced bimanual coordination capabilities, which showcase the practical versatility and enhanced operational scope of quadrupedal robots equipped with our proposed loco-manipulators.

Furthermore, as Fig. 8.10 (b) shows, LocoMan can grasp a pair of socks hanged about 1 m high from the ground, which is relatively high compared to the height of the robot torso during four legs stance.

Loco-Manipulation

As illustrated in Fig. 8.1 (c), we instruct LocoMan to manipulate a flashlight aimed at a predetermined orientation trajectory within the world frame, while trotting forward at a velocity of $0.15m/s$ and shifting left at $0.05m/s$. This necessitates the coordination of its locomotion and manipulation capabilities. Throughout the execution of this task, we record the orientation of the end effector, and also the orientation of its conjunct foot for calculating the vanilla end effector orientation based on their initial relative orientation. As illustrated in Fig. 8.11, in the loco-manipulation mode, LocoMan can track the desired orientation of the flashlight significantly better than relying solely on locomotion.

8.5 Related Works

8.5.1 Quadrupedal Loco-Manipulation

Without Additional Manipulators Researchers have had a long history exploring the manipulation capability of quadrupedal robots. Without an additional manipulator, researchers have designed the robot to use its head [166, 174, 175, 176], torso [165], or foot [162, 163, 164, 177, 178, 179, 180] to interact with objects in the environment, where [174, 175, 176] rely on multiple robots to push larger size objects. However, without an additional mechanism like a gripper, these approaches are limited to relatively simple tasks that do not require fetching objects, such as kicking balls [163], pressing buttons [162], pushing boxes [176], opening doors [179], and probing [178].

With Additional Manipulators Many previous works mount a dedicated arm on top of the quadrupedal robot to perform more complicated manipulation tasks such as pick-and-place [181, 182], pulling doors [183, 170, 184], tablecloth spreading [185], turning wheels [186], wiping a whiteboard [167], and collaboratively carrying large objects [187, 188]. On the other hand, equipping a robot with a dedicated arm typically adds to the mechanical complexity and power requirements, leading to an increase in both weight and expense. Moreover, such an addition makes it difficult to operate in a narrow space and reduce quadrupedal robot's

agility. Some other works alleviate these problems by installing more lightweight manipulators or grippers on the mouth [189] or foot [168, 169] of the quadrupedal robot. Tsvetkov et al. [168] propose a novel design of a small-scale quadrupedal robot with manipulators built into the legs, each requires three additional actuators, where two are used for grasping. It facilitates single-arm and two-arm manipulation but only supports specifying target positions in joint space. Arm et al. [169] propose learning-based controller that can track quadrupedal foot position targets through whole-body motions, and a 2-DoF gripper is attached to the foot to perform tasks like collecting rock samples. However, how to efficiently reach a specified 6D pose of the end effector on the leg to perform more complex manipulation tasks is still underexplored. While some hexapod robots [190, 191, 192, 193, 194] use legs as manipulators where the integrated grippers can fetch objects with high degrees of freedom, prehensile manipulation with dexterity could be challenging for quadrupedal robots.

8.5.2 Controllers for Leg-Manipulation

To accomplish a wide range of manipulation tasks, the controller of the loco-manipulation robot need to support a variety of operation modes, including reaching a narrow space with a single arm, walking while carrying an object, and bi-manual manipulation with upright pose. While prior works have achieved similar tasks such as grasping [170, 183, 167], object pushing and kicking [166, 163, 164], and object carrying [193] using optimization-based [170, 183, 181, 166, 193] or learning-based [165, 163, 164, 167, 162] methods, most of these controllers are designed for specific tasks, and additional effort is required for multi-task support. In contrast, LocoMan adopts a unified control framework, which uses the same low-level whole-body controller [10] for all five operation modes. By tracking desired trajectories at both the kinematics and dynamics level, our unified framework requires little task-specific tuning and performs all manipulation tasks at high accuracy.

8.6 Discussion

In this paper, we present LocoMan, a novel approach that enhances the manipulation dexterity of quadrupedal robots through the integration of designed lightweight loco-manipulators, expanding their operational workspace and enabling precise control over complex 6D manipulation tasks. Our design effectively combine the mobility of the quadrupedal robots with the functionality of manipulators, without compromising agility or requiring extensive payload capacity. The developed unified control framework assures accurate and stable movement across a spectrum of tasks, illustrating LocoMan's versatility in environments ranging from confined spaces to tasks requiring intricate dual-arm coordination. In the future, we would utilize this platform to train a wider range of quadrupedal loco-manipulation tasks through imitation learning, and achieve smooth and efficient mode switch by reinforcement learning.

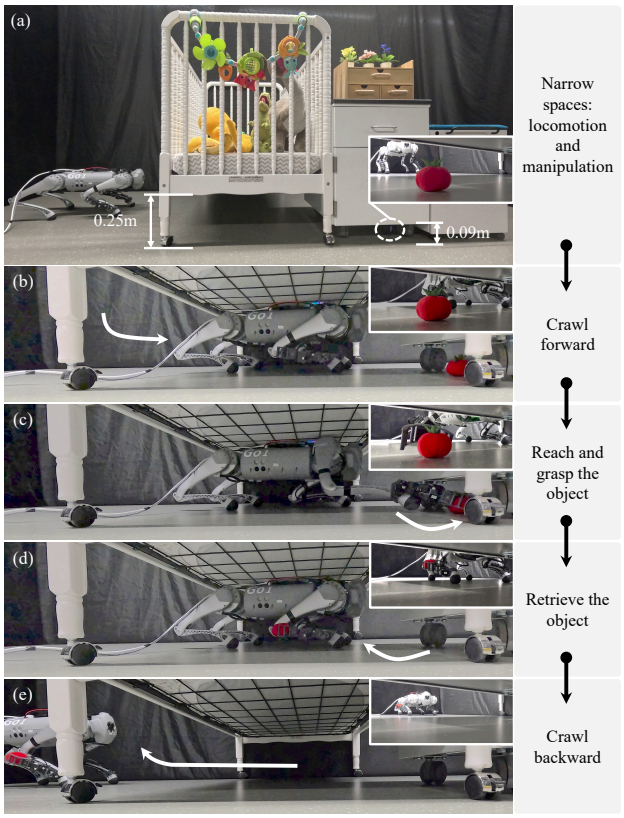
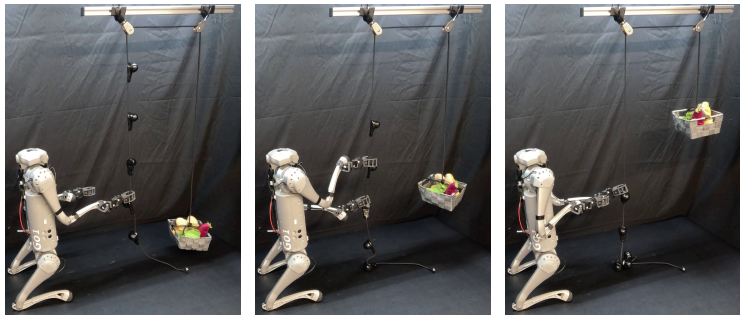
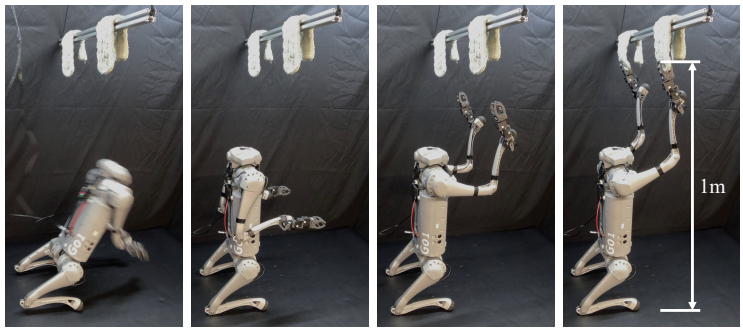


Figure 8.9: LocoMan demonstrates its locomotion and manipulation capabilities in constrained environments under joystick teleoperation. (a) The task requires the robot to walk beneath a crib with a clearance of 0.25 m and grasp the object under the cabinet with a clearance of 0.09 m. (b) LocoMan crawls forward to approach the targeted object. (c) LocoMan extends its loco-manipulator to reach and grasp the object. (d) LocoMan retrieves the object. (e) LocoMan crawls backward and exits the crib.



(a) Hoist a Basket



(b) Grasp a pair of Socks

Figure 8.10: LocoMan is capable of performing challenging bimanual manipulation tasks with its dual lightweight loco-manipulators. (a) LocoMan collaboratively operates its two grippers to hoists a basket under vision-based human motion teleoperation. (b) LocoMan can reach $1m$ high to grasp a pair of socks under joystick teleoperation.

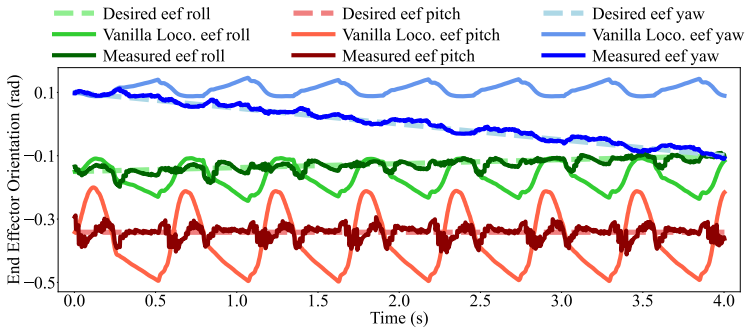


Figure 8.11: Vanilla Loco. denotes carrying the flashlight using the vanilla locomotion mode. The curves indicate that the loco-manipulation mode of LocoMan can consistently track the desired orientation of the flashlight in the loco-manipulation mode.

Chapter 9

CONCLUSION

In this thesis, we demonstrated the effectiveness of combining reinforcement learning (RL) with optimal control (OC) to advance the agility of legged robots. To this end, we develop an algorithmic framework that can achieve high-speed, terrain-aware locomotion on quadrupedal robots,and extend the framework to broader applications including off-road locomotion and loco-manipulation.

First, we used this hierarchical framework to learn speed-aware energy-efficient gait selections. By designing a smooth gait representation for the high-level gait policy and training the gait policy with the low-level MPC controller in the loop, our framework learns a wide variety of gaits with smooth gait transitions. Moreover, the framework retains the robustness of MPC-based controllers, and can walk robustly under real-world perturbations.

We then focus on the task of continuous jumping using a similar hierarchical learning-control framework. We start by learning a residual policy in the high level to complement a heuristically designed motion planner, and then redesigned the high-level policy to output the contact schedule, swing foot trajectory, and body trajectories simultaneously. Additionally, we redesigned the low-level optimal controller, so that the entire optimal controller can be executed efficiently on GPU. Using this framework, we can learn a variety of locomotion gaits, including walking, bounding, and pronking, in less than 20 minutes on a single GPU. Lastly, we incorporate perception into this hierarchical framework, so that the robot can adjust its motion plans based on terrain information. We adopt a modular design with an additional heightmap predictor, and redesigned the low-level controller and the policy-controller interface to reduce the sim-to-real gap. The final version of our hierarchical learning-control framework is able to perform high-speed continuous jumping on stairs and

stepping stones, as well as long-distance horizontal or vertical jumps on terrains with single discontinuities.

Building on top of the success in continuous jumping, we extend the scope of agile legged locomotion to more environments and more tasks. In the first work, we train a skill policy to select different locomotion gaits based on perceived terrain semantics, so that the robot can achieve high-speed, failure-free locomotion in complex offroad environments. In the second work, we extend legged robot hardware with custom-built grippers, so that the robot can use its foot for versatile loco-manipulation. Using a similar hierarchical framework, the robot can perform versatile tasks including single-arm and bi-manual manipulation.

I would like to conclude this thesis with a remark on the role of optimal control in "modern" robotics, and discuss a few exciting future directions.

9.1 Optimal Control as an IDEA

While Optimal Control is an important component of "classical" robotics, and have been deployed in a variety of robot applications, modern-day robotics tend to shift away from this paradigm, and pursuit more data-driven, learning-based approaches, with good successes [195, 196, 197, 198, 199, 200]. As we collect more data and develop better ways (e.g. Transformers) to handle large amounts of data, data-driven approaches will likely enable even more complex, human-like behaviors on robots. In this case, a natural question to ask would arise:

What would be the benefit of optimal control in the area of big data and large models?

To me, optimal control is not just a set of methods, it is an important *idea* in the design of robot controllers. While optimal controllers can take various forms, the idea of *leveraging domain knowledge, maintaining structure and interpretability, and online re-optimization* explains their effectiveness in robotic applications. Under the guidance of these ideas, we introduce domain knowledge to the model-free reinforcement learning process, construct modular hierarchical frameworks, and maintain a robust low-level controller throughout the works in this thesis, which is the key to achieve agile behaviors on real robots.

However, this is not to say that the proposed framework has completely integrated these "ideas" with data-driven approaches. Such a manually designed, hierarchical combination of learning with optimal control inevitably leads to significant amounts of efforts in the policy-controller interface design, as well as close monitoring of the training progress. While this framework does retain a lot of the benefits of optimal control, especially in coordinating robust and smooth of robot behaviors, designing such a framework can sometimes cost significant human labor and (at least partially) break the promise of "autonomous end-to-end training" of learning-based methods.

So how would we better integrate optimal control with data-driven approaches? While I cannot identify a clear answer from the current research progress, we can certainly draw some inspiration from the developments in other fields of machine learning. For example, "modern" designs of convolutional neural networks and recurrent neural networks mostly retain the structure of convolutional filters or auto-regressive time series models. These network architectures retain the idea of hand-designed models, while seamlessly integrate with machine learning techniques to enable expressive behaviors. In the future, I hope to see locomotion frameworks that can learn complex, robust and real-world deployable policies with minimal human effort.

9.2 Future Directions

To conclude this thesis, we would like to outline two exciting directions of future research.

9.2.1 From Quadrupedal Robot to Humanoid Robots

While we have seen rapid developments in both the hardware and the control frameworks of legged robots, it is important to note the inherent limitations of these platforms. Due to the lack of passive-standing mechanisms, these robots consume significant energy to stand up and maintain balance, and can operate for significantly shorter hours compared to a robot vehicle of similar size and weight. Moreover, to maintain stability, these robots typically adopt a low body height (< 60cm), and have difficulty reaching common objects in world

occupied by human. Unless these bottlenecks are resolved, it can be difficult to deploy quadrupedal robots widely into real-life applications.

Fortunately, years of development in quadrupedal robot hardware has led to exciting progress of humanoid robots [33, 34, 35, 201], which addresses both bottlenecks mentioned. Compared to quadrupeds, humanoids can achieve a much larger variety of agile behaviors, from track-and-field sports to high-speed rock climbing. They also seamlessly integrate locomotion with manipulation, enabling a versatile range of dexterous behaviors. An exciting future direction of this thesis is to achieve agile locomotion and dexterous manipulation on humanoid robots. To achieve that, it would be important to ensure the *safety and stability* of the robot, while enabling the robot to learn a *diverse* range of skills at the same time.

9.2.2 Integrate Foundational Models into Legged Robots

While massive-scale training in simulation and sim-to-real has enabled a wide variety of robot behaviors, it is important to notice the limitations of such approach when facing the complexity of the real world. For example, the stair jumping policy mentioned in this thesis, which are only trained on rectangular-shaped stairs in simulation, can easily scale to rectangle-shaped stairs in the real world, but faces significant difficulty in scaling to more complex stair shapes such as planked stairs with no vertical connections. While it is possible to reduce this sim-to-real gap by simulating a wider range of stairs, the task can get prohibitively difficult when we try to recreate *all* real-world discontinuous terrains in simulation. How do we develop a robot that can handle the complexity of real-world from limited simulation trainings?

Recent advancements in foundational models provide a lot of insights for this question. Modern day foundational models, especially visual language models such as GPT4 [] or LLAMA [] have developed a reasonable understanding about the real world [195, 196, 197]. Such understanding has been shown to provide insights for robots in real-world applications []. It would be exciting to develop a deeper connection between foundational models and existing robot controllers, and extend the agility and dexteriy of the robots today into complex,

everyday scenarios.

BIBLIOGRAPHY

[1] Gerardo Bledt, Matthew J Powell, Benjamin Katz, Jared Di Carlo, Patrick M Wensing, and Sangbae Kim. Mit cheetah 3: Design and control of a robust, dynamic quadruped robot. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2245–2252. IEEE, 2018.

[2] ANYbotics. Anymal: Autonomous legged robot. Product Information, 2020. URL https://www.anybotics.com/anymal.

[3] J. Di Carlo, P. M. Wensing, B. Katz, G. Bledt, and S. Kim. Dynamic locomotion in the mit cheetah 3 through convex model-predictive control. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–9, 2018. doi: 10.1109/IROS.2018.8594448.

[4] Tomislav Horvat, Kamilo Melo, and Auke J Ijspeert. Model predictive control based framework for com control of a quadruped robot. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3372–3378. IEEE, 2017.

[5] Hae-Won Park, Patrick M Wensing, and Sangbae Kim. Jumping over obstacles with mit cheetah 2. *Robotics and Autonomous Systems*, 136:103703, 2021.

[6] Ruben Grandia, Fabian Jenelten, Shaohui Yang, Farbod Farshidian, and Marco Hutter. Perceptive locomotion through nonlinear model-predictive control. *IEEE Transactions on Robotics*, 39(5):3402–3421, 2023.

[7] Jie Tan, Tingnan Zhang, Erwin Coumans, Atil Iscen, Yunfei Bai, Danijar Hafner, Steven Bohez, and Vincent Vanhoucke. Sim-to-real: Learning agile locomotion for

quadruped robots. In *Proceedings of Robotics: Science and Systems*, Pittsburgh, Pennsylvania, June 2018. doi: 10.15607/RSS.2018.XIV.010.

[8] Ashish Kumar, Zipeng Fu, Deepak Pathak, and Jitendra Malik. Rma: Rapid motor adaptation for legged robots. *Robotics: Science and Systems*, 2021.

[9] Takahiro Miki, Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning robust perceptive locomotion for quadrupedal robots in the wild. *Science Robotics*, 7(62):eabk2822, 2022.

[10] Donghyun Kim, Jared Di Carlo, Benjamin Katz, Gerardo Bledt, and Sangbae Kim. Highly dynamic quadruped locomotion via whole-body impulse control and model predictive control. *arXiv preprint arXiv:1909.06586*, 2019.

[11] Fabian Jenelten, Jemin Hwangbo, Fabian Tresoldi, C Dario Bellicoso, and Marco Hutter. Dynamic locomotion on slippery ground. *IEEE Robotics and Automation Letters*, 4(4):4170–4176, 2019.

[12] He Li and Patrick M Wensing. Cafe-mpc: A cascaded-fidelity model predictive control framework with tuning-free whole-body control. *arXiv preprint arXiv:2403.03995*, 2024.

[13] Alexander W Winkler, C Dario Bellicoso, Marco Hutter, and Jonas Buchli. Gait and trajectory optimization for legged systems through phase-based end-effector parameterization. *IEEE Robotics and Automation Letters*, 3(3):1560–1567, 2018.

[14] Quan Nguyen, Matthew J Powell, Benjamin Katz, Jared Di Carlo, and Sangbae Kim. Optimized jumping on the mit cheetah 3 robot. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 7448–7454. IEEE, 2019.

[15] C Nguyen and Q Nguyen. Contact-timing and trajectory optimization for 3d jumping on quadruped robots. arxiv. 10.48550. *arXiv preprint ARXIV.2110.06764*, 2021.

[16] Jemin Hwangbo, Joonho Lee, Alexey Dosovitskiy, Dario Bellicoso, Vassilios Tsounis, Vladlen Koltun, and Marco Hutter. Learning agile and dynamic motor skills for legged robots. *Science Robotics*, 4(26):eaau5872, 2019.

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

[18] Ananye Agarwal, Ashish Kumar, Jitendra Malik, and Deepak Pathak. Legged locomotion in challenging terrains using egocentric vision. *arXiv preprint arXiv:2211.07638*, 2022.

[19] Nikita Rudin, David Hoeller, Philipp Reist, and Marco Hutter. Learning to walk in minutes using massively parallel deep reinforcement learning. In *Conference on Robot Learning*, pages 91–100. PMLR, 2022.

[20] Atil Iscen, Ken Caluwaerts, Jie Tan, Tingnan Zhang, Erwin Coumans, Vikas Sindhwani, and Vincent Vanhoucke. Policies modulating trajectory generators. In *Conference on Robot Learning*, pages 916–926. PMLR, 2018.

[21] Igor Mordatch, Zoran Popović, and Emanuel Todorov. Contact-invariant optimization for hand manipulation. In *Proceedings of the ACM SIGGRAPH/Eurographics symposium on computer animation*, pages 137–144, 2012.

[22] Mark E. Rosheim. *Leonardo's Lost Robots*. Springer, Berlin, 2006. ISBN 978-3-540-28440-5.

[23] Domenico Laurenza. *Leonardo's Machines: Da Vinci's Inventions Revealed*. David & Charles, Cincinnati, OH, 2006. ISBN 978-0-7153-2637-3.

[24] Tad McGeer. Passive dynamic walking. *International Journal of Robotics Research*, 9 (2):62–82, 1990. doi: 10.1177/027836499000900206.

[25] Theo Jansen. Strandbeest: The dream machines of theo jansen. Website or Exhibition Name, Year. URL https://www.strandbeest.com/.

[26] General Electric. Walking truck technical report. Report by General Electric for U.S. Army, 1969. Developed under the direction of Ralph Mosher.

[27] Marc H Raibert. *Legged robots that balance*. MIT press, 1986.

[28] Boston Dynamics. Bigdog, the rough-terrain robot. Technical Demonstration Overview, 2005. URL https://www.bostondynamics.com/bigdog.

[29] Boston Dynamics. Atlas: The next generation. Product Release Information, 2016. URL https://www.bostondynamics.com/atlas.

[30] Ltd. Honda Motor Co. Asimo: The world's most advanced humanoid robot. Product Information, 2021. URL http://asimo.honda.com.

[31] Boston Dynamics. Spot: The agile mobile robot. Product Information, 2020. URL https://www.bostondynamics.com/spot.

[32] Go1 Website, 2021. URL https://www.unitree.com/products/go1/.

[33] Unitree Robotics. H1: Advanced humanoid robot. Product Information, 2022. URL https://www.unitree.com/products/h1.

[34] Boston Dynamics. Atlas: Electric version of the advanced humanoid robot. Product Demonstration, 2021. URL https://www.bostondynamics.com/atlas.

[35] Figure. Introduction to figure humanoid robot. Product Release Information, 2023. URL https://www.figure.ai/products/humanoid.

[36] Roland Hafner, Tim Hertweck, Philipp Klöppner, Michael Bloesch, Michael Neunert, Markus Wulfmeier, Saran Tunyasuvunakool, Nicolas Heess, and Martin Riedmiller.

Towards general and autonomous learning of core skills: A case study in locomotion. In *Conference on Robot Learning*, pages 1084–1099. PMLR, 2021.

[37] Gilbert Feng, Hongbo Zhang, Zhongyu Li, Xue Bin Peng, Bhuvan Basireddy, Linzhu Yue, Zhitao Song, Lizhi Yang, Yunhui Liu, Koushil Sreenath, et al. Genloco: Generalized locomotion controllers for quadrupedal robots. In *Conference on Robot Learning*, pages 1893–1903. PMLR, 2023.

[38] Roy Featherstone. *Rigid body dynamics algorithms*. Springer, 2014.

[39] A1 Website, 2020. URL https://www.unitree.com/products/a1/.

[40] Yuxiang Yang, Tingnan Zhang, Erwin Coumans, Jie Tan, and Byron Boots. Fast and efficient locomotion via learned gait transitions. In *Conference on Robot Learning*, pages 773–783. PMLR, 2022.

[41] Chuong Nguyen, Lingfan Bao, and Quan Nguyen. Continuous jumping for legged robots on stepping stones via trajectory optimization and model predictive control. In *2022 IEEE 61st Conference on Decision and Control (CDC)*, pages 93–99. IEEE, 2022.

[42] Hae-Won Park, Patrick M Wensing, Sangbae Kim, et al. Online planning for autonomous running jumps over obstacles in high-speed quadrupeds. In *Robotics: Science and Systems*, 2015.

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

[44] Horia Mania, Aurelia Guy, and Benjamin Recht. Simple random search of static linear policies is competitive for reinforcement learning. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 1805–1814, 2018.

[45] Yuxiang Yang, Ken Caluwaerts, Atil Iscen, Tingnan Zhang, Jie Tan, and Vikas Sindhwani. Data efficient reinforcement learning for legged robots. In *Conference on Robot Learning*, pages 1–10. PMLR, 2020.

[46] Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, and Gavriel State. Isaac gym: High performance gpu-based physics simulation for robot learning, 2021.

[47] Xingyou Song, Yuxiang Yang, Krzysztof Choromanski, Ken Caluwaerts, Wenbo Gao, Chelsea Finn, and Jie Tan. Rapidly adaptable legged robots via evolutionary meta-learning. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3769–3776. IEEE, 2020.

[48] Wenhao Yu, Jie Tan, Yunfei Bai, Erwin Coumans, and Sehoon Ha. Learning fast adaptation with meta strategy optimization. *IEEE Robotics and Automation Letters*, 5(2):2950–2957, 2020.

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

[50] Zipeng Fu, Ashish Kumar, Ananye Agarwal, Haozhi Qi, Jitendra Malik, and Deepak Pathak. Coupling vision and proprioception for navigation of legged robots. *CoRR*, abs/2112.02094, 2021. URL https://arxiv.org/abs/2112.02094.

[51] Ziwen Zhuang, Zipeng Fu, Jianren Wang, Christopher G Atkeson, Sören Schwertfeger, Chelsea Finn, and Hang Zhao. Robot parkour learning. In *CoRL 2023*, 2023.

[52] Xuxin Cheng, Kexin Shi, Ananye Agarwal, and Deepak Pathak. Extreme parkour with legged robots. *arXiv preprint arXiv:2309.14341*, 2023.

[53] Gabriel B Margolis, Ge Yang, Kartik Paigwar, Tao Chen, and Pulkit Agrawal. Rapid locomotion via reinforcement learning. *arXiv preprint arXiv:2205.02824*, 2022.

[54] Donald F Hoyt and C Richard Taylor. Gait and the energetics of locomotion in horses. *Nature*, 292(5820):239–240, 1981.

[55] R McN Alexander and AS Jayes. A dynamic similarity hypothesis for the gaits of quadrupedal mammals. *Journal of zoology*, 201(1):135–152, 1983.

[56] Marc H Raibert. Trotting, pacing and bounding by a quadruped robot. *Journal of biomechanics*, 23:79–98, 1990.

[57] Nicolas Heess, Dhruva TB, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, SM Eslami, et al. Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286*, 2017.

[58] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pages 1861–1870. PMLR, 2018.

[59] Wenhao Yu, Greg Turk, and C Karen Liu. Learning symmetric and low-energy locomotion. *ACM Transactions on Graphics (TOG)*, 37(4):1–12, 2018.

[60] Ofir Nachum, Michael Ahn, Hugo Ponte, Shixiang (Shane) Gu, and Vikash Kumar. Multi-agent manipulation via locomotion using hierarchical sim2real. In Leslie Pack Kaelbling, Danica Kragic, and Komei Sugiura, editors, *Proceedings of the Conference on Robot Learning*, volume 100 of *Proceedings of Machine Learning Research*, pages 110–121. PMLR, 30 Oct–01 Nov 2020.

[61] Xingyou Song, Yuxiang Yang, Krzysztof Choromanski, Ken Caluwaerts, Wenbo Gao, Chelsea Finn, and Jie Tan. Rapidly adaptable legged robots via evolutionary meta-learning. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3769–3776, 2020. doi: 10.1109/IROS45743.2020.9341571.

[62] Dai Owaki and Akio Ishiguro. A quadruped robot exhibiting spontaneous gait transitions from walking to trotting to galloping. *Scientific reports*, 7(1):1–10, 2017.

[63] Dong Jin Hyun, Jongwoo Lee, SangIn Park, and Sangbae Kim. Implementation of trot-to-gallop transition and subsequent gallop on the mit cheetah i. *The International Journal of Robotics Research*, 35(13):1627–1650, 2016.

[64] V Radhakrishnan. Locomotion: dealing with friction. *Proceedings of the National Academy of Sciences*, 95(10):5448–5455, 1998.

[65] Nikolaus Hansen. The cma evolution strategy: A tutorial. *arXiv preprint arXiv:1604.00772*, 2016.

[66] Jie Tan, Yuting Gu, Greg Turk, and C Karen Liu. Articulated swimming creatures. *ACM Transactions on Graphics (TOG)*, 30(4):1–12, 2011.

[67] Jie Tan, Zhaoming Xie, Byron Boots, and C Karen Liu. Simulation-based design of dynamic controllers for humanoid balancing. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2729–2736. IEEE, 2016.

[68] Thomas Geijtenbeek, Michiel Van De Panne, and A Frank Van Der Stappen. Flexible muscle-based locomotion for bipedal creatures. *ACM Transactions on Graphics (TOG)*, 32(6):1–11, 2013.

[69] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*, 2017.

[70] Anirudh Vemula, Wen Sun, and J Bagnell. Contrasting exploration in parameter and action space: A zeroth-order optimization perspective. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 2926–2935. PMLR, 2019.

[71] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. http://pybullet.org, 2016–2020.

[72] C Dario Bellicoso, Fabian Jenelten, Péter Fankhauser, Christian Gehring, Jemin Hwangbo, and Marco Hutter. Dynamic locomotion and whole-body control for quadrupedal robots. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3359–3365. IEEE, 2017.

[73] Shamel Fahmi, Carlos Mastalli, Michele Focchi, and Claudio Semini. Passive whole-body control for quadruped robots: Experimental validation over challenging terrain. *IEEE Robotics and Automation Letters*, 4(3):2553–2560, 2019.

[74] Deepali Jain, Atil Iscen, and Ken Caluwaerts. From pixels to legs: Hierarchical learning of quadruped locomotion. *arXiv preprint arXiv:2011.11722*, 2020.

[75] Xue Bin Peng and Michiel van de Panne. Learning locomotion skills using deeprl: Does the choice of action space matter? In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 1–13, 2017.

[76] R McN Alexander. The gaits of bipedal and quadrupedal animals. *The International Journal of Robotics Research*, 3(2):49–59, 1984.

[77] Ruben Grandia, Farbod Farshidian, René Ranftl, and Marco Hutter. Feedback mpc for torque-controlled legged robots. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4730–4737, 2019. doi: 10.1109/IROS40897.2019.8968251.

[78] Peter Eckert, Alexander Spröwitz, Hartmut Witte, and Auke Jan Ijspeert. 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)*, pages 3128–3133. IEEE, 2015.

[79] Duane W Marhefka, David E Orin, James P Schmiedeler, and Kenneth J Waldron. Intelligent control of quadruped gallops. *IEEE/ASME Transactions On Mechatronics*, 8(4):446–456, 2003.

[80] Farbod Farshidian, Edo Jelavic, Asutosh Satapathy, Markus Giftthaler, and Jonas Buchli. Real-time motion planning of legged robots: A model predictive control approach. In *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, pages 577–584. IEEE, 2017.

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

[82] Zachary Manchester and Scott Kuindersma. Variational contact-implicit trajectory optimization. In *Robotics Research*, pages 985–1000. Springer, 2020.

[83] Chiheb Boussema, Matthew J Powell, Gerardo Bledt, Auke J Ijspeert, Patrick M Wensing, and Sangbae Kim. Online gait transitions and disturbance recovery for legged robots via the feasible impulse set. *IEEE Robotics and automation letters*, 4(2): 1611–1618, 2019.

[84] Jonah Siekmann, Kevin Green, John Warila, Alan Fern, and Jonathan Hurst. Blind bipedal stair traversal via sim-to-real reinforcement learning. *arXiv preprint arXiv:2105.08328*, 2021.

[85] Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel van de Panne. Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Transactions on Graphics (TOG)*, 37(4):1–14, 2018.

[86] Xue Bin Peng, Erwin Coumans, Tingnan Zhang, Tsang-Wei Edward Lee, Jie Tan, and Sergey Levine. Learning agile robotic locomotion skills by imitating animals. In *Robotics: Science and Systems*, 07 2020. doi: 10.15607/RSS.2020.XVI.064.

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

[88] KangKang Yin, Kevin Loken, and Michiel Van de Panne. Simbicon: Simple biped locomotion control. *ACM Transactions on Graphics (TOG)*, 26(3):105–es, 2007.

[89] Doina Precup. Temporal abstraction in reinforcement learning. *University of Massachusetts Amherst ProQuest Dissertations & Theses, 2000.9978540.*, 2001.

[90] Tianyu Li, Nathan Lambert, Roberto Calandra, Franziska Meier, and Akshara Rai. Learning generalizable locomotion skills with hierarchical reinforcement learning. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 413–419. IEEE, 2020.

[91] Xingye Da, Zhaoming Xie, David Hoeller, Byron Boots, Animashree Anandkumar, Yuke Zhu, Buck Babich, and Animesh Garg. Learning a contact-adaptive controller for robust, efficient legged locomotion. *arXiv preprint arXiv:2009.10019*, 2020.

[92] Tianyu Li, Hartmut Geyer, Christopher G Atkeson, and Akshara Rai. Using deep reinforcement learning to learn high-level policies on the atrias biped. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 263–269. IEEE, 2019.

[93] Helei Duan, Jeremy Dao, Kevin Green, Taylor Apgar, Alan Fern, and Jonathan Hurst. Learning task space actions for bipedal locomotion. *arXiv preprint arXiv:2011.04741*, 2020.

[94] Takahiro Miki, Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning robust perceptive locomotion for quadrupedal robots in the wild. *Science Robotics*, 7(62):eabk2822, 2022.

[95] He Li, Tingnan Zhang, Wenhao Yu, and Patrick M Wensing. Versatile real-time motion synthesis via kino-dynamic mpc with hybrid-systems ddp. *arXiv preprint arXiv:2209.14138*, 2022.

[96] Matthew Chignoli, Savva Morozov, and Sangbae Kim. Rapid and reliable quadruped motion planning with omnidirectional jumping. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 6621–6627. IEEE, 2022.

[97] Philipp Wu, Alejandro Escontrela, Danijar Hafner, Ken Goldberg, and Pieter Abbeel. Daydreamer: World models for physical robot learning. *arXiv preprint arXiv:2206.14176*, 2022.

[98] Xiaoyu Huang, Zhongyu Li, Yanzhen Xiang, Yiming Ni, Yufeng Chi, Yunhao Li, Lizhi Yang, Xue Bin Peng, and Koushil Sreenath. Creating a dynamic quadrupedal robotic goalkeeper with reinforcement learning. *arXiv preprint arXiv:2210.04435*, 2022.

[99] Tobias Johannink, Shikhar Bahl, Ashvin Nair, Jianlan Luo, Avinash Kumar, Matthias Loskyll, Juan Aparicio Ojea, Eugen Solowjow, and Sergey Levine. Residual reinforcement learning for robot control. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6023–6029. IEEE, 2019.

[100] He Li, Tingnan Zhang, Wenhao Yu, and Patrick M Wensing. Zero-shot retargeting of learned quadruped locomotion policies using hybrid kinodynamic model predictive control. *arXiv preprint arXiv:2209.14123*, 2022.

[101] Zhaoming Xie, Xingye Da, Buck Babich, Animesh Garg, and Michiel van de Panne. Glide: Generalizable quadrupedal locomotion in diverse environments with a centroidal model. *arXiv preprint arXiv:2104.09771*, 2021.

[102] Guillaume Bellegarda and Quan Nguyen. Robust quadruped jumping via deep reinforcement learning. *arXiv preprint arXiv:2011.07089*, 2020.

[103] Christian Gehring, C Dario Bellicoso, Stelian Coros, Michael Bloesch, Péter Fankhauser, Marco Hutter, and Roland Siegwart. Dynamic trotting on slopes for quadrupedal robots. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5129–5135. IEEE, 2015.

[104] Hendrik Kolvenbach, Philip Arm, Elias Hampp, Alexander Dietsche, Valentin Bickel, Benjamin Sun, Christoph Meyer, and Marco Hutter. Traversing steep and granular martian analog slopes with a dynamic quadrupedal robot. *arXiv preprint arXiv:2106.01974*, 2021.

[105] Ken Caluwaerts, Atil Iscen, J Chase Kew, Wenhao Yu, Tingnan Zhang, Daniel Freeman, Kuang-Huei Lee, Lisa Lee, Stefano Saliceti, Vincent Zhuang, et al. Barkour: Benchmarking animal-level agility with quadruped robots. *arXiv preprint arXiv:2305.14654*, 2023.

[106] Ananye Agarwal, Ashish Kumar, Jitendra Malik, and Deepak Pathak. Legged locomotion in challenging terrains using egocentric vision. In *Conference on Robot Learning*, pages 403–415. PMLR, 2023.

[107] Scott Gilroy, Derek Lau, Lizhi Yang, Ed Izaguirre, Kristen Biermayer, Anxing Xiao, Mengti Sun, Ayush Agrawal, Jun Zeng, Zhongyu Li, et al. Autonomous navigation for quadrupedal robots with optimized jumping through constrained obstacles. In *2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)*, pages 2132–2139. IEEE, 2021.

[108] Yanran Ding, Abhishek Pandala, and Hae-Won Park. Real-time model predictive control for versatile dynamic motions in quadrupedal robots. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8484–8490. IEEE, 2019.

[109] Christian Gehring, Stelian Coros, Marco Hutter, Carmine Dario Bellicoso, Huub Heijnen, Remo Diethelm, Michael Bloesch, Péter Fankhauser, Jemin Hwangbo, Mark

Hoepflinger, et al. Practice makes perfect: An optimization-based approach to controlling agile motions for a quadruped robot. *IEEE Robotics & Automation Magazine*, 23(1):34–43, 2016.

[110] Zhitao Song, Linzhu Yue, Guangli Sun, Yihu Ling, Hongshuo Wei, Linhai Gui, and Yun-Hui Liu. An optimal motion planning framework for quadruped jumping. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 11366–11373. IEEE, 2022.

[111] Gabriel B Margolis and Pulkit Agrawal. Walk these ways: Tuning robot control for generalization with multiplicity of behavior. In *Conference on Robot Learning*, pages 22–31. PMLR, 2023.

[112] Arnaud Klipfel, Nitish Sontakke, Ren Liu, and Sehoon Ha. Learning a single policy for diverse behaviors on a quadrupedal robot using scalable motion imitation. *arXiv preprint arXiv:2303.15331*, 2023.

[113] Laura Smith, J Chase Kew, Tianyu Li, Linda Luu, Xue Bin Peng, Sehoon Ha, Jie Tan, and Sergey Levine. Learning and adapting agile locomotion skills by transferring experience. *arXiv preprint arXiv:2304.09834*, 2023.

[114] Wenhao Yu, Deepali Jain, Alejandro Escontrela, Atil Iscen, Peng Xu, Erwin Coumans, Sehoon Ha, Jie Tan, and Tingnan Zhang. Visual-locomotion: Learning to walk on complex terrains with vision. In *5th Annual Conference on Robot Learning*, 2021.

[115] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

[116] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch:

An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[117] Zipeng Fu, Ashish Kumar, Jitendra Malik, and Deepak Pathak. Minimizing energy consumption leads to the emergence of gaits in legged robots. *arXiv preprint arXiv:2111.01674*, 2021.

[118] Gabriel B Margolis, Tao Chen, Kartik Paigwar, Xiang Fu, Donghyun Kim, Sang bae Kim, and Pulkit Agrawal. Learning to jump from pixels. In *Conference on Robot Learning*, pages 1025–1034. PMLR, 2022.

[119] Brandon Amos and J Zico Kolter. Optnet: Differentiable optimization as a layer in neural networks. In *International Conference on Machine Learning*, pages 136–145. PMLR, 2017.

[120] Zhongyu Li, Xue Bin Peng, Pieter Abbeel, Sergey Levine, Glen Berseth, and Koushil Sreenath. Robust and versatile bipedal jumping control through multi-task reinforcement learning. *arXiv preprint arXiv:2302.09450*, 2023.

[121] Yuxiang Yang, Xiangyun Meng, Wenhao Yu, Tingnan Zhang, Jie Tan, and Byron Boots. Continuous versatile jumping using learned action residuals. *arXiv preprint arXiv:2304.08663*, 2023.

[122] Siddhant Gangapurwala, Mathieu Geisert, Romeo Orsolino, Maurice Fallon, and Ioannis Havoutis. Rloc: Terrain-aware legged locomotion using reinforcement learning and optimal control. *arXiv preprint arXiv:2012.03094*, 2020.

[123] Péter Fankhauser, Marko Bjelonic, C Dario Bellicoso, Takahiro Miki, and Marco Hutter. Robust rough-terrain locomotion with a quadrupedal robot. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5761–5768. IEEE, 2018.

[124] Octavio Villarreal, Victor Barasuol, Patrick M Wensing, Darwin G Caldwell, and Claudio Semini. Mpc-based controller with terrain insight for dynamic legged locomotion. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2436–2442. IEEE, 2020.

[125] Fabian Jenelten, Takahiro Miki, Aravind E Vijayan, Marko Bjelonic, and Marco Hutter. Perceptive locomotion in rough terrain–online foothold optimization. *IEEE Robotics and Automation Letters*, 5(4):5370–5376, 2020.

[126] Sang-Ho Hyon, Joshua G Hale, and Gordon Cheng. Full-body compliant human–humanoid interaction: balancing in the presence of unknown external forces. *IEEE transactions on robotics*, 23(5):884–898, 2007.

[127] Matthew Chignoli and Patrick M Wensing. Variational-based optimal control of underactuated balancing for dynamic quadrupeds. *IEEE Access*, 8:49785–49797, 2020.

[128] Ziyi Zhou and Ye Zhao. Accelerated admm based trajectory optimization for legged locomotion with coupled rigid body dynamics. In *2020 American Control Conference (ACC)*, pages 5082–5089. IEEE, 2020.

[129] David Hoeller, Nikita Rudin, Dhionis Sako, and Marco Hutter. Anymal parkour: Learning agile navigation for quadrupedal robots. *arXiv preprint arXiv:2306.14874*, 2023.

[130] Yuxiang Yang, Guanya Shi, Xiangyun Meng, Wenhao Yu, Tingnan Zhang, Jie Tan, and Byron Boots. Cajun: Continuous adaptive jumping using a learned centroidal controller. *arXiv preprint arXiv:2306.09557*, 2023.

[131] 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*, pages 627–635. JMLR Workshop and Conference Proceedings, 2011.

[132] Shuhao Qi, Wenchun Lin, Zejun Hong, Hua Chen, and Wei Zhang. Perceptive autonomous stair climbing for quadrupedal robots. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2313–2320. IEEE, 2021.

[133] Abhishek Pandala, Randall T Fawcett, Ugo Rosolia, Aaron D Ames, and Kaveh Akbari Hamed. Robust predictive control for quadrupedal locomotion: Learning to close the gap between reduced-and full-order models. *IEEE Robotics and Automation Letters*, 7(3):6622–6629, 2022.

[134] Zhaoming Xie, Hung Yu Ling, Nam Hee Kim, and Michiel van de Panne. Allsteps: curriculum-driven learning of stepping stone skills. In *Computer Graphics Forum*, volume 39, pages 213–224. Wiley Online Library, 2020.

[135] James J Gibson. *The ecological approach to visual perception: classic edition*. Psychology press, 2014.

[136] Yu-Wei Chao, Zhan Wang, Rada Mihalcea, and Jia Deng. Mining semantic affordances of visual object categories. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4259–4267, 2015.

[137] Dean A Pomerleau. Alvinn: An autonomous land vehicle in a neural network. *Advances in neural information processing systems*, 1, 1988.

[138] Ping Chao, Chao-Yang Kao, Yu-Shan Ruan, Chien-Hsiang Huang, and Youn-Long Lin. Hardnet: A low memory traffic network. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 3552–3561, 2019.

[139] Maggie Wigness, Sungmin Eum, John G Rogers, David Han, and Heesung Kwon. A rugd dataset for autonomous navigation and visual perception in unstructured outdoor environments. In *International Conference on Intelligent Robots and Systems (IROS)*, 2019.

[140] Christian Gehring, C Dario Bellicoso, Stelian Coros, Michael Bloesch, Péter Fankhauser, Marco Hutter, and Roland Siegwart. Dynamic trotting on slopes for quadrupedal robots. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5129–5135. IEEE, 2015.

[141] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, Andrew Y Ng, et al. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, number 3.2 in 3, page 5. Kobe, Japan, 2009.

[142] Péter Fankhauser, Michael Bloesch, and Marco Hutter. Probabilistic terrain mapping for mobile robots with uncertain localization. *IEEE Robotics and Automation Letters*, 3(4):3019–3026, 2018.

[143] Jonas Frey, David Hoeller, Shehryar Khattak, and Marco Hutter. Locomotion policy guided traversability learning using volumetric representations of complex environments. *arXiv preprint arXiv:2203.15854*, 2022.

[144] Ruihan Yang, Minghao Zhang, Nicklas Hansen, Huazhe Xu, and Xiaolong Wang. Learning vision-guided quadrupedal locomotion end-to-end with cross-modal transformers. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=nhnJ3oo6AB.

[145] Lorenz Wellhausen, Alexey Dosovitskiy, René Ranftl, Krzysztof Walas, Cesar Cadena, and Marco Hutter. Where should i walk? predicting terrain properties from images via self-supervised learning. *IEEE Robotics and Automation Letters*, 4(2):1509–1516, 2019.

[146] Jean-François Lalonde, Nicolas Vandapel, Daniel F Huber, and Martial Hebert. Natural terrain classification using three-dimensional ladar data for ground robot mobility. *Journal of field robotics*, 23(10):839–861, 2006.

[147] Yasir Niaz Khan, Philippe Komma, and Andreas Zell. High resolution visual terrain classification for outdoor robots. In *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, pages 1014–1021. IEEE, 2011.

[148] Fabian Schilling, Xi Chen, John Folkesson, and Patric Jensfelt. Geometric and visual terrain classification for autonomous mobile navigation. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2678–2684. IEEE, 2017.

[149] Amirreza Shaban, Xiangyun Meng, JoonHo Lee, Byron Boots, and Dieter Fox. Semantic terrain classification for off-road autonomous driving. In *Conference on Robot Learning*, pages 619–629. PMLR, 2022.

[150] Vivekanandan Suryamurthy, Vignesh Sushrutha Raghavan, Arturo Laurenzi, Nikos G Tsagarakis, and Dimitrios Kanoulas. Terrain segmentation and roughness estimation using rgb data: Path planning application on the centauro robot. In *2019 IEEE-RAS 19th International Conference on Humanoid Robots (Humanoids)*, pages 1–8. IEEE, 2019.

[151] David D Fan, Kyohei Otsu, Yuki Kubo, Anushri Dixit, Joel Burdick, and Ali-Akbar Agha-Mohammadi. Step: Stochastic traversability evaluation and planning for risk-aware off-road navigation. In *Robotics: Science and Systems*, pages 1–21. RSS Foundation, 2021.

[152] Krzysztof Walas, Dimitrios Kanoulas, and Przemyslaw Kryczka. Terrain classification and locomotion parameters adaptation for humanoid robots using force/torque sensing. In *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*, pages 133–140. IEEE, 2016.

[153] Xiaoyi Cai, Michael Everett, Jonathan Fink, and Jonathan P How. Risk-aware off-

road navigation via a learned speed distribution map. *arXiv preprint arXiv:2203.13429*, 2022.

[154] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. Carla: An open urban driving simulator. In *Conference on robot learning*, pages 1–16. PMLR, 2017.

[155] Fabian Jenelten, Junzhe He, Farbod Farshidian, and Marco Hutter. Dtc: Deep tracking control. *Science Robotics*, 9(86):eadh5401, 2024.

[156] Suyoung Choi, Gwanghyeon Ji, Jeongsoo Park, Hyeongjun Kim, Juhyeok Mun, Jeong Hyun Lee, and Jemin Hwangbo. Learning quadrupedal locomotion on deformable terrain. *Science Robotics*, 8(74):eade2256, 2023.

[157] Ruihan Yang, Ge Yang, and Xiaolong Wang. Neural volumetric memory for visual locomotion control. In *CVPR 2023*, 2023.

[158] Ashish Kumar, Zipeng Fu, Deepak Pathak, and Jitendra Malik. Rma: Rapid motor adaptation for legged robots. *arXiv preprint arXiv:2107.04034*, 2021.

[159] Björn Lindqvist, Samuel Karlsson, Anton Koval, Ilias Tevetzidis, Jakub Haluška, Christoforos Kanellakis, Ali-akbar Agha-mohammadi, and George Nikolakopoulos. Multimodality robotic systems: Integrated combined legged-aerial mobility for subterranean search-and-rescue. *Robotics and Autonomous Systems*, 2022.

[160] Chenhao Li, Marin Vlastelica, Sebastian Blaes, Jonas Frey, Felix Grimminger, and Georg Martius. Learning agile skills via adversarial imitation of rough partial demonstrations. In *Conference on Robot Learning*. PMLR, 2023.

[161] Anxing Xiao, Wenzhe Tong, Lizhi Yang, Jun Zeng, Zhongyu Li, and Koushil Sreenath. Robotic guide dog: Leading a human with leash-guided hybrid physical interaction. In *ICRA 2021*, 2021.

[162] Xuxin Cheng, Ashish Kumar, and Deepak Pathak. Legs as manipulator: Pushing quadrupedal agility beyond locomotion. *arXiv preprint arXiv:2303.11330*, 2023.

[163] Yandong Ji, Zhongyu Li, Yinan Sun, Xue Bin Peng, Sergey Levine, Glen Berseth, and Koushil Sreenath. Hierarchical reinforcement learning for precise soccer shooting skills using a quadrupedal robot. In *IROS 2022*, 2022.

[164] Yandong Ji, Gabriel B Margolis, and Pulkit Agrawal. Dribblebot: Dynamic legged manipulation in the wild. *arXiv preprint arXiv:2304.01159*, 2023.

[165] Seunghun Jeon, Moonkyu Jung, Suyoung Choi, Beomjoon Kim, and Jemin Hwangbo. Learning whole-body manipulation for quadrupedal robot. *IEEE RA-L*, 2023.

[166] Mohsen Sombolestan and Quan Nguyen. Hierarchical adaptive loco-manipulation control for quadruped robots. In *ICRA 2023*, 2023.

[167] Zipeng Fu, Xuxin Cheng, and Deepak Pathak. Deep whole-body control: learning a unified policy for manipulation and locomotion. In *CoRL 2023*, 2023.

[168] Yordan Tsvetkov and Subramanian Ramamoorthy. A novel design and evaluation of a dactylus-equipped quadruped robot for mobile manipulation. In *IROS 2022*, 2022.

[169] Philip Arm, Mayank Mittal, Hendrik Kolvenbach, and Marco Hutter. Pedipulate: Enabling manipulation skills using a quadruped robot's leg. *arXiv preprint arXiv:2402.10837*, 2024.

[170] Jean-Pierre Sleiman, Farbod Farshidian, Maria Vittoria Minniti, and Marco Hutter. A unified mpc framework for whole-body dynamic locomotion and manipulation. *IEEE RA-L*, 2021.

[171] Edwin Olson. Apriltag: A robust and flexible visual fiducial system. In *ICRA 2011*, 2011.

[172] Xingyu Liu, Rico Jonschkowski, Anelia Angelova, and Kurt Konolige. Keypose: Multi-view 3d labeling and keypoint estimation for transparent objects. In *CVPR 2020*, 2020.

[173] Xingyu Liu, Shun Iwase, and Kris M Kitani. Stereobj-1m: Large-scale stereo image dataset for 6d object pose estimation. In *ICCV 2021*, 2021.

[174] Mohsen Sombolestan and Quan Nguyen. Hierarchical adaptive control for collaborative manipulation of a rigid object by quadrupedal robots. *arXiv preprint arXiv:2303.06741*, 2023.

[175] Ofir Nachum, Michael Ahn, Hugo Ponte, Shixiang Gu, and Vikash Kumar. Multi-agent manipulation via locomotion using hierarchical sim2real. *arXiv preprint arXiv:1908.05224*, 2019.

[176] Maja J Mataric, Martin Nilsson, and Kristian T Simsarin. Cooperative multi-robot box-pushing. In *Proceedings 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human Robot Interaction and Cooperative Robots*, volume 3, pages 556–561. IEEE, 1995.

[177] Fan Shi, Timon Homberger, Joonho Lee, Takahiro Miki, Moju Zhao, Farbod Farshidian, Kei Okada, Masayuki Inaba, and Marco Hutter. Circus anymal: A quadruped learning dexterous manipulation with its limbs. In *ICRA 2021*, 2021.

[178] Hendrik Kolvenbach, Christian Bärtschi, Lorenz Wellhausen, Ruben Grandia, and Marco Hutter. Haptic inspection of planetary soils with legged robots. *IEEE RA-L*, 2019.

[179] T Turner Topping, Gavin Kenneally, and Daniel E Koditschek. Quasi-static and dynamic mismatch for door opening and stair climbing with a legged robot. In *ICRA 2017*, 2017.

[180] Sunwoo Kim, Maks Sorokin, Jehee Lee, and Sehoon Ha. Human Motion Control of Quadrupedal Robots using Deep Reinforcement Learning. In *Proceedings of Robotics:*

Science and Systems, New York City, NY, USA, 2022. doi: 10.15607/RSS.2022.XVIII.021.

[181] Naoki Yokoyama, Alexander William Clegg, Eric Undersander, Sehoon Ha, Dhruv Batra, and Akshara Rai. Adaptive skill coordination for robotic mobile manipulation. *arXiv preprint arXiv:2304.00410*, 2023.

[182] Jia-Ruei Chiu, Jean-Pierre Sleiman, Mayank Mittal, Farbod Farshidian, and Marco Hutter. A collision-free mpc for whole-body dynamic locomotion and manipulation. In *ICRA 2022*, 2022.

[183] Jean-Pierre Sleiman, Farbod Farshidian, and Marco Hutter. Versatile multicontact planning and control for legged loco-manipulation. *Science Robotics*, 8(81):eadg5014, 2023.

[184] Mayank Mittal, David Hoeller, Farbod Farshidian, Marco Hutter, and Animesh Garg. Articulated object interaction in unknown scenes with whole-body mobile manipulation. In *IROS 2022*, 2022.

[185] Xiangyu Chu, Shengzhi Wang, Minjian Feng, Jiaxi Zheng, Yuxuan Zhao, Jing Huang, and KW Samuel Au. Model-free large-scale cloth spreading with mobile manipulation: Initial feasibility study. In *2023 IEEE 19th International Conference on Automation Science and Engineering (CASE)*, pages 1–6. IEEE, 2023.

[186] Henrique Ferrolho, Vladimir Ivan, Wolfgang Merkt, Ioannis Havoutis, and Sethu Vijayakumar. Roloma: Robust loco-manipulation for quadruped robots with arms. *Autonomous Robots*, 47(8):1463–1481, 2023.

[187] Flavio De Vincenti and Stelian Coros. Centralized model predictive control for collaborative loco-manipulation. *RSS 2023*, 2023.

[188] Jeeseop Kim and Kaveh Akbari Hamed. Cooperative locomotion via supervisory predictive control and distributed nonlinear controllers. *Journal of Dynamic Systems, Measurement, and Control*, 2022.

[189] Artem Lykov, Mikhail Litvinov, Mikhail Konenkov, Rinat Prochii, Nikita Burtsev, Ali Alridha Abdulkarim, Artem Bazhenov, Vladimir Berman, and Dzmitry Tsetserukou. Cognitivedog: Large multimodal model based system to translate vision and language into action of quadruped robot. *arXiv preprint arXiv:2401.09388*, 2024.

[190] Arne Roennau, Georg Heppner, Michal Nowicki, and Rüdiger Dillmann. Lauron v: A versatile six-legged walking robot with advanced maneuverability. In *2014 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, pages 82–87. IEEE, 2014.

[191] G Heppner, T Buettner, A Roennau, and R Dillmann. Versatile-high power gripper for a six legged walking robot. In *Mobile Service Robotics*, pages 461–468. World Scientific, 2014.

[192] G Heppner, A Roennau, J Oberländer, S Klemm, and R Dillmann. Laurope-six legged walking robot for planetary exploration participating in the spacebot cup. *WS on Advanced Space Technologies for Robotics and Automation*, 2(13):69–76, 2015.

[193] Julian Whitman, Shuang Su, Stelian Coros, Alex Ansari, and Howie Choset. Generating gaits for simultaneous locomotion and manipulation. In *IROS 2017*, 2017.

[194] Wiebke Brinkmann, Alexander Dettmann, Leon Cedric Danter, Christopher Schulz, Tobias Stark, and Adrian Brandt. Enhancement of the six-legged robot mantis for assembly and construction tasks in lunar mission scenarios within a multi-robot team. In *Proceedings: International Symposium on Artificial Intelligence, Robotics and Automation in Space*, 2020.

[195] Wenhao Yu, Nimrod Gileadi, Chuyuan Fu, Sean Kirmani, Kuang-Huei Lee, Montse Gonzalez Arenas, Hao-Tien Lewis Chiang, Tom Erez, Leonard Hasenclever, Jan Humplik, et al. Language to rewards for robotic skill synthesis. *arXiv preprint arXiv:2306.08647*, 2023.

[196] Anthony Brohan, Yevgen Chebotar, Chelsea Finn, Karol Hausman, Alexander Herzog, Daniel Ho, Julian Ibarz, Alex Irpan, Eric Jang, Ryan Julian, et al. Do as i can, not as i say: Grounding language in robotic affordances. In *Conference on robot learning*, pages 287–318. PMLR, 2023.

[197] Yen-Jen Wang, Bike Zhang, Jianyu Chen, and Koushil Sreenath. Prompt a robot to walk with large language models. *arXiv preprint arXiv:2309.09969*, 2023.

[198] Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Eureka: Human-level reward design via coding large language models. *arXiv preprint arXiv:2310.12931*, 2023.

[199] Abhishek Padalkar, Acorn Pooley, Ajinkya Jain, Alex Bewley, Alex Herzog, Alex Irpan, Alexander Khazatsky, Anant Rai, Anikait Singh, Anthony Brohan, et al. Open x-embodiment: Robotic learning datasets and rt-x models. *arXiv preprint arXiv:2310.08864*, 2023.

[200] Tairan He, Zhengyi Luo, Wenli Xiao, Chong Zhang, Kris Kitani, Changliu Liu, and Guanya Shi. Learning human-to-humanoid real-time whole-body teleoperation. *arXiv preprint arXiv:2403.04436*, 2024.

[201] Fourier Intelligence. Fourier humanoid robot: Advanced robotics technology. Product Brochure, 2023. URL https://fourierintelligence.com/gr1/.