Perceptive Hexapod Legged Locomotion for Climbing Joist Environments

Zixian Zang
Avideh Zakhor

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2023-89
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2023/EECS-2023-89.html

May 10, 2023

Perceptive Hexapod Legged Locomotion for Climbing Joist Environments

by

Zixian Zang

A thesis submitted in partial satisfaction of the

requirements for the degree of

Master of Science

in

Electrical Engineering and Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Avideh Zakhor, Chair
Professor Sergey Levine

Spring 2023

Perceptive Hexapod Legged Locomotion for Climbing Joist Environments

by Zixian Zang

Research Project

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

Approval for the Report and Comprehensive Examination:

Committee:

[illegible]

Professor Avideh Zakhor
Research Advisor

5/10/23

(Date)

* * * * * * *

[illegible]

Professor Sergey Levine
Second Reader

5/10/23

(Date)

Perceptive Hexapod Legged Locomotion for Climbing Joist Environments

Abstract

Perceptive Hexapod Legged Locomotion for Climbing Joist Environments

by

Zixian Zang

Master of Science in Electrical Engineering and Computer Science

University of California, Berkeley

Professor Avideh Zakhor, Chair

Attics are one of the largest sources of energy loss in residential homes, but they are uncomfortable and dangerous for human workers to conduct air sealing and insulation. Hexapod robots are potentially suitable for carrying out those tasks in tight attic spaces since they are stable, compact, and lightweight. For hexapods to succeed in these tasks, they must be able to navigate inside tight attic spaces of single-family residential homes in the U.S., which typically contain rows of approximately 6 or 8-inch tall joists placed 16 inches apart from each other. Climbing over such obstacles is challenging for autonomous robotics systems. In this work, we develop a perceptive walking model for legged hexapods that can traverse over terrain with random joist structures using egocentric vision. Our method can be used on low-cost hardware not requiring real-time joint state feedback. We train our model in a teacher-student fashion in 2 phases: In phase 1, we use reinforcement learning with access to privileged information such as local elevation maps and joint feedback. In phase 2, we use supervised learning to distill the model into one with access to only onboard observations, consisting of egocentric depth images and robot orientation captured by a tracking camera. We demonstrate zero-shot sim-to-real transfer on a Hiwonder[15] SpiderPi robot, equipped with a depth camera onboard, climbing over joist courses we construct to simulate the environment in the field. Our proposed method achieves nearly 100% success rate climbing over the test courses, significantly outperforming the model without perception and the controller provided by the manufacturer.

Moreover, we develop an interactive visualization tool to improve the measurement and verification process of building retrofit. With drone-captured RGB and Infrared (IR) images of facades, our tool lets the user compare IR images capturing a specified location on a facade before and after the retrofit. We also design a pipeline to stitch IR images capturing different parts of a facade into a panorama, making it easier to find the metric locations of studs for envelop retrofit projects.

Contents

Contents **i**

List of Figures **ii**

List of Tables **v**

1 Introduction **1**
- 1.1 Learning to Control Physical Systems . . . 3
- 1.2 Related Work . . . 4

2 Perceptive Hexapod Legged Locomotion **6**
- 2.1 Overall Approach . . . 6
- 2.2 Method . . . 8

3 Experiments and Results **19**
- 3.1 Test Setup . . . 19
- 3.2 Performance on Actual Hardware . . . 19
- 3.3 Ablation Studies . . . 21

4 Building Retrofit Analysis and Verification **25**
- 4.1 Associating RGB and IR Information . . . 25
- 4.2 Stitching IR Images . . . 28

5 Conclusions and Future Work **37**

Bibliography **38**

A Custom URDF Robot Representation **41**

List of Figures

1.1 Photo of an unfurnished attic with joists. 1
1.2 (a): The SpiderPi hexapod robot used in our work, with Intel L515 and T265 mounted onboard. (b): URDF model of SpiderPi robot we create for training. . 2
1.3 Time-lapse photo capturing the SpiderPi robot climbing over 8 joists using the method proposed in our work. 3

2.1 The robot has six legs with three degrees of freedom per leg, each actuated by a LX-224HV three-port bus servo. Each actuator can actuate to ± 120 degrees from its depicted position and provides a maximum of 1.96 Nm of torque and 5.24 rad/s of angular velocity. 7
2.2 High-level overview of our method. The model directly predicts the joint target positions q_t. In simulation, the actions are used as position target for a tuned proportional-derivative (PD) controller that outputs torques τ_t which are applied to simulated actuators in the simulation. This figure is adapted from [19]. 7
2.3 Simulated terrains for training. (a): easy rough terrain. (b): difficult joist terrain. The first half of phase 1 training is conducted on the easy rough terrain shown in (a), and the second half is conducted on the difficult joist terrain shown in (b). 9
2.4 Structure of our model. The state s_t is the concatenation of proprioception information x_t and perception feature f_t. s_t is encoded by observation encoder E_o into latent vector z_t, then passed to the policy π to predict the joint angles a_t. . 9
2.5 Learning curves for each reward term during phase 1 reinforcement learning training. The first 8,000 of phase 1 training are conducted on the easier rough terrain shown in Figure 2.3(a) and the second 8,000 on the harder joist terrain shown in Figure 2.3(b). (a): Linear velocity in global x. (b): Linear velocity in body y. (c): Global heading. (d): Angular velocity: yaw. (e): Ground impact. (f): Collision penalty. (g): Action rate. (h): Action magnitude. (i): Torques. (j): Joint acceleration. (k): Joint limit penalty. (l): End effector height. 12
2.6 Learning curves for reward achieved by the agent during phase 1 and phase 2 training. The first 8,000 of phase 1 training are conducted on the easier rough terrain shown in Figure 2.3(a) and the second 8,000 on the harder joist terrain shown in Figure 2.3(b). Phase 2 training is only conducted on the harder joist terrain shown in Figure 2.3(b) for 2,000 iterations. 13

2.7 Screenshots from simulation rollout videos. The yellow dots rendered in (a) and (b) are the locations where elevation map values are sampled. (a): A simulated rollout of the phase 1 model only trained for 8,000 iterations on rough terrain shown in Figure 2.3(a) can be seen in this video. (b): A simulated rollout of phase 1 model trained both on rough terrain shown in Figure 2.3(a) and joist terrain shown in Figure 2.3(b) can be seen in this video. (c): A simulated rollout of phase 2 model trained on joist terrain shown in Figure 2.3(b) can be seen in this video. 14
2.8 Diagram for phase 2 training. We train the student encoder and policy to recover the same z_t and a_t as the teacher, using more limited observations. The student policy is initialized with the weights from the trained teacher policy. This figure is adapted from [19]. 16
2.9 Techniques used in depth image processing. (a): Mask used to avoid the legs being captured. (b): Overlay of the leg mask on an egocentric depth image with the leg in view, with the highlighted area at the bottom corners masked out. Part of the robot leg captured by depth camera is visible at the lower right corner. (c): An example depth image captured in simulation. (d): Depth image in (c) with pixels randomly masked out according to depth value. (e): Distribution of depth value captured by L515 mounted on the robot. (f): $p_{invalid}$ as function of d with $k = 3m$ and $d_{bound} = 2m$ 18

3.1 Demonstrations of our perceptive model, the blind version of our model, and the Hiwonder controller deployed in the real world on joist courses. (a),(b): Examples of the Spiderpi hexapod climbing an 8-joist course using our perceptive model can be viewed in this video and this video. (c): A video of the Spiderpi hexapod climbing the joist course using our perceptive model at a 45-degree angle can be viewed in this video. (d): A video of the Spiderpi hexapod climbing the joist course using a blind version of our method can be viewed in this video. (e): A video of the Spiderpi hexapod climbing the joist course using the manufacturer's controller can be viewed in this video. 22
3.2 (a): Phase 1 reward curves of models with and without access to joint angle feedback. (b): Phase 2 reward curves, learning from phase 1 models with and without access to joint angle feedback. 24

4.1 Screenshot of our IV tool: (a): Panorama image of a façade of the trailer in the DoE weatherization lab in Santa Fe, NM created by auto-stitching 10 RGB drone images; the user clicks anywhere on the panorama, and the footprint of the two IR images captured in two different flights, for that point are automatically displayed, for pre and post insulation retrofit. (b): Same as (a) but for a multifamily building in central CA, which underwent air sealing retrofit. 26
4.2 Illustration of projecting drone camera's optical center on the facade by finding the intersection of the optical axis of the drone camera with the facade plane. . 27

4.3 Distorted original captured thermal images. 28
4.4 Illustration of image distortion mapping. (a): Forward mapping maps from an undistorted space to a distorted space. (b): backward mapping maps from a distorted space to an undistorted space. 29
4.5 Applying forward model to line-pattern with different values of K_2 30
4.6 Thermal images with calibrated distortion. 31
4.7 To stitch two thermal images in (a) and (b) together, pairs of corresponding points are selected between images, then homography matrix H can be solved. (c): Result of stitching (a) and (b). (d): Fully stitched panorama produced by iterating the stitching process. 34
4.8 (a): Given a pre-built 3D model of the building, the exact aspect ratio R according to the dimensions can be extracted. (b): Final head-on thermal facade panorama. 35
4.9 Screenshot of Rhino architecture tool. Users can click on studs on the thermal facade to find the metric locations of the studs with respect to the origin (0,0,0) which in this case is the lower right corner of the facade. 36

List of Tables

2.1 Constituent elements of the state. Uniform random noises with ranges indicated above are added to each state element before being used as model input. * denotes privileged observations that are only available during phase 1 training. 9
2.2 Constituent elements of the multi-objective reward during phase 1 training, with corresponding weights for each of the reward terms. This table is adapted from [19]. 15

3.1 When deployed on the 4-joist course, the average number of joists that each model successfully climbs over, and the average time for models to climb over each joist, evaluated on the proposed perceptive model, the blind model, and the controller provided by Hiwonder. The Hiwonder Controller cannot climb over any joist, therefore does not have a valid measurement of time per joist. 20
3.2 When deployed on the 8-joist course, the average number of joists that each model successfully climbs over, and the average time for models to climb over each joist, evaluated on the proposed perceptive model, the blind model, and the controller provided by Hiwonder. The Hiwonder Controller cannot climb over any joist, therefore does not have a valid measurement of time per joist. 20
3.3 When deployed on the 4-joist course, the average number of joists that each model successfully climbs over, and the average time for models to climb over each joist, evaluated on the perceptive models trained with (a) both leg masking and random invalid depth pixels, (b) without leg masking, (c) without random invalid depth pixels and (d) without both. The models trained without leg masking do not successfully climb over any joists, therefore do not have a valid measurement of time per joist. 21
3.4 When deployed on the 8-joist course, the average number of joists that each model successfully climbs over, and the average time for models to climb over each joist, evaluated on the perceptive models trained with (a) both leg masking and random invalid depth pixels, (b) without leg masking, (c) without random invalid depth pixels and (d) without both. The models trained without leg masking do not successfully climb over any joists, therefore do not have a valid measurement of time per joist. 23

Acknowledgments

None of this work could be done without the support of my advisor, Professor Zakhor. Thank you for giving me the opportunity to own this fascinating research problem and for your tremendous guidance in the past two years. I am also indebted to Wenhao and Tingnan from Google Robotics for sharing their expertise and knowledge about machine learning and legged robotics.

I would also like to express my gratitude to my family and friends for their unwavering support, love, and encouragement throughout my education. Their encouragement and motivation kept me going even during the most challenging times.

Thank you UC Berkeley for providing me with a rich academic community where I met some of my best friends and found my passion for automation and robotics. This has been an unforgettable journey that I will forever cherish.

Chapter 1

Introduction

Attics are one of the largest sources of energy loss in residential homes. A typical unfinished attic is shown in Figure 1.1. A substantial reduction in home energy costs and its carbon footprint can be achieved through attic air sealing and insulation. However, attics are tight spaces and difficult, uncomfortable, and potentially dangerous for workers to carry out these tasks. For example, since attics typically consist of multiple rows of joist structures, a human worker could easily fall through the attic floor and get seriously injured if he or she steps on the sheetrock between two joists by mistake. Workers may also need protective suits to protect themselves from toxic substances during vacuuming and spray foaming, which are common tasks for air sealing and insulation.

Lightweight legged robots are ideal platforms for navigating inside attics to carry out various tasks such as air-sealing and vacuuming. To do so, we need to enable legged robots to traverse in environments with dense and high joists. Since most single-family residential home attics in the U.S. contain approximately 6 or 8-inch high joists that are 16 inches apart,

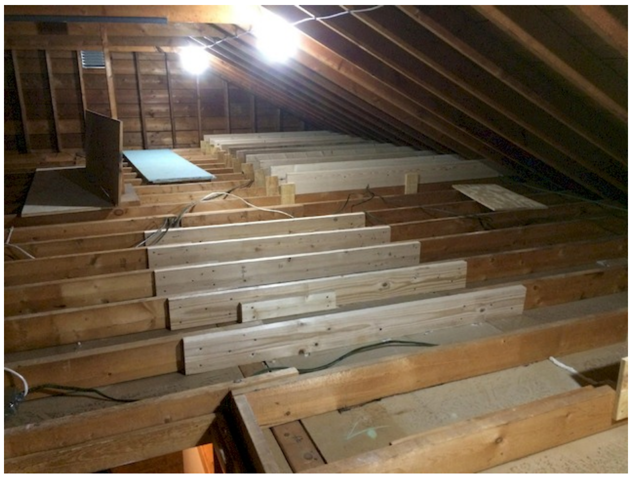


Figure 1.1: Photo of an unfurnished attic with joists.

it is important for legged robots to be able to autonomously climb such joists inside attics.

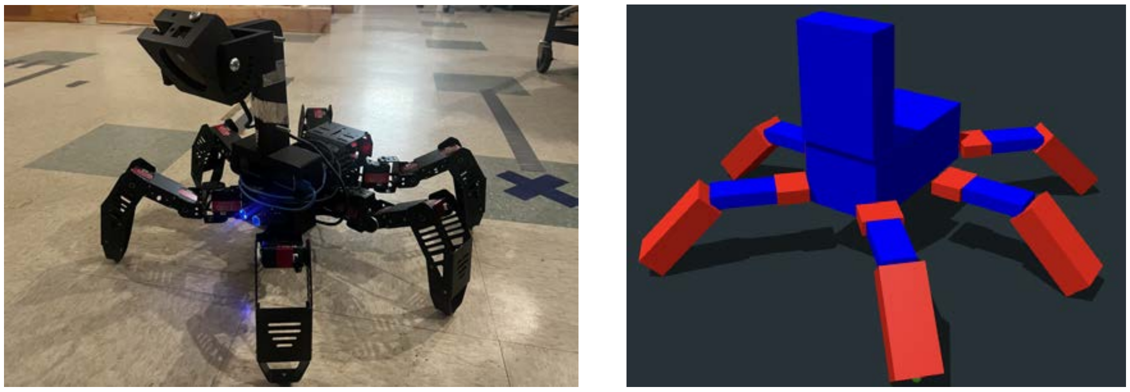

(a) Physical Robot (b) URDF

Figure 1.2: (a): The SpiderPi hexapod robot used in our work, with Intel L515 and T265 mounted onboard. (b): URDF model of SpiderPi robot we create for training.

There has been a significant amount of recent work on quadrupeds and bipedal robot locomotion. However, in this work, we focus on using hexapods for joist climbing for two main reasons. First, hexapod robots are by design more stable and lightweight than quadrupeds and humanoids of similar size. Secondly, bipedal or quadruped robots are significantly higher than hexapods from the ground and are therefore less suitable for traversing within tight spaces such as the corners of attics.

To facilitate the practical usage of robots in the retrofit business, it is important for legged locomotion controllers to work with low-cost hardware. However, most existing legged locomotion systems require high-end robots capable of real-time sensing of joint states, which could ultimately result in expensive hardware. For example, model predictive control methods such as [5] require powerful computation resources and real-time joint feedback from expensive robot platforms and often compromise real-time performance when incorporating more complex dynamics. Data-driven methods such as [1] can work with limited computation resources and are robust to a variety of perception failures but need fast joint state feedback. Many low-cost robots are not equipped with powerful onboard computation or real-time hardware feedback such as joint torque and angle that are accessible on more expensive platforms. At the same time, humans without leg sensing feedback, when equipped with prosthetics, can walk and even participate in competitive sports with only egocentric visual perception and a sense of body orientation [27].

In this thesis, we propose an end-to-end learning-based perceptive controller for low-cost, sub-thousand-dollar hexapods to autonomously climb over joists and demonstrate zero-shot sim-to-real transfer on joist terrain with configurations similar to the ones in typical attics. Our robot is a $600 SpiderPi robot manufactured by Hiwonder, shown in Figure 1.2(a), equipped with an Intel L515 depth camera and a T265 tracking camera with a customized

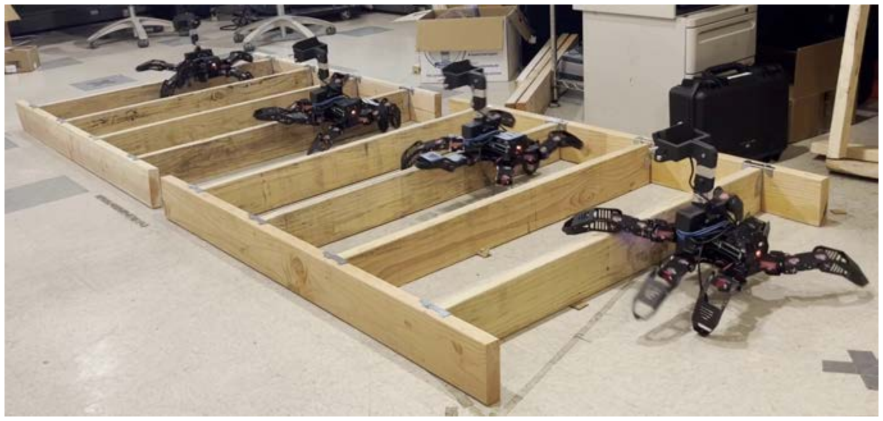


Figure 1.3: Time-lapse photo capturing the SpiderPi robot climbing over 8 joists using the method proposed in our work.

camera mount. The robot does not provide real-time joint feedback. The entire system including the sensors costs less than \$1500. We propose a two-stage teacher-student training procedure to learn models that can work without real-time joint feedback: the first stage involves Reinforcement Learning (RL) with access to privileged observations and the second stage uses supervised learning to distill the model using only onboard observations including body orientation and egocentric depth images. Since optimal joist climbing motions are fundamentally different from walking, we train our controllers without human-defined prior gait knowledge, guiding the models to explore task-appropriate motions. We compare our method to a few baselines including end-to-end training of a blind model and a hand-designed controller. Our proposed method significantly outperforms both by achieving a much higher success rate climbing over joist courses and taking much less time to climb over each joist. In addition, we study the design of the privileged information during phase 1 training and find that it is critical to include information such as joint feedback even though it is not available onboard. A time-lapse photo of using our proposed method to control the robot climbing over 8 joists with 6-inch height and 16-inch spacing is presented in Figure 1.3.

1.1 Learning to Control Physical Systems

In recent years, the development of Artificial Neural Networks as complex function approximators makes it the perfect choice for replacing previously hand-coded control methods for physical robotics systems. However, training neural networks to handle robotics control problems is not as trivial as other tasks. Typically, the training process of neural networks consists of millions of steps for the models to successfully converge. In the case of legged robotics, it is not practical to have the robot platform attempt to walk thousands or millions of times, which will not only take an excessive amount of time but also can potentially damage the robot hardware, especially when the model's actions are not reasonable at the beginning of the training process. Therefore, accurate simulation engines are developed to enable parallel data gathering for hundreds or thousands of training environments, in which realistic models of the robots are used to learn optimal motions for the given task.

At the same time, the task of controlling a physical system involves sequential decision-making. Many other machine learning problems have clearly defined correct answers, such as class labels in classification tasks. However, to learn a model that controls a physical system, it is basically impossible for humans to come up with a sequence of actuator actions that yield the best outcome. Take the task of legged locomotion of hexapods on challenging terrains as an example; there are many possible gaits combinations to choose from, and it is hard to judge whether one sequence of motions is better or worse than another sequence. Moreover, only learning one reasonable sequence of action is far from sufficient for the physical systems to handle various environment settings - the environment is modified in different ways as the system uses different control actions. On the other hand, a classification network only needs to consider the input it is given at the moment and does not need to worry about the next input being changed according to its reaction to the current input.

Because of the limitations listed above and beyond, the problem of learning a controller for physical systems is often approached as reinforcement learning tasks, a method for learning from action sequences. In reinforcement learning, the objective is no longer trying to match ground truth, but to maximize the reward, the signal that indicates the fitness of the chosen action, accumulated across timesteps.

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

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

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

The objective of reinforcement learning is to find a policy $\pi_\theta\left(a_t \mid s_t\right)$ that maximizes the cumulative reward:

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

1.2 Related Work

Hexapods Legged Locomotion on Challenging Terrain

Hexapod robots traversing challenging terrain have been studied for decades. In 1990, [21] proposed a set of rules to control and navigate a hexapod robot. In [12], Frankhauser *et al.* developed a ROS library for navigating a hexapod or quadruped using elevation map. Shortly after that, Frankhauser *et al.* demonstrated solid results by deploying their method on a quadruped in [10]. [5] proposes a method to control hexapods on unstructured terrain using combination of exteroceptive and proprioceptive terrain characterization. Faigl *et al.* [9] use only position feedback to let a hexapod walk on uneven terrain. [23] proposes a teleoperator for hexapods under environmental perturbations. [8] improves passability of gait selection on sparse foothold environments using a Monte Carlo tree search algorithm. [32] proposes an adaptive hexapod controller using a force sensing pipeline. However, these methods are only capable of managing explicitly modeled uncertainties and therefore are not suitable for tackling uncommon terrains. To overcome this, data-driven methods are proposed by Li *et al.* in [24] and Qiu *et al.* in [28]. [3] enables hexapods to traverse rough terrain in simulation using binary contact sensors on feet endpoints. [22] uses RL to train a central pattern generator with spiking neural networks. However, there is no existing work showing a hexapod robot traversing challenging terrains using a learned perceptive controller. As a continuation of the blind controller that Kawawa-Beaudan proposed earlier in [19], our work adopts end-to-end learning without any human-defined gait prior knowledge, allowing the agent to be deployed on inexpensive hexapods and to perform reliably on joist structures.

Legged Locomotion with Perception

For a legged robot to efficiently climb high obstacles such as joists, it must be equipped with a perception system to identify and step over the obstacles. To incorporate perception in legged locomotion, most previous model-based methods first convert raw input from depth cameras or LiDARs into other forms such as local elevation maps; they then use elevation maps to evaluate local areas for foothold feasibility [29, 17, 26, 11, 2, 20, 18, 30] or traversability [6, 13]. These methods assume perfect conversion from raw sensor data to a desired form such as an elevation map. Conversion to elevation map is computation intensive, requiring expensive and heavy compute engines that must be carried by the robot. Recent data-driven approaches map from perception input directly to actions. Specifically, [31] proposes a method for quadrupeds to walk on complex terrain by predicting actions directly from depth captured onboard and executing actions with a model predictive controller. [16] trains an end-to-end controller that maps from depth images to motor commands and demonstrates the robot traversing various terrains in simulation. [25] proposes a method to produce high-level commands for jumping over gaps. In [1], Agarwal *et al.* use egocentric depth to directly predict joint angles. However, to work on actual robots, these methods need high-frequency state feedback provided by costly hardware. In contrast, in this work, we incorporate visual perception on low-cost hardware without access to real-time joint states.

Chapter 2

Perceptive Hexapod Legged Locomotion

2.1 Overall Approach

In this section, we present our overall approach, including the choice of hardware, training strategy, and ways to retrieve the robot model used for training. We use a \$600 SpiderPi hexapod robot, shown in Figure 1.2(a), which has a standing height of approximately 15cm without our custom-designed camera mount, 30cm with the mount, and weights around 2.4 kilograms. The onboard computer on our robot is a Raspberry Pi 4B with 4GB of RAM, 4 CPU cores, and 1.5 GHz clock rate. As shown in Figure 2.1, the robot has six legs with three degrees of freedom per leg, each actuated by a LX-224HV three-port bus servo. Each actuator can actuate to $\pm$ 120 degrees from its depicted position and provides a maximum of 1.96 Nm of torque and 5.24 rad/s of angular velocity.

Even though we train fully in simulation, there are no models to simulate the motors reliably since our robot uses actuators with unknown dynamics. Unlike actuators in high-end robots that allow the reading and writing of Proportional Derivative (PD) controller gains, PD control is performed at the circuit level on LX-224HV by modulating motor voltage with unknown gains that cannot be modified. As a result, we follow the approach in [19] and use position control, which is physically less accurate. As depicted in Figure 2.2, during training, the model outputs desired joint angles a_t which are converted to joint torques τ_t and fed into motors in the simulation engine. When deployed on hardware, the joint angles are directly used as angle targets for servo motors on the actual robot with unknown dynamics.

We train our model using a teacher-student training scheme, similar to the one used in [19], with two phases to target zero-shot sim-to-real transfer. The first phase is trained with access to privileged observations using reinforcement learning and the second phase is trained with only access to observations onboard. There are two reasons for having two training phases: First, depth rendering with current simulation significantly slows down simulation speed. Specifically, reinforcement learning takes a large number of samples to

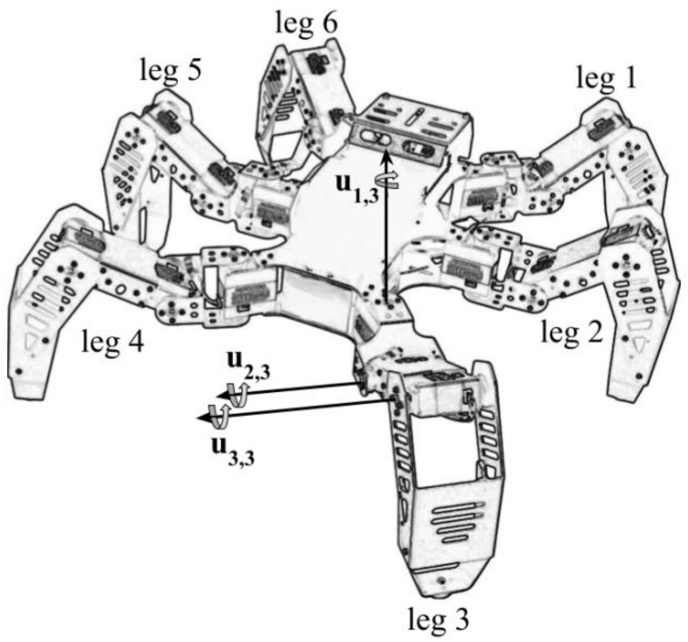


Figure 2.1: The robot has six legs with three degrees of freedom per leg, each actuated by a LX-224HV three-port bus servo. Each actuator can actuate to ± 120 degrees from its depicted position and provides a maximum of 1.96 Nm of torque and 5.24 rad/s of angular velocity.

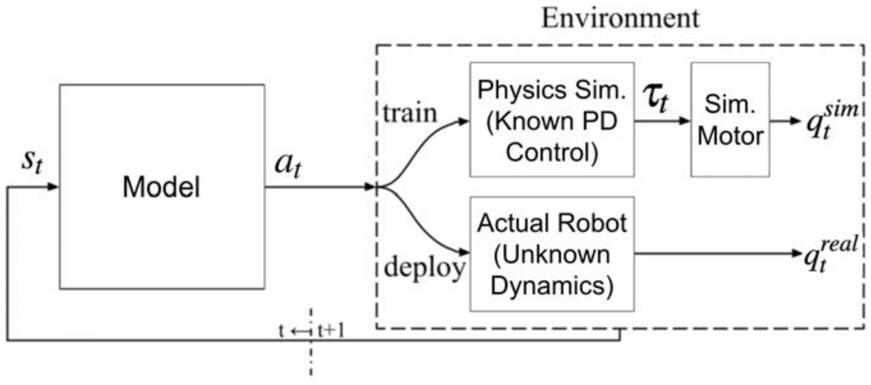


Figure 2.2: High-level overview of our method. The model directly predicts the joint target positions q_t. In simulation, the actions are used as position target for a tuned proportional-derivative (PD) controller that outputs torques τ_t which are applied to simulated actuators in the simulation. This figure is adapted from [19].

converge, so it is impractical to directly train perceptive models with depth sensing using reinforcement learning. Secondly, unlike actuators used in high-end robots with real-time joint feedback including position, velocity, and even torque, our servo positions are written to and read from joints through a shared serial port. This prohibits parallel access, so all joints are read and written to in serial, with wait times in between to clear the bus. One loop reading the position of all the joints can take multiple seconds, making it impossible to use them as real-time inputs to a controller. Similar to the case in [19], training without joint position feedback from scratch yields suboptimal performance, so we have to include joint feedback in phase 1 and distill it in phase 2. As a result, the model trained in phase 1 has access to joint angle feedback, body orientation, and elevation maps sampled around the robot. In phase 2, a student model with depth images as perception input learns from the model trained with elevation maps in phase 1 and takes body orientation as the only source of proprioception feedback. As we show later in Section 3.3, models trained with no joint feedback in phase 1 achieve much lower reward than our proposed method.

Since there is no available 3D model of our robot, we measure the length and weight of each leg segment and approximate the robot segments using boxes to build a URDF model for training in simulation, as shown in Figure 1.2(b) and included in Appendix A.

2.2 Method

In this section, we describe our training method in more detail. We follow a teacher-student training scheme with two phases, reinforcement and supervised learning. Since we directly deploy models trained from phase 2 on actual hardware, we incorporate several techniques to ensure the performance of zero-shot sim-to-real transfer.

Training Setup

We conduct training in IsaacGym simulator with the legged_gym library. It is difficult to train without motion priors on terrain with high and dense joist structures completely from scratch. Therefore, we first train on an easier rough terrain with randomly generated low joist courses, shown in Figure 2.3(a), arranged to give rise to a learning curriculum similar to [4]. Then we fine-tune the model on a terrain that closely mimics the joists pattern in the field, as shown in Figure 2.3(b).

We use the same basic setup as in [4] to realize parallelism for our training. Elements of the state are listed in Table 2.1 with corresponding units. Uniform random noises with listed ranges are added to the elements. As shown in Figure 2.4, at each time step, the perception input p_t, either an elevation map or a depth image, is first encoded by the perception head E_p into perception feature f_t:

$$f_t = E_p(p_t) \tag{2.1}$$

State Element	Constituents	Units	Noise Range
Base attitude	o_t={roll, pitch, yaw}	Radians	$[-0.1, 0.1]$
Joint positions*	$q_t \in \mathbb{R}^{18}$	Radians	$[-0.01, 0.01]$
Previous actions	$a_{t-1} \in \mathbb{R}^{18}$	Radians	$[0, 0]$
Elevation Map*	$h_t \in \mathbb{R}^{221}$	Meters	$[0.1, 0.1]$
Depth Image	$d_t \in \mathbb{R}^{320*240}$	Meters	$[0.1, 0.1]$

Table 2.1: Constituent elements of the state. Uniform random noises with ranges indicated above are added to each state element before being used as model input. * denotes privileged observations that are only available during phase 1 training.

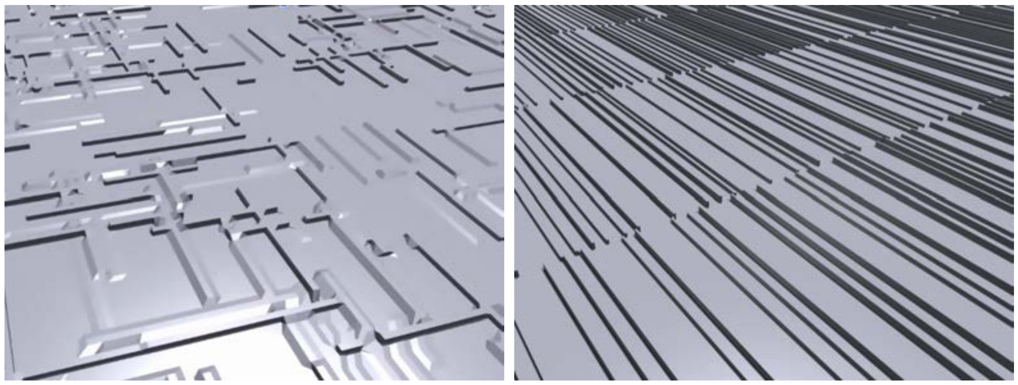

(a) Rough terrain (b) Joist terrain

Figure 2.3: Simulated terrains for training. (a): easy rough terrain. (b): difficult joist terrain. The first half of phase 1 training is conducted on the easy rough terrain shown in (a), and the second half is conducted on the difficult joist terrain shown in (b).

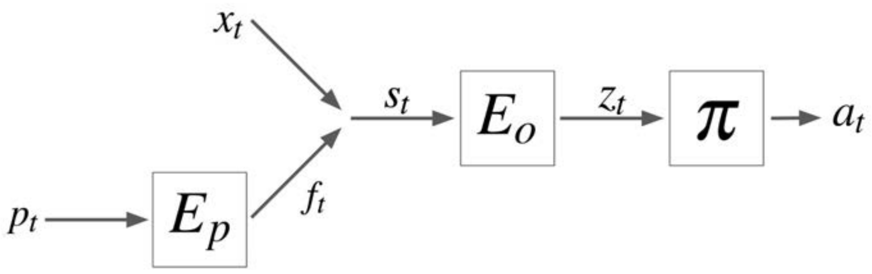

$$\begin{tikzcd} & x_t \arrow[dr] & & & & \\ & & \cdot \arrow[r, "s_t"] & \boxed{E_o} \arrow[r, "z_t"] & \boxed{\pi} \arrow[r] & a_t \\ p_t \arrow[r] & \boxed{E_p} \arrow[ur, "f_t"'] & & & & \end{tikzcd}$$

Figure 2.4: Structure of our model. The state s_t is the concatenation of proprioception information x_t and perception feature f_t. s_t is encoded by observation encoder E_o into latent vector z_t, then passed to the policy π to predict the joint angles a_t.

Then the state s_t, resulting from the concatenation of proprioception information x_t and perception feature f_t, is fed into the observation encoder E_o to produce a latent vector z_t:

$$s_t = [x_t, f_t] \tag{2.2}$$

$$z_t = E_o(s_t) \tag{2.3}$$

Then the encoded feature z_t is passed to the policy π that predicts the joint angles a_t:

$$a_t = \pi(z_t) \tag{2.4}$$

Our observation encoder E_o is a 1-layer LSTM with hidden size 64 and a CNN or MLP as perception head to extract features from perception input. The recurrent nature of the observation encoder enables the model to remember past observations and incorporate salient information into z_t, making it possible to traverse challenging terrain using proprioception and learn to overcome obstacles captured by egocentric vision. Our policy is an actor-critic with both actor and critic as a 3-layer MLP with hidden layers of size 128, 64, and 32. Exponential linear unit activation is used between the layers. In the simulation, our model makes predictions at 25Hz frequency. A simulation step refers to each time IsaacGym renders a new view, and each simulation time step is set to 0.005 second. Therefore, the simulation steps 8 times every time the model makes a prediction. Even though the lightweight nature of our robot is useful in our application, its low inertial links require small time steps in simulation and hence long training times. A physics engine step refers to each time IsaacGym's physics engine simulates physics and updates the state of each simulated body. By default, the physics simulation engine has the same frequency as the simulation rendering update frequency. However, we have found it necessary to increase the number of steps of the physics engine per simulation rendering step from 1 to 4 to simulate physics accurately, even though it increases training time linearly.

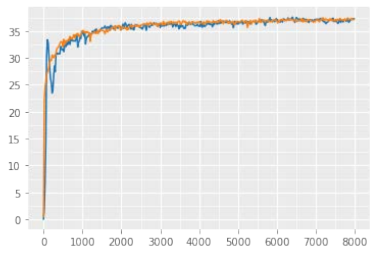
Phase 1: Reinforcement Learning

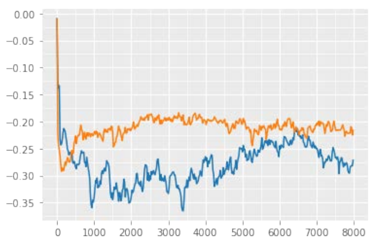
In phase 1, we train the model using the open-source implementation of Proximal Policy Optimization from [4]. The elevation map sampled around the robot is used as perception input in phase 1. As mentioned in [1], a critical step is to find the optimal elevation map configuration such that the elevation map does not contain information that cannot be inferred from the depth images used in phase 2 training. To achieve this, we start with a rough estimation of the ground area that the depth images on the actual robot capture. With the hardware configuration shown in Figure 1.2, we consider a scenario with the robot standing still and the camera pointing at 30 degrees downward with FOV provided by L515 intrinsic information, and compute the projection area of the camera's view based on geometry. We then search elevation map configurations close to the rough projection area and use the one with the lowest phase 2 loss. This approach has been shown in [1] to result in optimal performance. The final elevation map is sampled from a 0.6m $\times$ 0.8m grid that is 0.3m in front of the robot with a sample spacing of 0.05m, resulting in a 13 $\times$ 17 elevation map.

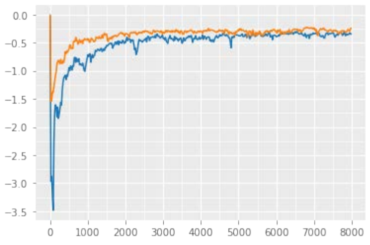
Our reward is the weighted sum of the elements in Table 2.2, adapted from [19]. Linear velocity in the x direction of the global coordinate v_x is first clipped to range $[-0.4, 0.4]$ meters to encourage the robot to traverse forward and overcome joist obstacles. Linear velocity in the y direction of the local coordinate v_y is penalized to avoid the robot's sideway movements. Magnitudes of the robot's heading in the global coordinate θ and the robot's yaw angular velocity ω_z are penalized to avoid the robot from turning away from the forward direction. $\|f_t - f_{t-1}\|^2$ is penalized to avoid severe impact exerted on the robot's end effectors, where f_t stands for the force applied to the end effectors at time t. Collision is penalized and computed as a binary flag in the form $\mathbb{1}\{\text{coxa, femur, or base contacting terrain}\}$ to indicate contact between chosen parts on the robot and the terrain surface. To avoid drastic and exaggerated actions, we penalize action rate $\|a_t - a_{t-1}\|^2$ and action magnitude $\|a_t\|^2$, where a_t stands for the model's action prediction at time t. $\|\tau\|^2$ is penalized to avoid damage to our hardware, where τ is the sum of torque at all the joints. We penalize measured joint acceleration $\hat{\ddot{q}}^2 = \frac{\dot{q}_t - \dot{q}_{t-1}}{\Delta t}^2$, where q_t is the measured joint position at time t. Furthermore, to avoid hardware damage caused by hitting joint limit, we penalize the term $\text{clip}\,(q_{\min} - q_t, \max = 0) + \text{clip}\,(q_t - q_{\max}, \min = 0)$, where $q_{\min}$ = -1rad and $q_{\max}$ = 1rad are the lower and upper soft angle limits. We also penalize $\|z_{end\_effector}\|$ to prevent the robot from raising its legs too high up in the air. A one-layer MLP is used in phase 1 as perception head E_p of the observation encoder to encode elevation map h_t. During phase 1, proprioception information x_t consists of base orientation o_t, joint angles q_t, and the previous action a_{t-1}. Therefore, the state s_t has the following form:

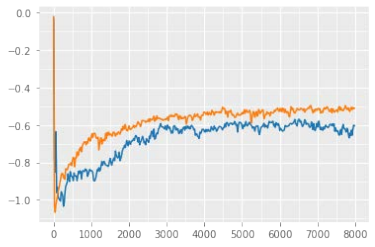
$$s_t = [o_t, q_t, a_{t-1}, E_p(h_t)] \tag{2.5}$$

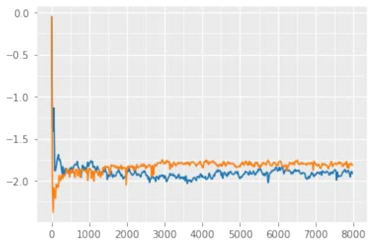
Models are first trained with the easier terrain shown in Figure 2.3(a) and then tuned on the terrain with high and dense joists shown in Figure 2.3(b), which is more difficult but more closely mimics the environment in the field. On each terrain, we use 4,000 robots simulated in parallel and train for 8,000 iterations, with 24 timesteps for all agents per iteration. 96,000 transition samples are created during each iteration and are split into 4 mini-batches with sizes of 24,000 each. To ensure the robustness and stability of the learned gait, we randomize the friction coefficient of each robot in the range [0.5, 1.25], and apply pushes in random directions to each robot's base mass every 8 seconds. Learning curves for each reward term during phase 1 reinforcement learning are shown in Figure 2.5. Our reward formulation successfully encourages the robot to walk forward in the global x direction, as reflected by the curve of the main reward term $\text{clip}\,(v_x, -0.4, 0.4)$ shown in Figure 2.5(a). As seen from the plots of the penalizing terms, such as Figure 2.5(c), Figure 2.5(d), and Figure 2.5(g), almost all the terms that penalize drastic actions suffer a fall during the first few iterations as the robot tries to find a way to maneuver forward. In the later iterations, almost all the penalizing terms steadily rise, indicating that our reward formulation enables the robot to find the most energy-efficient and safe motions. In Figure 2.6, we present the learning curves for the mean total reward during the first and second half of phase 1 training. As seen from the curves in Figure 2.6, the model fine-tuned on joist terrain can reach a similar reward as

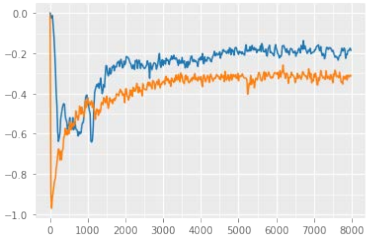

(a) $\mathrm{clip}\,(v_x, -0.4, 0.4)$

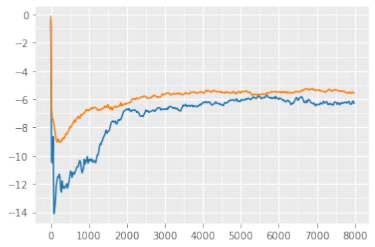

(b) v_y^2

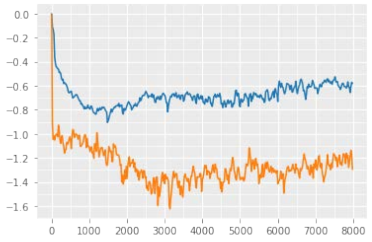

(c) θ^2

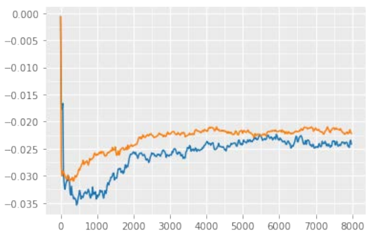

(d) ω_z^2

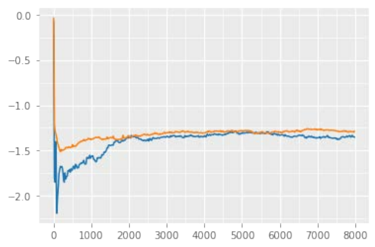

(e) $\|f_t - f_{t-1}\|^2$

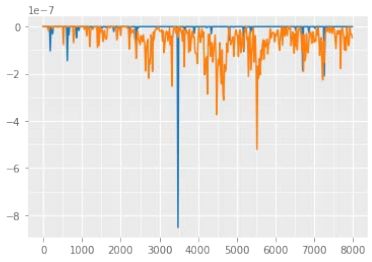

(f) $\mathbb{1}\{$coxa, femur, or base contacting terrain$\}$


(g) $\|a_t - a_{t-1}\|^2$

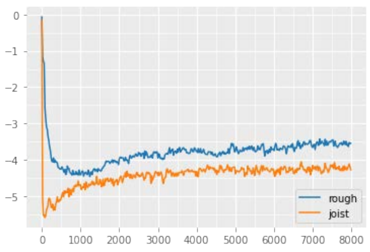

(h) $\|a_t\|^2$


(i) $\|\tau\|^2$


(j) $\hat{\ddot{q}}^2 = {\frac{\dot{q}_t - \dot{q}_{t-1}}{\Delta t}}^2$



(k) $\mathrm{clip}\,(q_{\min} - q_t, \max = 0) +$ $\mathrm{clip}\,(q_t - q_{\max}, \min = 0)$



(l) $\|z_{end\_effector}\|$

Figure 2.5: Learning curves for each reward term during phase 1 reinforcement learning training. The first 8,000 of phase 1 training are conducted on the easier rough terrain shown in Figure 2.3(a) and the second 8,000 on the harder joist terrain shown in Figure 2.3(b). (a): Linear velocity in global x. (b): Linear velocity in body y. (c): Global heading. (d): Angular velocity: yaw. (e): Ground impact. (f): Collision penalty. (g): Action rate. (h): Action magnitude. (i): Torques. (j): Joint acceleration. (k): Joint limit penalty. (l): End effector height.

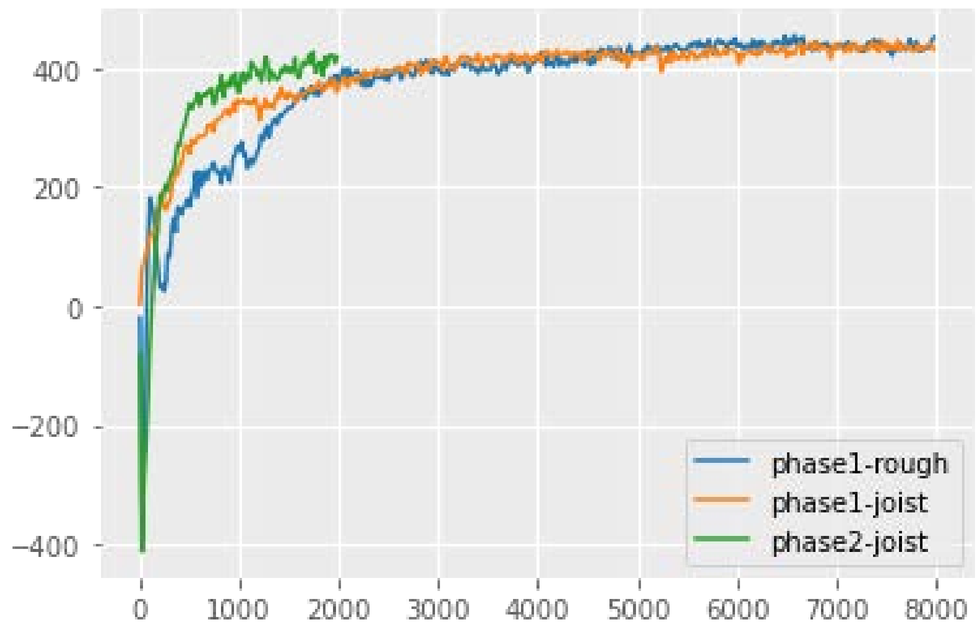


Figure 2.6: Learning curves for reward achieved by the agent during phase 1 and phase 2 training. The first 8,000 of phase 1 training are conducted on the easier rough terrain shown in Figure 2.3(a) and the second 8,000 on the harder joist terrain shown in Figure 2.3(b). Phase 2 training is only conducted on the harder joist terrain shown in Figure 2.3(b) for 2,000 iterations.

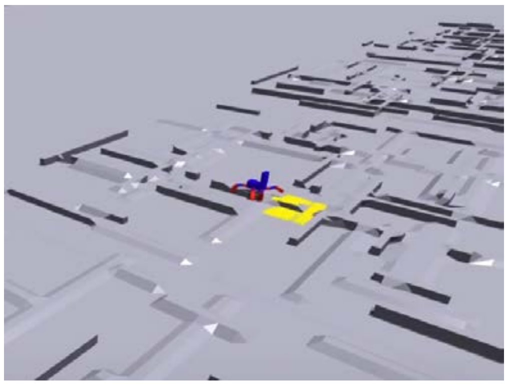
the model trained on easier rough terrain. A simulation rollout video of the model trained with the first half of phase 1 on terrain shown in Figure 2.3(a) is shown in Figure 2.7(a), and a simulation rollout video of the model trained fully with phase 1 is shown in Figure 2.7(b). As shown in the video, the model trained fully with phase 1 can traverse joist terrain with similar configurations as typical home attics.

Phase 2: Supervised Learning for Sim-to-Real

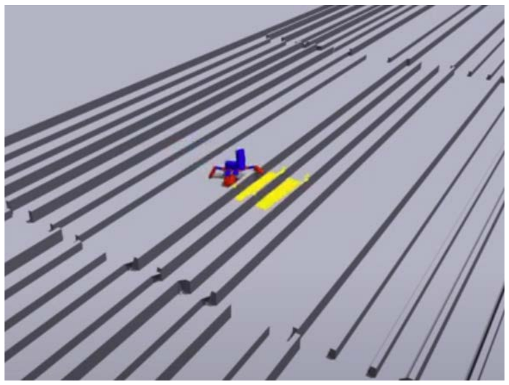
In phase 2, we use supervised learning to train the model with access to only proprioception and perception available onboard, supervised by a teacher model trained in phase 1. For the student model during phase 2 training, proprioception information $\hat{x}_t$ consists of base attitude o_t, previous action a_{t-1}, and egocentric depth image d_t as perception input. The state $\hat{s}_t$ for the student model then has the following form:

$$\hat{s}_t = [o_t, a_{t-1}, E_p(d_t)] \tag{2.6}$$

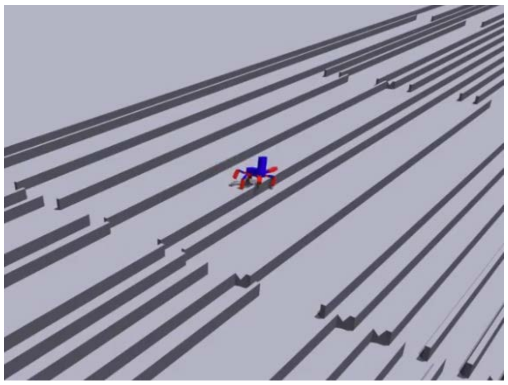
Only the observation encoder E_p and its perception head are trained from scratch, with the policy being initialized with the trained policy weight from phase 1. Figure 2.8 shows the



(a)



(b)



(c)

Figure 2.7: Screenshots from simulation rollout videos. The yellow dots rendered in (a) and (b) are the locations where elevation map values are sampled. (a): A simulated rollout of the phase 1 model only trained for 8,000 iterations on rough terrain shown in Figure 2.3(a) can be seen in this video. (b): A simulated rollout of phase 1 model trained both on rough terrain shown in Figure 2.3(a) and joist terrain shown in Figure 2.3(b) can be seen in this video. (c): A simulated rollout of phase 2 model trained on joist terrain shown in Figure 2.3(b) can be seen in this video.

Reward Term	Expression	Weight
Linear velocity in global x	$\text{clip}\left(v_x, \min = -0.4, \max = 0.4\right)$	1e2
Linear velocity in body y	v_y^2	-1e1
Global heading	θ^2	-3e1
Angular velocity: yaw	ω_z^2	-1e0
Ground impact	$\lVert f_t - f_{t-1} \rVert^2$	-1e-1
Collision penalty	$\mathbb{1}\{\text{coxa, femur, or base contacting terrain}\}$	-1e0
Action rate	$\lVert a_t - a_{t-1} \rVert^2$	-5e-1
Action magnitude	$\lVert a_t \rVert^2$	-1e-2
Torques	$\lVert \tau \rVert^2$	-1e-3
Joint acceleration	$\hat{\ddot{q}}^2 = \frac{\dot{q}_t - \dot{q}_{t-1}}{\Delta t}^2$	-1e-5
Joint limit penalty	$\text{clip}\left(q_{\min} - q_t, \max = 0\right) + \text{clip}\left(q_t - q_{\max}, \min = 0\right)$	-1e0
End effector height	$\lVert z_{end\_effector} \rVert$	-1e1

Table 2.2: Constituent elements of the multi-objective reward during phase 1 training, with corresponding weights for each of the reward terms. This table is adapted from [19].

phase 2 training process. We supervise the student encoder's $\hat{z}_t$ with the teacher encoder's z_t, and student policy's $\hat{a}_t$ with the teacher policy's a_t. In other words, we train the student encoder and policy to recover the same z_t and a_t, using the more limited observation set, by optimizing the following loss, as demonstrated in Figure 2.8:

$$L_{\text{encoding}} = \text{MSE}\left(z_t, \hat{z}_t\right) \tag{2.7}$$

$$L_{\text{action}} = \text{MSE}\left(a_t, \hat{a}_t\right) \tag{2.8}$$

$$L_{\text{phase-ii}} = L_{\text{encoding}} + L_{\text{action}} \tag{2.9}$$

We also need to consider the sim-to-real gap of depth perception in order to successfully deploy the model trained in phase 2 on hardware. Depth images retrieved from IsaacGym are nearly perfectly simulated, which is not the case for our actual depth sensor. Since our robot is close to the ground and the distance between the depth camera and the ground is not too large, in order to obtain the best depth quality for near objects, we operate the onboard depth camera in low ambient light mode. By doing so however, far-away pixels often end up with invalid measurements. Moreover, since we do not have an exact 3D model of the robot and we approximate segments of the robot with boxes when the robot legs are within the view of the depth camera, there is a domain gap between the simulation and actual depth images.

To address the issue of front legs being inadvertently captured in the actual captured depth images, we mask out all the areas that could possibly be occupied by the legs with all physically possible joint angle combinations. The masked-out areas are presented as the

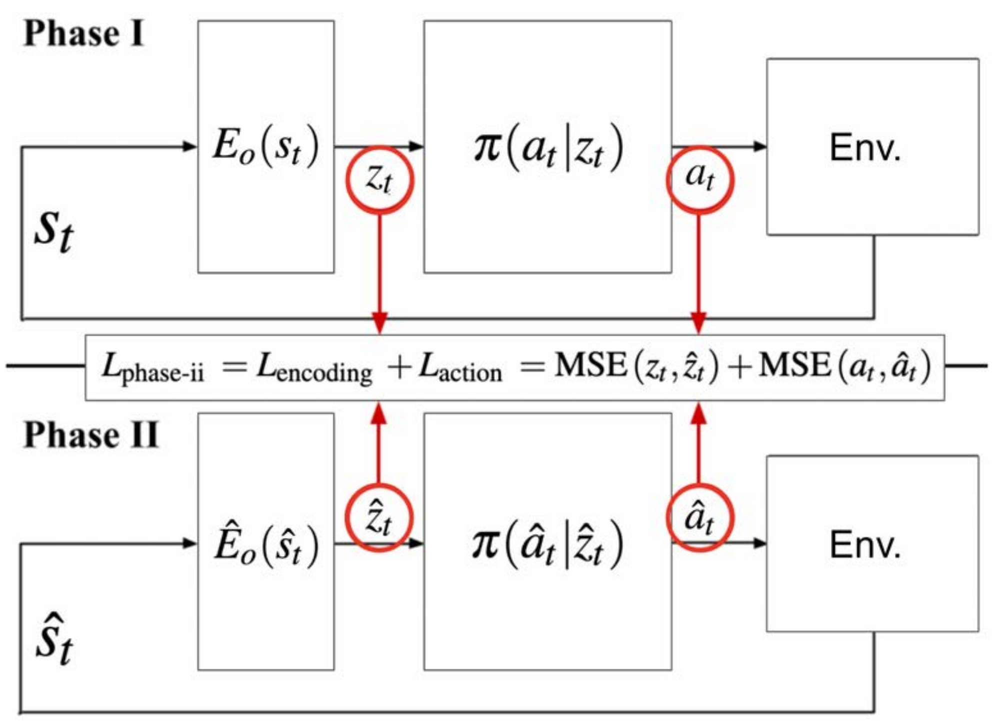

Phase I
$E_o(s_t)$
z_t
$\pi(a_t|z_t)$
a_t
Env.
s_t
$L_{\text{phase-ii}} = L_{\text{encoding}} + L_{\text{action}} = \text{MSE}(z_t, \hat{z}_t) + \text{MSE}(a_t, \hat{a}_t)$
Phase II
$\hat{E}_o(\hat{s}_t)$
$\hat{z}_t$
$\pi(\hat{a}_t|\hat{z}_t)$
$\hat{a}_t$
Env.
$\hat{s}_t$

Figure 2.8: Diagram for phase 2 training. We train the student encoder and policy to recover the same z_t and a_t as the teacher, using more limited observations. The student policy is initialized with the weights from the trained teacher policy. This figure is adapted from [19].

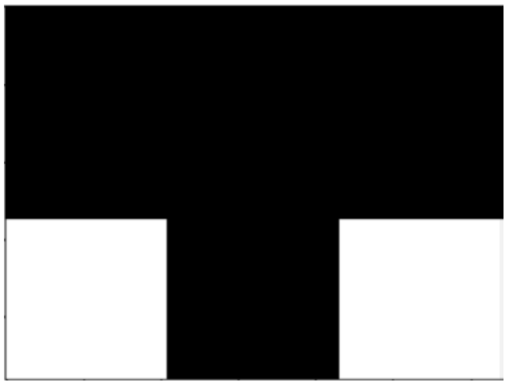
white areas in Figure 2.9(a). An overlay image of the leg mask on an egocentric depth image with the leg in view is shown in Figure 2.9(b), with part of the robot leg captured by the depth camera is shown in the lower right corner and the highlighted area at the bottom corners being masked out.

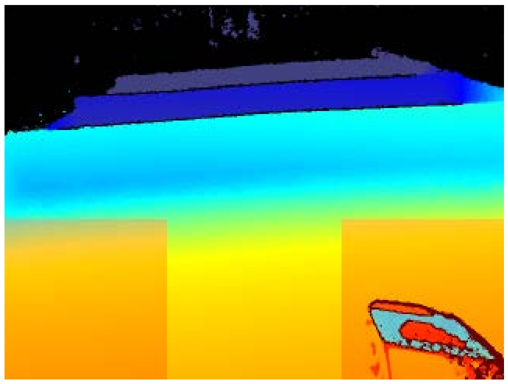
To deal with the pixels in actual depth images that are invalid due to range, we randomly mask out pixels in simulated depth images as follows: the probability of each pixel with simulated depth measurement d meters being masked out is:

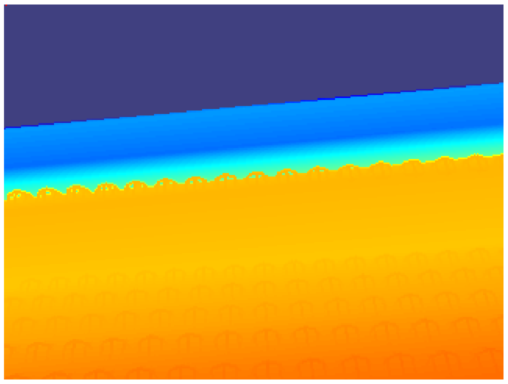
$$p_{invalid}(d) = \frac{exp(min(d_{bound}, d))}{exp(k)} \tag{2.10}$$

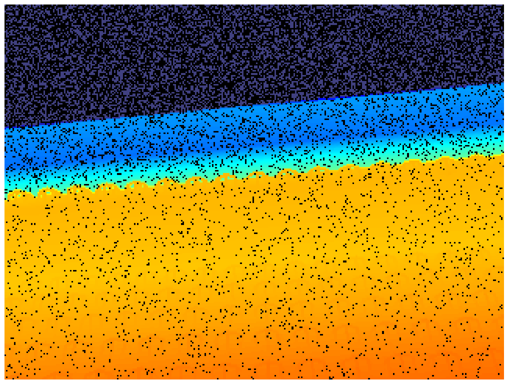
In the above expression, k is set to be the approximate upper bound of depth values for the majority of pixels on actual hardware. However, depth pixels in simulation can have valid measurement even when being far away from the camera, so we first clip the depth measurement d with an upper bound d_{bound}, otherwise those pixels would all be masked out since they all have $p_{invalid} > 1$. The value of d_{bound} is tuned to yield the best zero-shot sim-to-real performance. An example depth image captured in simulation is visualized in Figure 2.9(c). The image with pixels being randomly masked out according to their depth values is shown in 2.9(d), with black pixels corresponding to the pixels that are masked out. To determine an appropriate range for k, we walk the robot around with the depth camera mounted using the built-in Hiwonder controller and plot the distribution of valid pixel values in multiple randomly selected captured depth images, shown in Figure 2.9(e). As seen from the distribution in Figure 2.9(e), most valid pixels are within 3 meters. Therefore, we choose $k = 3m$ and $d_{bound} = 2m$ to result in the best performance. A curve showing $p_{invalid}$ as function of d with $k = 3m$ and $d_{bound} = 2m$ is shown in Figure 2.9(f).

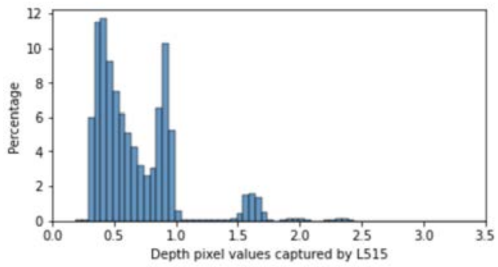
Phase 2 training is only conducted on the more difficult joist terrain environment to emphasize the trained model's ability to adapt to the environment in the field. Due to the compute intensity of depth retrieval in simulation, we only simulate 100 robots in parallel. We train the model for 2,000 iterations using Adam optimizer and learning rate of 5e-4. A simulation rollout of the model trained in phase 2 traversing joist terrain using depth perception is shown in Figure 2.7(c).


(a) Leg Mask

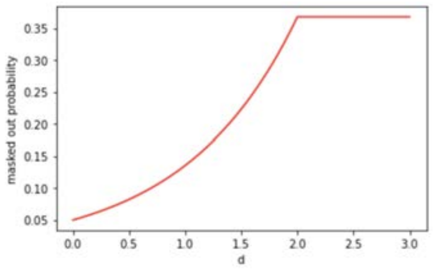

(b) Overlay of leg mask


(c) Example depth image


(d) Depth-based mask



(e) Depth Distribution on L515



(f) $p_{invalid}$ as function of d

Figure 2.9: Techniques used in depth image processing. (a): Mask used to avoid the legs being captured. (b): Overlay of the leg mask on an egocentric depth image with the leg in view, with the highlighted area at the bottom corners masked out. Part of the robot leg captured by depth camera is visible at the lower right corner. (c): An example depth image captured in simulation. (d): Depth image in (c) with pixels randomly masked out according to depth value. (e): Distribution of depth value captured by L515 mounted on the robot. (f): $p_{invalid}$ as function of d with $k = 3m$ and $d_{bound} = 2m$

Chapter 3

Experiments and Results

To evaluate the performance of our method, we deploy our proposed method, a blind version of our method, and the controller provided by the manufacturer, on a SpiderPi robot and have it climb over joist courses we have constructed to mimic attic environments. We also conduct ablation studies to justify the techniques we have introduced to ensure sim-to-real transfer.

3.1 Test Setup

To evaluate the robot's ability to traverse joists with a pattern similar to the ones in the field, we construct a testing ground using wood joists with 2-inch thickness and 6-inch height. We construct 4-joist and 8-joist courses to test the models. Joists are arranged in a pattern with 16-inch center-to-center gap. The 8-joist course is shown in Figure 1.1. We test the walking controllers by having the robot traverse in the direction that is perpendicular to the joists for multiple trials. The time limit to complete each trial is set to be 30 seconds for the 4-joist course and 60 seconds for the 8-joist course. The performance of each method is evaluated using two metrics: the average number of joists the robot successfully climbs over in each trial, and the average time it takes for the robot to climb over each joist. Success is defined as the entire body and all the legs of the robot passing over the joist. The time interval between passing the previous joist to passing the current joist is taken as the time to climb the current joist. We only interrupt a trial if the robot falls over and cannot recover on its own.

3.2 Performance on Actual Hardware

For comparison, we also train a blind model and use it without the depth camera. The training process of the blind model is exactly the same as the perceptive policy, except that it does not have access to any perception information, without a perception head on the observation encoder. The blind model also takes body orientation as proprioception, so only

Method	Average #Joists	Average Time / Joist (s)
Perceptive	**4**	**3.7**
Blind	1.8	3.8
HiWonder Controller	0	N/A

Table 3.1: When deployed on the 4-joist course, the average number of joists that each model successfully climbs over, and the average time for models to climb over each joist, evaluated on the proposed perceptive model, the blind model, and the controller provided by Hiwonder. The Hiwonder Controller cannot climb over any joist, therefore does not have a valid measurement of time per joist.

Method	Average #Joists	Average Time / Joist (s)
Perceptive	**7.8**	**3.4**
Blind	1.9	4.1
HiWonder Controller	0	N/A

Table 3.2: When deployed on the 8-joist course, the average number of joists that each model successfully climbs over, and the average time for models to climb over each joist, evaluated on the proposed perceptive model, the blind model, and the controller provided by Hiwonder. The Hiwonder Controller cannot climb over any joist, therefore does not have a valid measurement of time per joist.

the T265 tracking camera is attached to the robot when using the blind model. We compare the performance of our perceptive model to both the blind model and the walking controller provided by HiWonder. All models are used on a low-cost Raspberry Pi 4B with 4GB of RAM. Experimental results are presented in Tables 3.1 and 3.2. When tested on the 4-joist course over 10 trails, the perceptive model deployed directly on hardware achieves a 100% success rate climbing over all joists within the given time, and uses around 3.7 seconds to climb over each joist. When tested on the 8-joist course, the perceptive model climbs over 7.8 joists per trail on average, and uses around 3.4 seconds to climb over each joist. In the only failed trial, the robot makes it very close to the end. It loses balance as it pitches up and falls over. Examples of the robot traversing over the 8-joist course is shown in Figure 3.1(a) and Figure 3.1(b) with video links attached in the figure caption.

The blind model manages to climb over 1.8 joists in each trial on average and uses slightly more time than the perceptive model to climb over each joist. However, the blind model learns gaits with high impact and with its front legs raised high in the air even when there are no joists present in front of the robot. This is because it does not have access to any source of perception to interpret the appropriate action magnitude. The controller from

Method	Average #Joists	Average Time / Joist (s)
Perceptive	**4**	**3.7**
w/o leg mask	0	N/A
w/o random invalid	1.3	6
w/o both	0	N/A

Table 3.3: When deployed on the 4-joist course, the average number of joists that each model successfully climbs over, and the average time for models to climb over each joist, evaluated on the perceptive models trained with (a) both leg masking and random invalid depth pixels, (b) without leg masking, (c) without random invalid depth pixels and (d) without both. The models trained without leg masking do not successfully climb over any joists, therefore do not have a valid measurement of time per joist.

Hiwonder is not capable of climbing over any joists over all the trials.

To further test the robustness of our proposed method, we test the robot approaching and climbing over the 4-joist course at an angle. The robot is set to start with a 45-degree angle and is tested for 10 trials. We find that it successfully climbs over the joist course in 9 of the 10 trials, as shown in Figure 3.1(c).

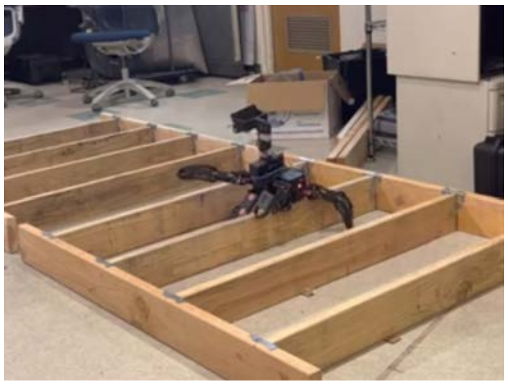
We also test the proposed perceptive model with a 12-inch high joist course to determine its behavior when encountering obstacles that are physically too high to climb over. We find that the robot stops moving once it gets close to the joists. This is expected since the depth camera view is fully blocked by the high obstacle, and such a scenario is seldom encountered during training.

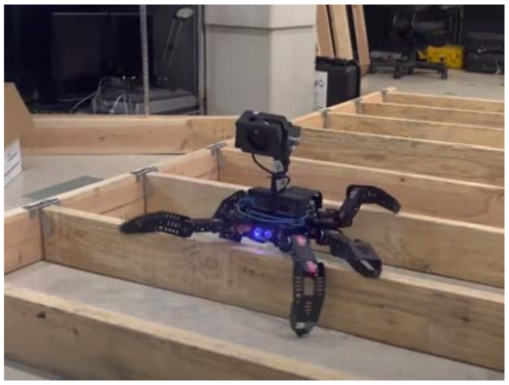
3.3 Ablation Studies

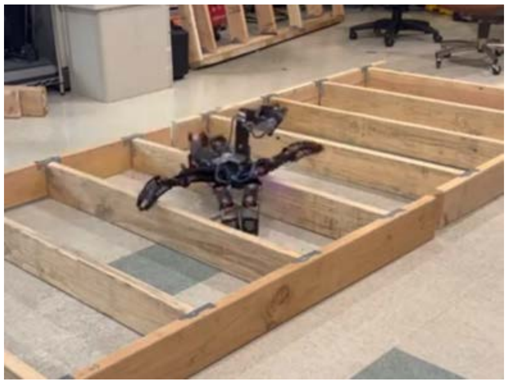
Depth Processing Techniques

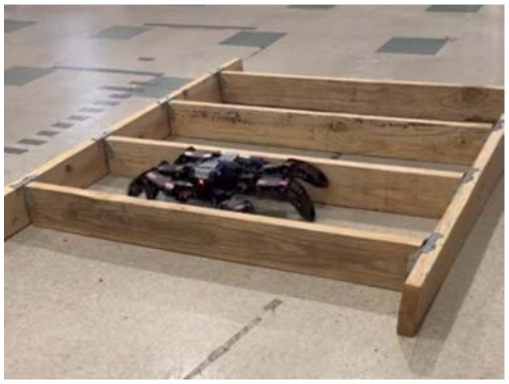
In previous sections, we presented two depth image preprocessing techniques to enable zero-shot sim-to-real transfer. To justify the effects of these techniques, we conduct ablation studies and evaluate the models with similar test settings as above. We compare the performance of the perceptive model trained with depth images preprocessed using (a) both leg masking and random invalid pixel, (b) without leg masking, (c) without random invalid pixels, and (d) without both. When deploying a model trained without leg masking on hardware, we do not apply leg masking to depth images to match the depth processing used during training.

Experimental results are presented in Tables 3.3 and 3.4. We observe that models trained and deployed without leg masking would behave abnormally as soon as legs are captured by depth camera during climbing, and are not capable of climbing over any joists. This is


(a)


(b)


(c)


(d)

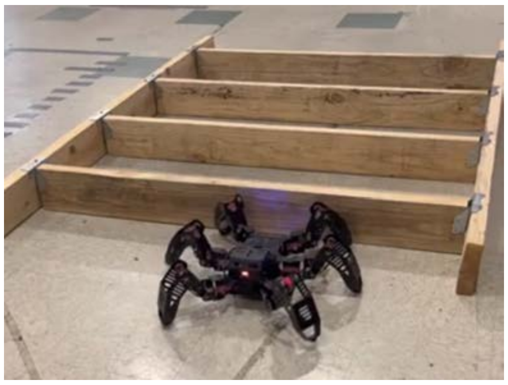

(e)

Figure 3.1: Demonstrations of our perceptive model, the blind version of our model, and the Hiwonder controller deployed in the real world on joist courses. (a),(b): Examples of the Spiderpi hexapod climbing an 8-joist course using our perceptive model can be viewed in this video and this video. (c): A video of the Spiderpi hexapod climbing the joist course using our perceptive model at a 45-degree angle can be viewed in this video. (d): A video of the Spiderpi hexapod climbing the joist course using a blind version of our method can be viewed in this video. (e): A video of the Spiderpi hexapod climbing the joist course using the manufacturer's controller can be viewed in this video.

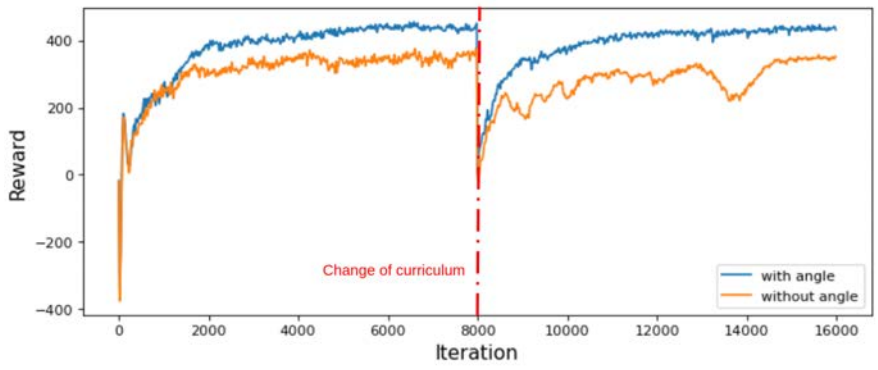
Method	Average #Joists	Average Time / Joist (s)
Perceptive	**7.8**	**3.4**
w/o leg mask	0	N/A
w/o random invalid	1.5	6.1
w/o both	0	N/A

Table 3.4: When deployed on the 8-joist course, the average number of joists that each model successfully climbs over, and the average time for models to climb over each joist, evaluated on the perceptive models trained with (a) both leg masking and random invalid depth pixels, (b) without leg masking, (c) without random invalid depth pixels and (d) without both. The models trained without leg masking do not successfully climb over any joists, therefore do not have a valid measurement of time per joist.

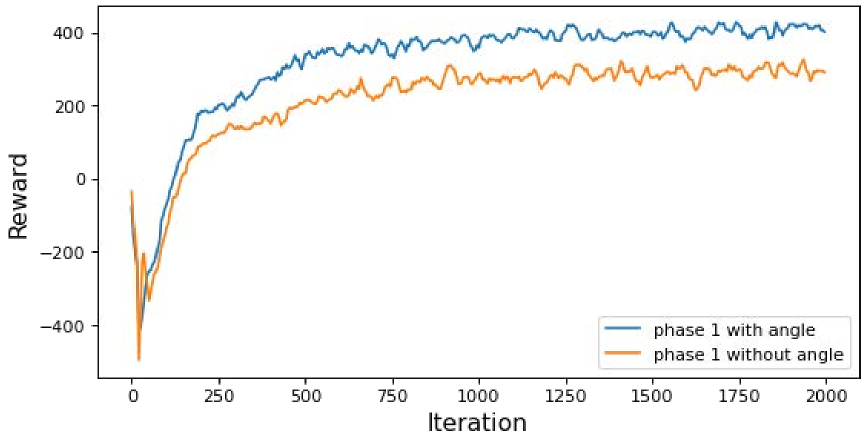
likely due to the domain gap presented between the depth measurements of the actual robot legs and the legs in the approximate robot URDF model. The model using only leg masking but not random invalid pixels makes some progress and climbs over joists, but gets stuck or falls over quite often. When tested on either the 4-joist or the 8-joist course, it is able to climb around 1.3 joists before it fails and takes around 6 seconds on average to successfully climb over each joist. Also, it never finishes a single trail. This proves the effectiveness of our proposed depth-based pixel masking technique to reduce the domain gap between depth images in simulation and captured by the actual L515 sensor.

Phase 1 without Joint Feedback

One of the main reasons we divide training into 2 phases is to learn an effective expert policy with access to privileged state feedback in phase 1. To prove the necessity of having access to joint state feedback in phase 1, we compare phases 1 and 2 training performances between models with and without access to joint angle feedback, as shown in Figure 3.2. As seen in Figure 3.2(a), the drop at the middle is due to curriculum change - the first 8000 iterations of phase 1 training are conducted on lower joist terrain shown in Figure 2.3(a) and the second 8,000 iterations on higher joist terrain in Figure 2.3(b). During phase 1, the model with access to joint angle feedback significantly outperforms the one without access to joint angle feedback. During phase 2 training, the student model without access to the joint angle is able to reach a similar reward as the teacher model with access to the joint angle. Student model learning from the teacher model without access to joint angle yields relatively low reward. This demonstrates the necessity to include critical privileged joint angle feedback in phase 1 training in order to optimize the model's zero-shot sim-to-real performance.



(a) Phase 1 reward with vs without joint angle



(b) Phase 2 reward

Figure 3.2: (a): Phase 1 reward curves of models with and without access to joint angle feedback. (b): Phase 2 reward curves, learning from phase 1 models with and without access to joint angle feedback.

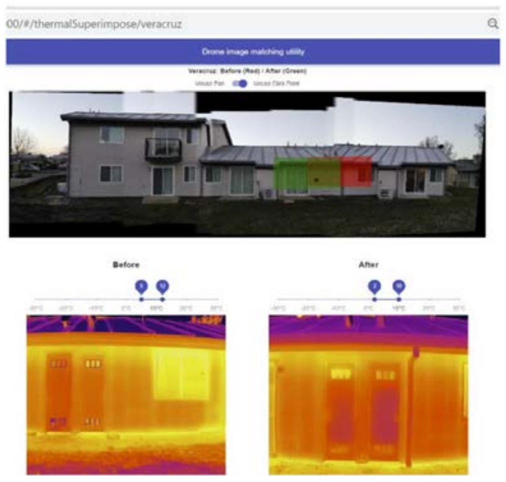
Chapter 4

Building Retrofit Analysis and Verification

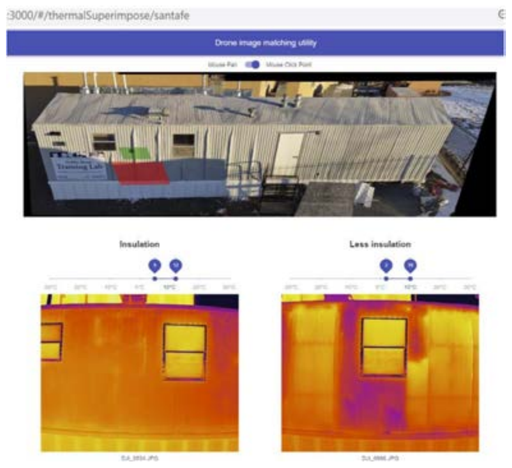
This chapter presents an interactive visualization tool we developed to analyze and verify building retrofit using drone-captured RGB and Infrared (IR) images. Infrared thermography is typically used to detect the three main culprits in envelope retrofit: thermal bridging, air infiltration, and insulation deficiencies. This is typically done by having a human operator use a thermal camera to simultaneously capture IR images and visually detect anomalies real time as s/he walks around the perimeter of the building. There are multiple problems associated with the current practice: first for buildings with more than one story, the resulting IR images cannot properly capture higher stories. Second, the approach is error prone in that the human operator can potentially miss thermal anomalies due to real time cognitive overload on the spot. Third, the process is laborious and takes a long time since each time a defect is detected, the operator has to stop and mark the location of the defect in construction drawings or use imprecise or excessive words to identify the exact location of the defect. This is further complicated when comparing pictures of the same spot on the building pre and post retrofit.

4.1 Associating RGB and IR Information

Interactive Visualization (IV) tool is useful for visualizing the difference between structures before and after the retrofit. For larger buildings, ideally one would like to capture the thermal pictures up close in order to capture the details of thermal anomalies. However, the close-up pictures necessarily make it difficult to determine which part of the building each picture came from, due to lack of context. We developed an IV user interface to solve this problem, as shown in Figure 4.1. Our IV tool first uses open-source software, such as [14], that finds homography matrix H between images automatically using RANSAC to stitch multiple RGB pictures, in order to generate a “synthetic” panorama of the facade. Every thermal image has attached metadata including global positioning system (GPS) data


00/#/thermalSuperimpose/veracruz
Drone image matching utility
Veracruz: Before (Red) / After (Green)
Before
After

(a)


3000/#/thermalSuperimpose/santafe
Drone image matching utility
Insulation
Less insulation

(b)

Figure 4.1: Screenshot of our IV tool: (a): Panorama image of a façade of the trailer in the DoE weatherization lab in Santa Fe, NM created by auto-stitching 10 RGB drone images; the user clicks anywhere on the panorama, and the footprint of the two IR images captured in two different flights, for that point are automatically displayed, for pre and post insulation retrofit. (b): Same as (a) but for a multifamily building in central CA, which underwent air sealing retrofit.

of the image capture device location and the pose of the image capture device. With this information, we use projection geometry to find each thermal image's corresponding facade as well as the projected optical center on that facade, as illustrated in Figure 4.2. We assume facades are all perpendicular to the ground plane, and therefore only need to use the two endpoints location of the facade $(x_0, y_0, 0)$ and $(x_1, y_1, 0)$ with x and y in global coordinate system retrieved from Google Earth to solve for the facade's plane equation

$$Ax + By + Cz + D = 0 \tag{4.1}$$

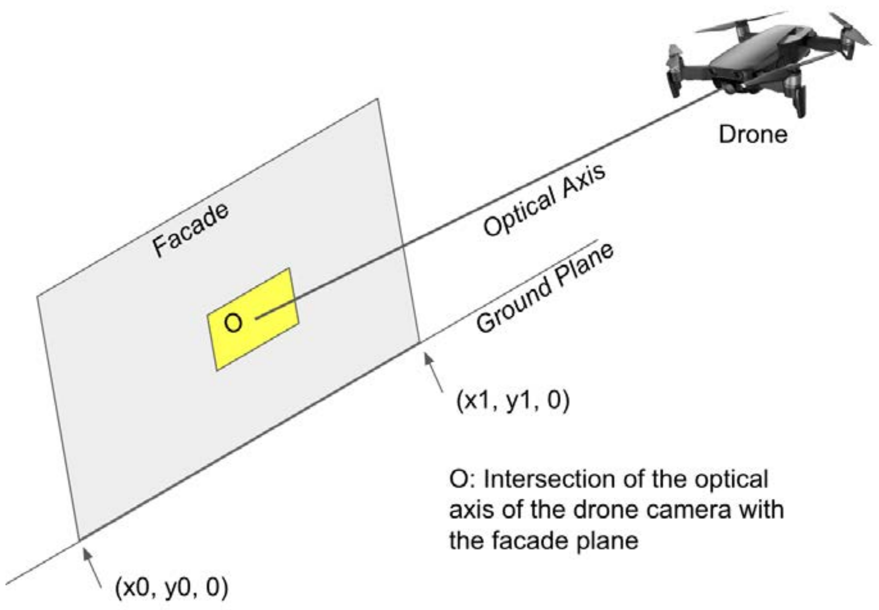
To find the projection of the optical center on the facade plane, we compute the optical axis of the drone camera. With the drone's position $(x_{drone}, y_{drone}, z_{drone})$ and roll, pitch and yaw $(0, v_{drone}, w_{drone})$, since the drone does not roll when taking a photo, the equation of the optical axis of the drone camera is

$$\begin{aligned} x &= x_{drone} + t * cos(v_{drone}) * cos(w_{drone}) \\ y &= y_{drone} + t * cos(v_{drone}) * sin(w_{drone}) \\ z &= z_{drone} + t * sin(v_{drone}) \end{aligned} \tag{4.2}$$

where t is a parameter that represents any point on the line. After that, the intersection of the facade plane and the optical axis of the drone camera, which by definition is the optical center of the captured image, can be computed by solving the value of t as the intersection of the plane in Equation 4.1 and the line in Equation 4.2

$$\begin{gathered}(Ax_{drone} + By_{drone} + Cz_{drone} + D)+ \\ (cos(v_{drone}) * cos(w_{drone})A + cos(v_{drone}) * sin(w_{drone})B + sin(v_{drone})C)t = 0\end{gathered} \tag{4.3}$$

Next, insert the t value back into the line in Equation 4.2 to obtain the position of the projected optical center on the facade when the given image is captured. With the projected optical centers of all the images computed, as the user clicks on any part of the facade panorama, we display all the thermal images that have projected optical centers close to the clicked point. Furthermore, if there were multiple drone captures, corresponding to before and after insulation retrofit, they are displayed side by side as shown in Figure 4.1, allowing for rapid, qualitative measurement and verification processes. As seen in Figure 4.1(b), there is a pronounced difference between the thermal pictures before and after the insulation process. Our tool allows all stakeholders including the contractor and the owner to view the before and after pictures immediately after a retrofit to pinpoint and remedy flaws. On the other hand, current building envelope retrofit measurement and verification process has to first lumped with mechanical systems, then takes two years of reviewing utility bills and does not pinpoint flaws such as missing insulation.


Drone
Optical Axis
Facade
O
Ground Plane
(x1, y1, 0)
(x0, y0, 0)
O: Intersection of the optical axis of the drone camera with the facade plane

(a)

Figure 4.2: Illustration of projecting drone camera's optical center on the facade by finding the intersection of the optical axis of the drone camera with the facade plane.

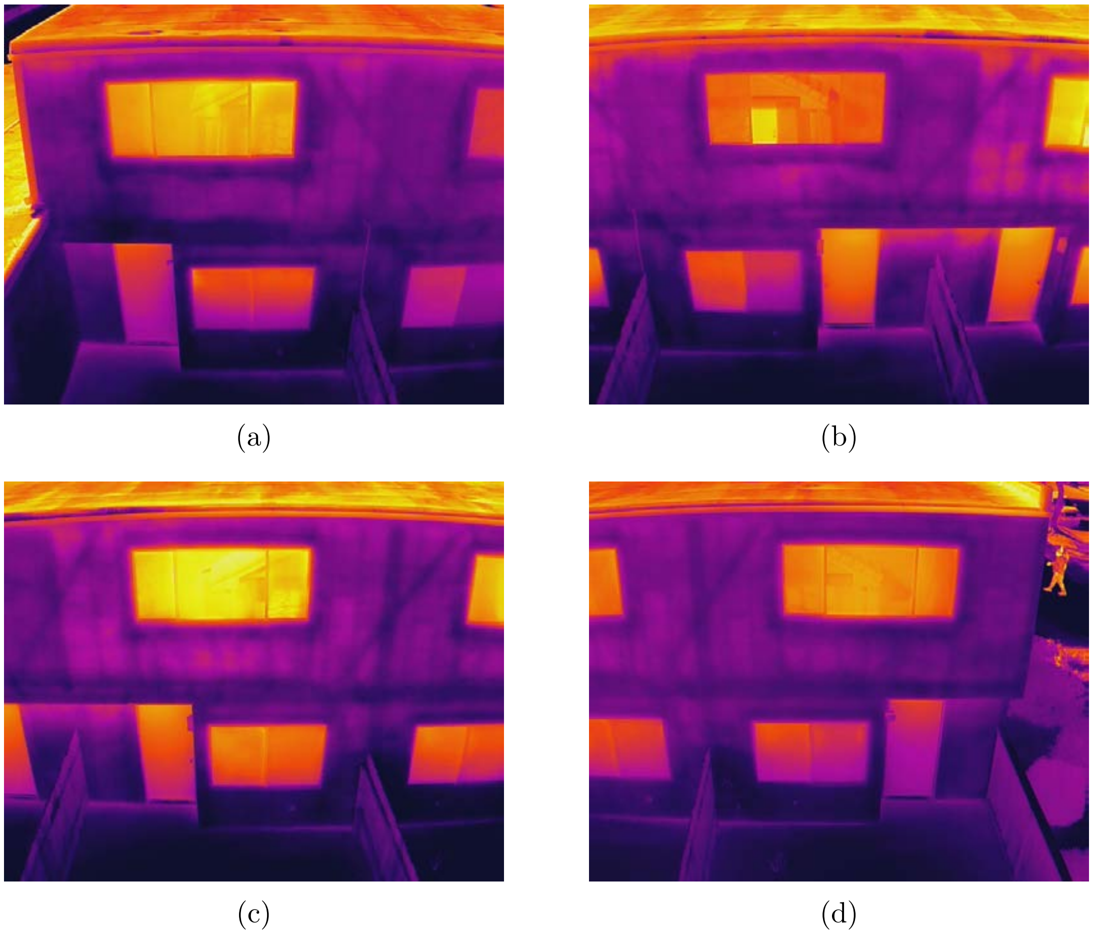


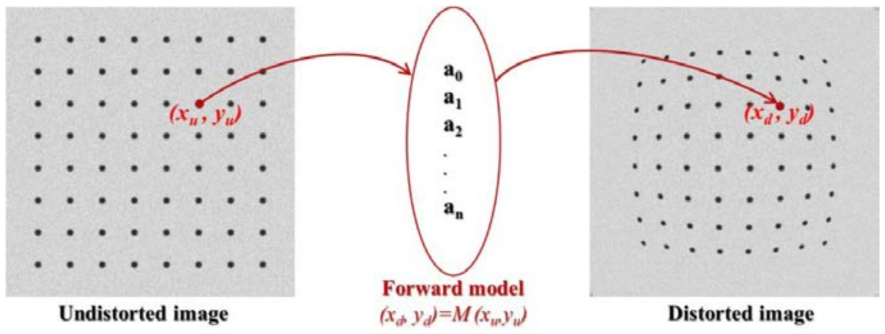
Figure 4.3: Distorted original captured thermal images.

4.2 Stitching IR Images

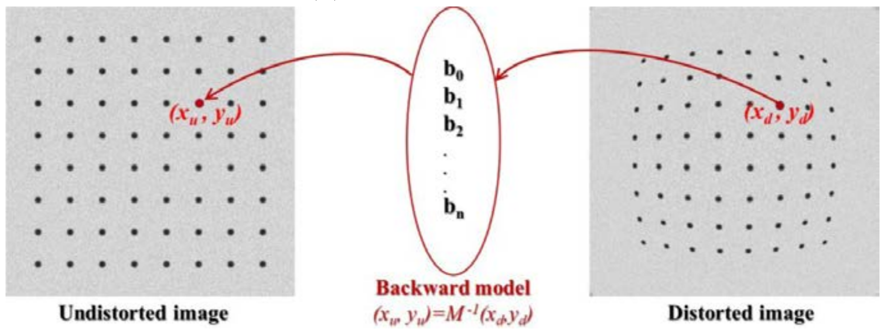
Another benefit of our IV tool is the ability to localize studs using drone-captured thermal imagery. This is useful in envelop retrofit projects such as recladding buildings with insulation panels. Currently, stud detection is done in the interior with a stud finder, which is then translated to the exterior, using landmarks such as corners of windows. As such, existing methods are tedious, laborious and error prone.

To improve the process of stud detection, our IV tool first stitches multiple thermal images of a façade into a thermal panorama. An example of a set of thermal images prior to stitching is shown in Figure 4.3.

Because of radial distortion, captured images initially do not guarantee that points located on a straight line in the real world will also be in a straight line on the images, so cannot be used directly to create a panorama. In the case of an IR thermography camera, image magnification decreases with distance from the optical axis, resulting in barrel dis-



(a) Forward mapping



(b) Backward mapping

Figure 4.4: Illustration of image distortion mapping. (a): Forward mapping maps from an undistorted space to a distorted space. (b): backward mapping maps from a distorted space to an undistorted space.

tortion. In general, to correct radial or perspective distortion, a model to map between the distorted plane and the undistorted plane is required. Mapping from an undistorted space to a distorted space is known as forward mapping as shown in Figure 4.4(a), and mapping from a distorted space to an undistorted space is called backward mapping and is shown in Figure 4.4(b). Parameters are calculated for determining a backward or inverse mapping model in order to undistort on thermal images according to the Brown-Conrady distortion model[7]:

$$\begin{aligned} x_u =& x_d + (x_d - x_c)\left(K_1 r^2 + K_2 r^4 + \cdots\right) + \left(P_1\left(r^2 + 2\left(x_d - x_c\right)^2\right)\right.\\ &\left.+2P_2\left(x_d - x_c\right)\left(y_d - y_c\right)\right)\left(1 + \mathrm{P}_3 r^2 + P_4 r^4 \cdots\right) \end{aligned} \tag{4.4}$$

$$\begin{aligned} y_u =& y_d + (y_d - y_c)\left(K_1 r^2 + K_2 r^4 + \cdots\right) + \left(2P_1\left(x_d - x_c\right)\left(y_d - y_c\right)\right.\\ &\left.+P_2\left(r^2 + 2\left(y_d - y_c\right)^2\right)\right)\left(1 + \mathrm{P}_3 r^2 + P_4 r^4 \cdots\right) \end{aligned} \tag{4.5}$$

where (x_d, y_d) is the distorted image point as projected on the image plane using the specified lens. (x_u, y_u) is the undistorted image point as projected by an ideal pinhole camera. (x_c, y_c)

is the distortion center, which is the center of the image since the distortion is symmetrical. K_n is the n^{th} q radial distortion coefficient, and $r = \left((x_d + x_c)^2 + (y_d - y_c)^2\right)^{1/2}$ is the Euclidean distance between the distorted image point and the distortion center. Tangential distortion is ignored since the lens and the image plane are parallel. The approach is to create a line pattern for visual inspection and apply an estimated forward model to the line pattern and overlay the result on top of the image in order to find the appropriate parameters for the forward model. This is accomplished by adjusting the parameters of a forward model, applying them to the line pattern, overlaying to the distorted image, and making sure the warped lines are visually matching with the curve feature in the image. Values for K_2, K_3, and K_4 are tuned one by one to gain an approximation. We skip K_1 since it mainly controls the scaling and the shearing of the warped image. Examples of applying forward model to line-pattern with different values of K_2 are presented in Figure 4.5. One can compare lines that are required to be straight in the distorted original thermal image with the distorted line-pattern to see which coefficient value best match the distortion present in the thermal image. For instance, within the distorted thermal image presented in Figure 4.5 the top edge of the facade, which should be straight in the real world, aligns best with the curve that has $K_2 = 1$. With the estimated coefficients of the backward model, we map pixels from distorted space to undistorted space to produce the calibrated image. When mapping a pixel from distorted space to undistorted space, the computed undistorted pixel coordinates x_u and y_u are not guaranteed to be integers. Therefore, we round the undistorted pixel coordinates to their nearest integer values before copying pixel values from the distorted image to the undistorted image. If any pixel on the undistorted image gets mapped from more than one pixel from the distorted image, the final pixel value on the undistorted image pixel is computed as the average of the pixel values mapped from the distorted image. For all the pixels on the undistorted image that do not get mapped to from at least one pixel, the final pixel value on the undistorted image pixel is computed as the average of all the surrounding pixel values on the undistorted image in order to fill in the blank. Examples of original thermal images in Figure 4.3 with distortion calibrated are provided in Figure 4.6.

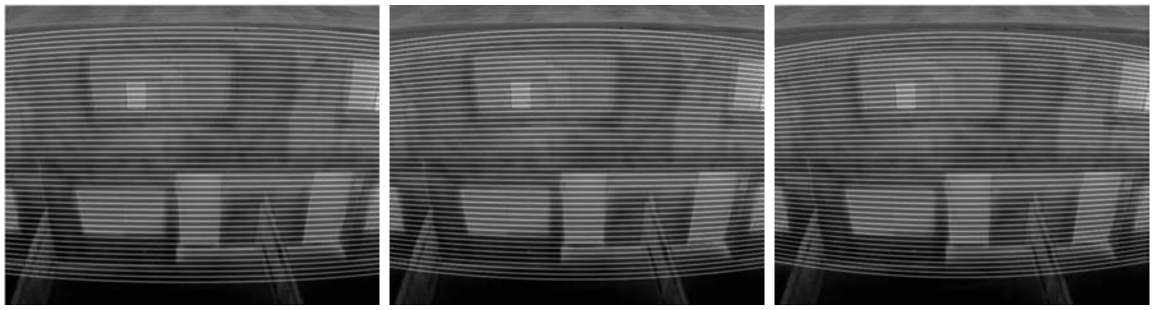

(a) $K_2 = 1$ (b) $K_2 = 6$ (c) $K_2 = 11$

Figure 4.5: Applying forward model to line-pattern with different values of K_2

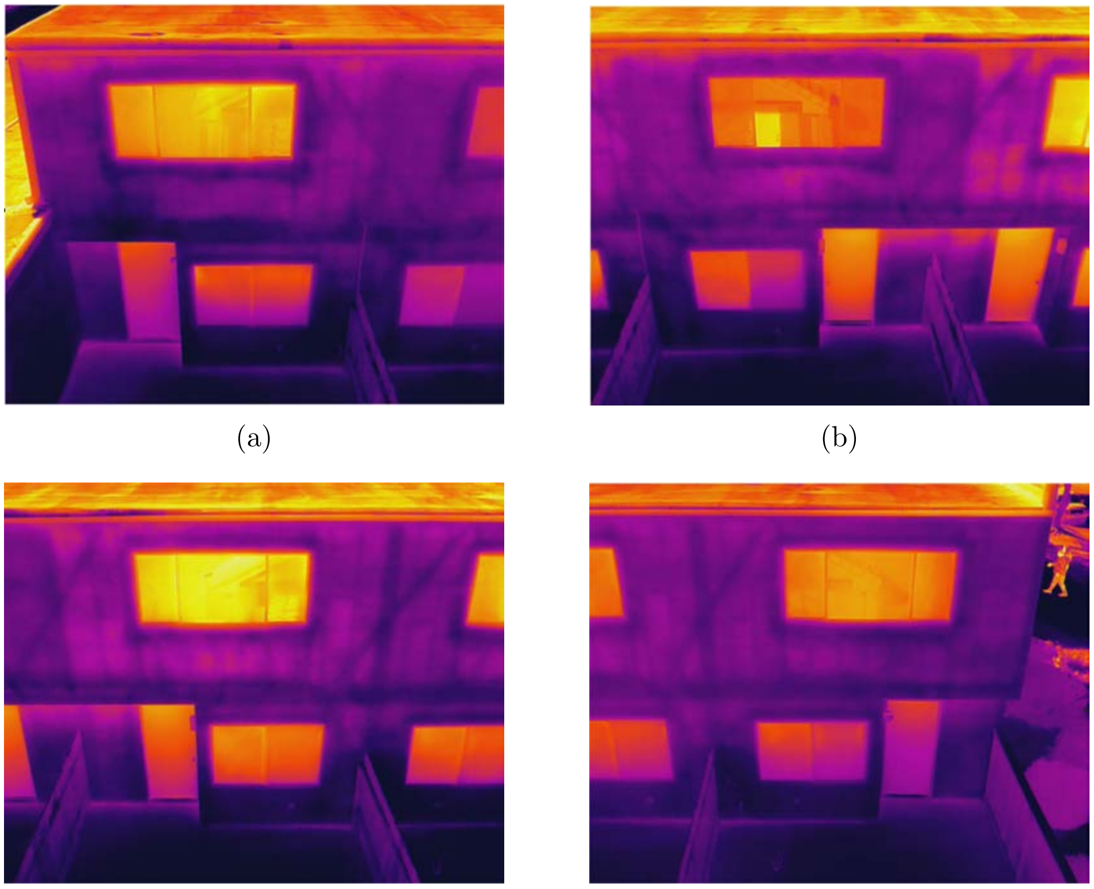


Figure 4.6: Thermal images with calibrated distortion.

With all the thermal images calibrated, we use projective transformations to stitch pairs of images together one at a time. To stitch two calibrated images together, we first find the homography matrix that defines the projective transformation between the two images. The homography matrix H is a 3×3 matrix with the form:

$$H = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \tag{4.6}$$

Any point (x, y) can be projected from one image to the corresponding point (x', y') on the other image with the transformation defined by:

$$\begin{bmatrix} x'/\lambda \\ y'/\lambda \\ \lambda \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{4.7}$$

The transformation parameters in H can be solved using the following equation, where (x_n, y_n) and (x'_n, y'_n) are the n^{th} pair of corresponding points on the pair of images.

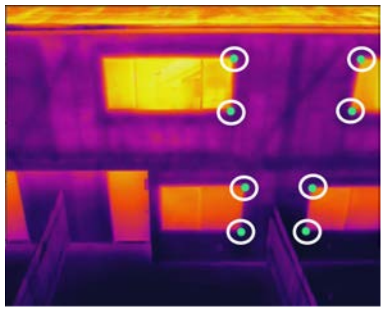
$$\begin{bmatrix} -x_1 & -y_1 & -1 & 0 & 0 & 0 & x_1x'_1 & y_1x'_1 & x'_1 \\ 0 & 0 & 0 & -x_1 & -y_1 & -1 & x_1y'_1 & y_1y'_1 & y'_1 \\ -x_2 & -y_2 & -1 & 0 & 0 & 0 & x_2x'_2 & y_2x'_2 & x'_2 \\ 0 & 0 & 0 & -x_2 & -y_2 & -1 & x_2y'_2 & y_2y'_2 & y'_2 \\ -x_3 & -y_y & -1 & 0 & 0 & 0 & x_3x'_3 & y_3x'_3 & x'_3 \\ 0 & 0 & 0 & -x_3 & -y_3 & -1 & x_3y'_3 & y_3y'_3 & y'_3 \\ -x_4 & -y_4 & -1 & 0 & 0 & 0 & x_4x'_4 & y_4x'_4 & x'_4 \\ 0 & 0 & 0 & -x_4 & -y_4 & -1 & x_4y'_4 & y_4y'_4 & y'_4 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} h1 \\ h2 \\ h3 \\ h4 \\ h5 \\ h6 \\ h7 \\ h8 \\ h9 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \tag{4.8}$$

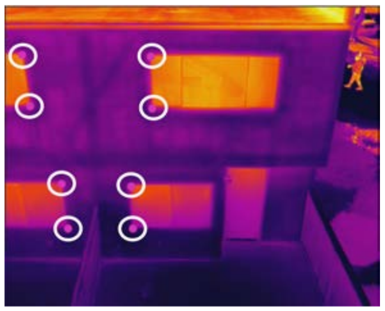
In theory, only four pairs of points are needed to compute the homography matrix. That said, in practice, we used six pairs to make the process more robust.

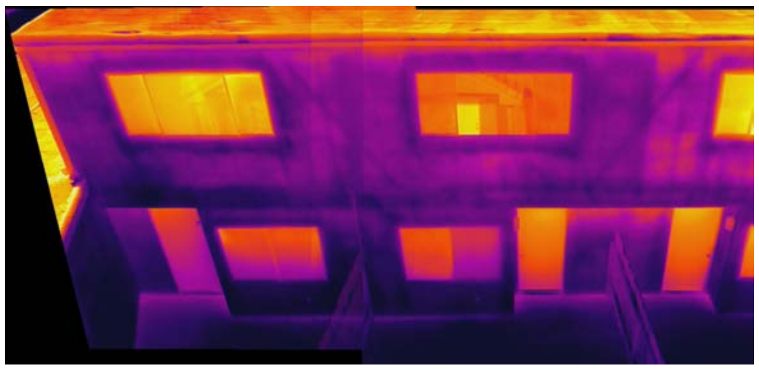
Unlike the process of stitching high resolution RGB images where key-point detection can be applied and the transformation can be found automatically using algorithms such as random sample consensus (RANSAC), thermal images have lower resolution compared to RGB images, and colormap visualization makes it difficult to extract meaningful key-points. Therefore, our IV tool utilizes a pipeline to let the user manually choose the correspondences between pair of images to be stitched together. For instance, if a user desires to stitch two images shown in Figures 4.7(a) and 4.7(b) together, pairs of corresponding points are selected between images. Using the pixel coordinates of the corresponding selected points, homography matrix H can be solved using least-square linear equation solver. Then we use the computed homography matrix to map pixels from one image onto the other image. A result of stitching using images in Figures 4.7(a) and 4.7(b) is shown in Figure 4.7(c). The process may be iterated to produce a fully stitched panorama shown in Figure 4.7(d).

In the next step, the method corrects perspective distortion of the façade image. Perspective distortion arises because images cannot often be captured head-on. Thus, ground-level photography may introduce low angle perspective distortion while aerial photography may introduce high angle perspective distortion. An example of high angle perspective distortion is illustrated by stitched IR thermography image shown in Figure 4.7(d). In addition to removing perspective distortion, the resulting image should have the same aspect ratio as the actual facade. Given a pre-built 3D model of the building, as shown in Figure 4.8(a), the exact aspect ratio R according to the dimensions of the facade can be extracted. A user may select the pixel coordinates of facade corners, with the two upper corners of the facade being (x_l, y_l) and (x_r, y_r) and the two lower corners of the facade being (x'_l, y'_l) and (x'_r, y'_r). Using the aspect ratio R, the pixel coordinates of the two lower corners of the facade after perspective distortion correction can be computed as $(x_l, y_l + \frac{x_r - x_l}{R})$ and $(x_r, y_r + \frac{x_r - x_l}{R})$. The relationship between facade panorama before and after perspective correction is again a perspective transformation. With the pairs of correspondences being $[(x_l, y_l), (x_l, y_l)]$, $[(x_r, y_r), (x_r, y_r)]$, $[(x'_l, y'_l), (x_l, y_l + \frac{x_r - x_l}{R})]$ and $[(x'_r, y'_r), (x_r, y_r + \frac{x_r - x_l}{R})]$. Then the homography matrix can be computed in a similar way as the case for stitching thermal images to produce the head-on thermal facade panorama, as shown in Figure 4.8(b).

In the final thermal panorama shown in Figure 4.8(b), the location of the studs is clearly indicated. Next, we export this to an architecture tool such as Rhino or Revit for the user to visually click on the studs as shown in Figure 4.9. Users can click on studs on the thermal facade to find the metric locations of the studs with respect to the origin (0,0,0) which in this case is the lower right corner of the facade.


(a)


(b)


(c)

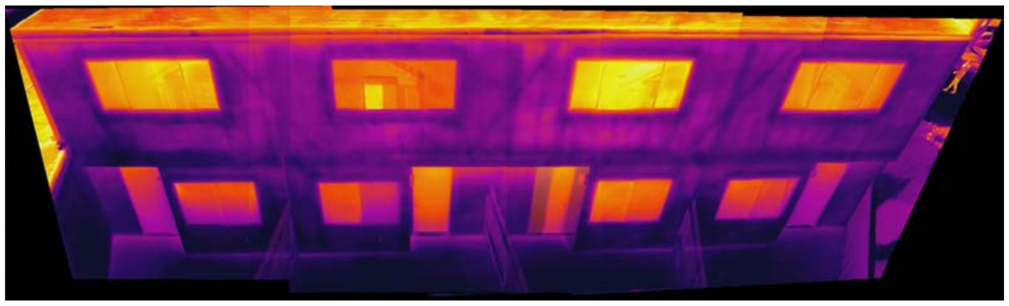
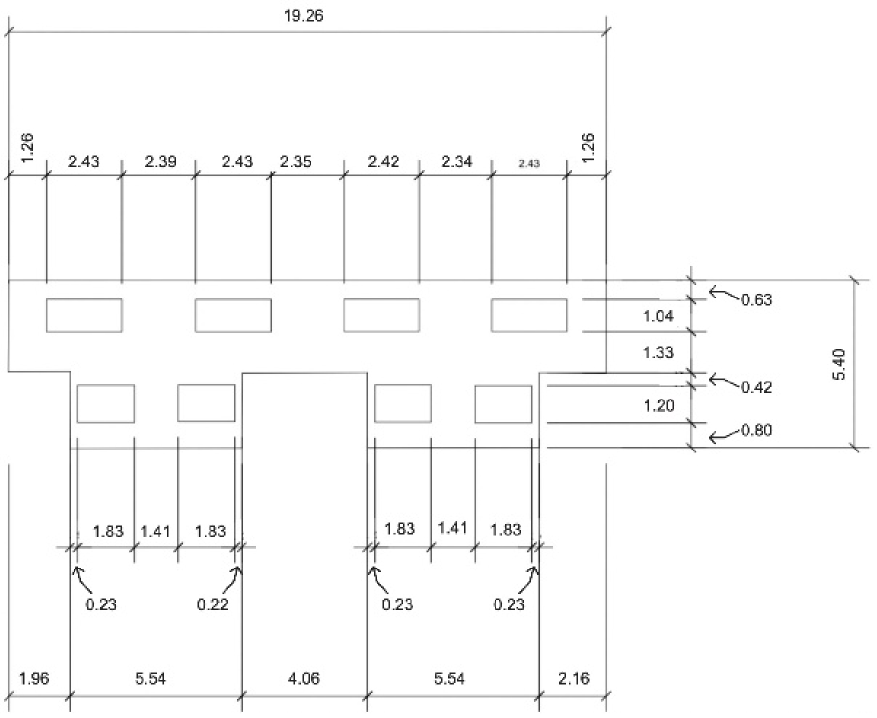
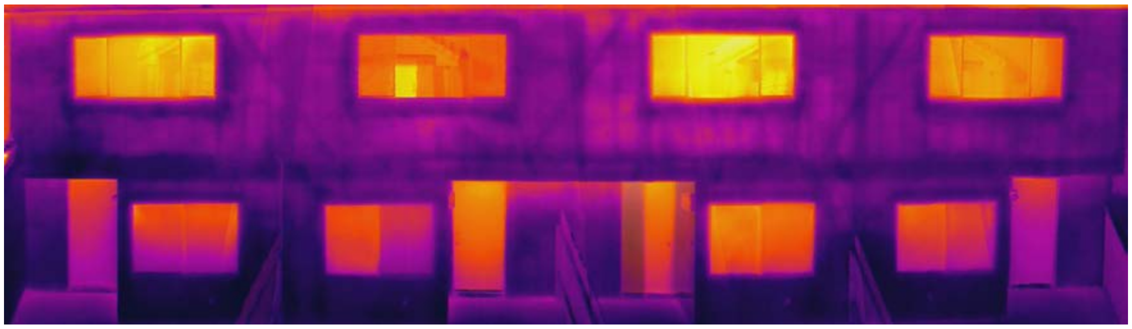

(d)

Figure 4.7: To stitch two thermal images in (a) and (b) together, pairs of corresponding points are selected between images, then homography matrix H can be solved. (c): Result of stitching (a) and (b). (d): Fully stitched panorama produced by iterating the stitching process.



(a) Dimensions extracted from 3D model of the building.



(b) Final thermal panorama with correct aspect ratio.

Figure 4.8: (a): Given a pre-built 3D model of the building, the exact aspect ratio R according to the dimensions can be extracted. (b): Final head-on thermal facade panorama.

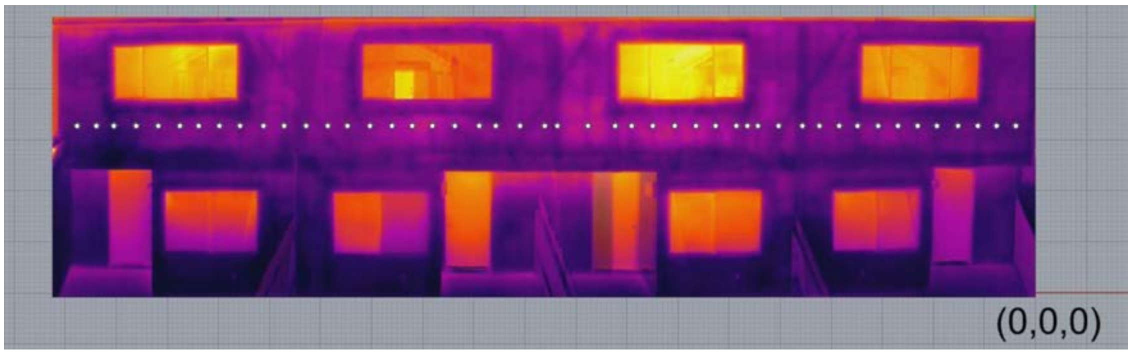


Figure 4.9: Screenshot of Rhino architecture tool. Users can click on studs on the thermal facade to find the metric locations of the studs with respect to the origin (0,0,0) which in this case is the lower right corner of the facade.

Chapter 5

Conclusions and Future Work

In this work, we present an end-to-end approach to learning a perceptive walking model for low-cost hexapods, to climb over high and dense joist structures. We also show zero-shot sim-to-real deployment of models on hardware in a constructed test environment that closely matches the proposed use case of our method in attics. Furthermore, we present an SIV tool that improves the measurement and verification process of building retrofit.

Our perceptive hexapod controller significantly outperforms the baseline methods in terms of both the number of joists it climbs over within given time and the average time it takes to climb over each joist. As shown in our ablation studies, the success of our method's sim-to-real transfer is credited to the effectiveness of our proposed depth image preprocessing techniques and including critical privileged information during phase 1 training.

Even though our robot's compactness and low height allow it to traverse tight spaces, the depth camera mounted above to obtain a view of the terrain elevates the center of mass of the entire system. There are certain instances where the robot falls likely due to the instability related to an elevated center of mass. In the future, we would like to further improve the stability of our system by changing the design of our camera mount to lower the center of mass. This might entail learning a visual legged locomotion controller with a lower egocentric view.

Using a L515 LiDAR instead of a stereo camera as the perception sensor allows our robot to operate in dark environments. However, the T265 tracking RGB camera often loses track of the robot's orientation in the dark. This can be improved by switching our choice of pose sensor from a tracking camera to a lidar-based tracking method.

Although being out of the scope of this work, we wish to explore letting the robot climb staircases or get around joists or other obstacles that are too large to climb. Eventually, we would like to teach the hexapod to perform high-level path planning in cluttered environments such as attics.

Bibliography

[1] Ananye Agarwal et al. *Legged Locomotion in Challenging Terrains using Egocentric Vision*. 2022. DOI: 10.48550/ARXIV.2211.07638. URL: https://arxiv.org/abs/2211.07638.

[2] Ayush Agrawal et al. *Vision-aided Dynamic Quadrupedal Locomotion on Discrete Terrain using Motion Libraries*. 2021. DOI: 10.48550/ARXIV.2110.00891. URL: https://arxiv.org/abs/2110.00891.

[3] Teymur Azayev and Karel Zimmerman. “Blind Hexapod Locomotion in Complex Terrain with Gait Adaptation Using Deep Reinforcement Learning and Classification”. In: *Journal of Intelligent & Robotic Systems* 99 (Sept. 2020). DOI: 10.1007/s10846-020-01162-8.

[4] Filip Bjelonic et al. “Learning-based Design and Control for Quadrupedal Robots with Parallel-Elastic Actuators”. In: *CoRR* abs/2301.03509 (2023). DOI: 10.48550/arXiv.2301.03509. arXiv: 2301.03509. URL: https://doi.org/10.48550/arXiv.2301.03509.

[5] Marko Bjelonic et al. “Weaver: Hexapod Robot for Autonomous Navigation on Unstructured Terrain”. In: *Journal of Field Robotics* 35 (June 2018), pp. 1063–1079. DOI: 10.1002/rob.21795.

[6] R. Omar Chavez-Garcia et al. “Learning Ground Traversability From Simulations”. In: *IEEE Robotics and Automation Letters* 3 (2017), pp. 1695–1702.

[7] A. E. Conrady. “Decentred Lens-Systems”. In: *Monthly Notices of the Royal Astronomical Society* 79.5 (Mar. 1919), pp. 384–390. ISSN: 0035-8711. DOI: 10.1093/mnras/79.5.384. eprint: https://academic.oup.com/mnras/article-pdf/79/5/384/18250798/mnras79-0384.pdf. URL: https://doi.org/10.1093/mnras/79.5.384.

[8] Liang Ding et al. *Fault Tolerant Free Gait and Footstep Planning for Hexapod Robot Based on Monte-Carlo Tree*. 2020. DOI: 10.48550/ARXIV.2006.07550. URL: https://arxiv.org/abs/2006.07550.

[9] Jan Faigl and Petr Čížek. “Adaptive locomotion control of hexapod walking robot for traversing rough terrains with position feedback only”. In: *Robotics and Autonomous Systems* 116 (Mar. 2019). DOI: 10.1016/j.robot.2019.03.008.

[10] Peter Fankhauser et al. "Robust Rough-Terrain Locomotion with a Quadrupedal Robot". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. 2018, pp. 5761–5768. DOI: 10.1109/ICRA.2018.8460731.

[11] Peter Fankhauser et al. "Robust Rough-Terrain Locomotion with a Quadrupedal Robot". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. 2018, pp. 5761–5768. DOI: 10.1109/ICRA.2018.8460731.

[12] Péter Fankhauser and Marco Hutter. "A Universal Grid Map Library: Implementation and Use Case for Rough Terrain Navigation". In: 2016.

[13] Siddhant Gangapurwala et al. "Real-Time Trajectory Adaptation for Quadrupedal Locomotion using Deep Reinforcement Learning". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. 2021, pp. 5973–5979. DOI: 10.1109/ICRA48506.2021.9561639.

[14] *GitHub - krupkat/xpano: Automated photo stiching tool.* https://github.com/krupkat/xpano.

[15] *Hiwonder*. https://hiwonder.hk/.

[16] Deepali Jain, Atil Iscen, and Ken Caluwaerts. "From Pixels to Legs: Hierarchical Learning of Quadruped Locomotion". In: *Conference on Robot Learning*. 2020.

[17] Fabian Jenelten et al. "Perceptive Locomotion in Rough Terrain – Online Foothold Optimization". In: *IEEE Robotics and Automation Letters* 5.4 (2020), pp. 5370–5376. DOI: 10.1109/LRA.2020.3007427.

[18] Mrinal Kalakrishnan et al. "Learning locomotion over rough terrain using terrain templates". In: *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2009, pp. 167–172. DOI: 10.1109/IROS.2009.5354701.

[19] Maxime Kawawa-Beaudan. *Learning to Walk: Legged Hexapod Locomotion from Simulation to the Real World*. Berkeley, CA, USA, 2022.

[20] J. Zico Kolter, Michael P. Rodgers, and A. Ng. "A control architecture for quadruped locomotion over rough terrain". In: *2008 IEEE International Conference on Robotics and Automation* (2008), pp. 811–818.

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

[22] Ashwin Sanjay Lele et al. "Learning to Walk: Spike Based Reinforcement Learning for Hexapod Robot Central Pattern Generation". In: *2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. 2020, pp. 208–212. DOI: 10.1109/AICAS48895.2020.9073987.

[23] Jiayu Li et al. "Dual-Master/Single-Slave Haptic Teleoperation System for Semiautonomous Bilateral Control of Hexapod Robot Subject to Deformable Rough Terrain". In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 52.4 (2022), pp. 2435–2449. DOI: `10.1109/TSMC.2021.3049848`.

[24] Tianyu Li et al. *Learning Generalizable Locomotion Skills with Hierarchical Reinforcement Learning*. 2019. DOI: `10.48550/ARXIV.1909.12324`. URL: `https://arxiv.org/abs/1909.12324`.

[25] Gabriel B. Margolis et al. *Learning to Jump from Pixels*. 2021. DOI: `10.48550/ARXIV.2110.15344`. URL: `https://arxiv.org/abs/2110.15344`.

[26] Carlos Mastalli et al. "On-line and On-board Planning and Perception for Quadrupedal Locomotion". In: May 2015. DOI: `10.1109/TePRA.2015.7219685`.

[27] Kharunya Paramaguru. *Oscar Pistorius: The blade runner makes olympic history*. July 2012. URL: `https://olympics.time.com/2012/07/05/oscar-pistorius-the-blade-runner-makes-olympic-history/`.

[28] Zhiying Qiu, Wu Wei, and Xiongding Liu. "Adaptive Gait Generation for Hexapod Robots Based on Reinforcement Learning and Hierarchical Framework". In: *Actuators* 12.2 (2023). ISSN: 2076-0825. DOI: `10.3390/act12020075`. URL: `https://www.mdpi.com/2076-0825/12/2/75`.

[29] Martin Wermelinger et al. "Navigation planning for legged robots in challenging terrain". In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2016, pp. 1184–1189. DOI: `10.1109/IROS.2016.7759199`.

[30] Martin Wermelinger et al. "Navigation planning for legged robots in challenging terrain". In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2016), pp. 1184–1189.

[31] Wenhao Yu et al. "Visual-Locomotion: Learning to Walk on Complex Terrains with Vision". In: *5th Annual Conference on Robot Learning*. 2021. URL: `https://openreview.net/forum?id=NDYbXf-DvwZ`.

[32] He Zhang et al. "A Force-Sensing System on Legs for Biomimetic Hexapod Robots Interacting with Unstructured Terrain". In: *Sensors* 17.7 (2017). ISSN: 1424-8220. DOI: `10.3390/s17071514`. URL: `https://www.mdpi.com/1424-8220/17/7/1514`.

Appendix A

Custom URDF Robot Representation

```
<?xml version="1.0" ?>
<robot name="pexod" xmlns:xacro="http://www.ros.org/wiki/xacro">
  <!-- <xacro:include filename="$(find pexod_description)/urdf/pexod_control.xacro" /> -
  <!-- MATERIALS -->
  <material name="Blue">
    <color rgba="0 0 1 1"/>
  </material>
  <material name="Red">
    <color rgba="1 0 0 1"/>
  </material>
  <material name="Green">
    <color rgba="0 1 0 1"/>
  </material>
  <material name="Yellow">
    <color rgba="1 1 0 1"/>
  </material>
  <material name="LightGrey">
    <color rgba="0.6 0.6 0.6 1.0"/>
  </material>
  <!-- END OF MATERIALS -->
  <!-- XACRO MACROS FOR VISUALS AND COLLISIONS -->
  <!-- END OF XACRO MACROS -->
  <!-- TORSO -->
  <link name="base_link">
    <visual>
      <origin rpy="0 0 0" xyz="0 0 0"/>
      <geometry>
        <box size="0.239 0.117 0.095"/>
      </geometry>
      <material name="Blue"/>
    </visual>
    <collision>
      <origin rpy="0 0 0" xyz="0 0 0"/>
      <geometry>
  <!-- END OF LEG LINKS/JOINTS -->
        <box size="0.239 0.117 0.095"/>
      </geometry>
    </collision>
    <inertial>
      <!-- CENTER OF MASS -->
      <origin rpy="0 0 0" xyz="0 0 0"/>
      <!-- <mass value="0.01"/> -->
      <mass value="1.396"/>
      <!-- box inertia: 1/12*m(y^2+z^2), ... -->
      <inertia ixx="0.003" ixy="0" ixz="0" iyy="0.008" iyz="0" izz="0.008"/>
    </inertial>
  </link>
  <joint name="body_camera" type="fixed">
    <parent link="base_link"/>
    <child link="camera_mount"/>
    <origin rpy="0 0 0" xyz="0.1 0 0.025"/>
</joint>

<link name="camera_mount">
    <visual>
      <origin rpy="0 0 0" xyz="0 0 0.11"/>
      <geometry>
        <box size=" 0.05 0.117 0.17"/>
      </geometry>
      <material name="Blue"/>
    </visual>
    <collision>
      <origin rpy="0 0 0" xyz="0 0 0.11"/>
      <geometry>
  <!-- END OF LEG LINKS/JOINTS -->
        <box size=" 0.05 0.117 0.17"/>
      </geometry>
    </collision>
    <inertial>
      <!-- CENTER OF MASS -->
      <origin rpy="0 0 0" xyz="0 0 0.11"/>
      <!-- <mass value="0.01"/> -->
      <mass value="0.50"/>
      <!-- box inertia: 1/12*m(y^2+z^2), ... -->
      <inertia ixx="0.0017" ixy="0" ixz="0" iyy="0.0013" iyz="0" izz="0.00067"/>
    </inertial>
</link>
  <!-- XACRO MACRO FOR LEGS LINKS/JOINTS -->
  <joint name="body_leg_2" type="revolute">
    <parent link="base_link"/>
    <child link="leg_2_1"/>
    <limit effort="1.961" lower="-2.0944" upper="2.0944" velocity="5.236"/>
    <origin rpy="0 0 -0.785" xyz="0.119 0.0585 -0.025"/>
    <!-- <xacro:if value="${index == 5 or index == 4 or index == 3}">
                <axis xyz="0 0 1"/>
            </xacro:if>
            <xacro:if value="${index == 2 or index == 1 or index == 0}"> -->
    <axis xyz="0 0 1"/>
    <!-- </xacro:if> -->
    <dynamics damping="0" friction="0"/>
  </joint>
  <link name="leg_2_1">
    <visual>
      <origin rpy="1.57079632679 0 0" xyz="0 0.0225 0"/>
      <geometry>
        <box size="0.055 0.02 0.045"/>
        <!-- <cylinder length="0.04" radius="0.02"/> -->
      </geometry>
      <material name="Red"/>
    </visual>

    <collision>
      <origin rpy="1.57079632679 0 0" xyz="0 0.0225 0"/>
      <geometry>
        <box size="0.055 0.02 0.045"/>
      </geometry>
    </collision>
    <inertial>
      <!-- CENTER OF MASS -->
      <origin rpy="1.57079632679 0 0" xyz="0 0.0225 0"/>
      <mass value="0.01"/>
      <!-- box inertia: 1/12*m(y^2+z^2), ... -->
      <inertia ixx="2.021e-6" ixy="0" ixz="0" iyy="4.208e-6" iyz="0" izz="2.854e-6"/>
    </inertial>
  </link>
  <joint name="leg_2_1_2" type="revolute">
    <parent link="leg_2_1"/>
    <child link="leg_2_2"/>
    <limit effort="1.961" lower="-2.0944" upper="2.0944" velocity="5.236"/>
    <origin rpy="0 0 0" xyz="0 0.045 0"/>
    <axis xyz="1 0 0"/>
    <dynamics damping="0" friction="0"/>
  </joint>
  <link name="leg_2_2">
    <visual>
      <origin rpy="1.57079632679 0 0" xyz="0 0.045 0"/>
      <geometry>
        <box size="0.044 0.02 0.09"/>
        <!-- <cylinder length="0.09" radius="0.02"/> -->
      </geometry>
      <material name="Blue"/>
    </visual>
    <collision>
      <origin rpy="1.57079632679 0 0" xyz="0 0.045 0"/>
      <geometry>
        <box size="0.044 0.02 0.09"/>
      </geometry>
    </collision>
    <inertial>
      <!-- CENTER OF MASS -->
      <origin rpy="1.57079632679 0 0" xyz="0 0.045 0"/>
      <mass value="0.124"/>
      <!-- box inertia: 1/12*m(y^2+z^2), ... -->
      <inertia ixx="0.00008783" ixy="0" ixz="0" iyy="1.037e-4" iyz="0" izz="2.414e-5"/>
    </inertial>
  </link>
  <joint name="leg_2_2_3" type="revolute">
    <parent link="leg_2_2"/>
    <child link="leg_2_3"/>
    <limit effort="1.961" lower="-2.0944" upper="2.0944" velocity="5.236"/>
    <origin rpy="0 0 0" xyz="0 0.09 0"/>
    <axis xyz="-1 0 0"/>
    <dynamics damping="0" friction="0"/>
  </joint>
  <link name="leg_2_3">
    <visual>
      <origin rpy="0.785 0 0" xyz="0 0.0441941738 -0.0441941738"/>
      <geometry>
        <box size="0.055 0.04 0.125"/>
        <!-- <cylinder length="0.125" radius="0.025"/> -->
      </geometry>
      <material name="Red"/>
    </visual>

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

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

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

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

  <joint name="body_leg_3" type="revolute">
    <parent link="base_link"/>
    <child link="leg_3_1"/>
    <limit effort="1.961" lower="-2.0944" upper="2.0944" velocity="5.236"/>
    <origin rpy="0 0 0.785" xyz="0.119 -0.0585 -0.025"/>
    <!-- <xacro:if value="${index == 5 or index == 4 or index == 3}">
                <axis xyz="0 0 1"/>
            </xacro:if>
            <xacro:if value="${index == 2 or index == 1 or index == 0}"> -->
    <axis xyz="0 0 -1"/>
    <!-- </xacro:if> -->
    <dynamics damping="0" friction="0"/>
  </joint>
  <link name="leg_3_1">
    <visual>
      <origin rpy="1.57079632679 0 0" xyz="0 -0.0225 0"/>
      <geometry>
        <box size="0.055 0.02 0.045"/>
        <!-- <cylinder length="0.04" radius="0.02"/> -->
      </geometry>
      <material name="Red"/>
    </visual>
    <collision>
      <origin rpy="1.57079632679 0 0" xyz="0 -0.0225 0"/>
      <geometry>
        <box size="0.055 0.02 0.045"/>
      </geometry>
    </collision>
    <inertial>
      <!-- CENTER OF MASS -->
      <origin rpy="1.57079632679 0 0" xyz="0 -0.0225 0"/>
      <mass value="0.01"/>
      <!-- box inertia: 1/12*m(y^2+z^2), ... -->
      <inertia ixx="2.021e-6" ixy="0" ixz="0" iyy="4.208e-6" iyz="0" izz="2.854e-6"/>
    </inertial>
  </link>
  <joint name="leg_3_1_2" type="revolute">
```

```
    <parent link="leg_3_1"/>
    <child link="leg_3_2"/>
    <limit effort="1.961" lower="-2.0944" upper="2.0944" velocity="5.236"/>
    <origin rpy="0 0 0" xyz="0 -0.045 0"/>
    <axis xyz="-1 0 0"/>
    <dynamics damping="0" friction="0"/>
  </joint>
  <link name="leg_3_2">
    <visual>
      <origin rpy="1.57079632679 0 0" xyz="0 -0.045 0"/>
      <geometry>
        <box size="0.044 0.02 0.09"/>
        <!-- <cylinder length="0.09" radius="0.02"/> -->
      </geometry>
      <material name="Blue"/>
    </visual>
    <collision>
      <origin rpy="1.57079632679 0 0" xyz="0 -0.045 0"/>
      <geometry>
        <box size="0.044 0.02 0.09"/>
      </geometry>
    </collision>
    <inertial>
      <!-- CENTER OF MASS -->
      <origin rpy="1.57079632679 0 0" xyz="0 -0.045 0"/>
      <mass value="0.124"/>
      <!-- box inertia: 1/12*m(y^2+z^2), ... -->
      <inertia ixx="0.00008783" ixy="0" ixz="0" iyy="1.037e-4" iyz="0" izz="2.414e-5"/>
    </inertial>
  </link>
  <joint name="leg_3_2_3" type="revolute">
    <parent link="leg_3_2"/>
    <child link="leg_3_3"/>
    <limit effort="1.961" lower="-2.0944" upper="2.0944" velocity="5.236"/>
    <origin rpy="0 0 0" xyz="0 -0.09 0"/>
    <axis xyz="1 0 0"/>
    <dynamics damping="0" friction="0"/>
  </joint>
  <link name="leg_3_3">
    <visual>
      <origin rpy="-0.785 0 0" xyz="0 -0.0441941738 -0.0441941738"/>
      <geometry>
```

```
        <box size="0.055 0.04 0.125"/>
        <!-- <cylinder length="0.125" radius="0.025"/> -->
      </geometry>
      <material name="Red"/>
    </visual>

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

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

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

  <link name="dummy_eef_3">
  <visual>
      <geometry>
        <sphere radius="0.005"/>
      </geometry>
      <material name="Green"/>
    </visual>
    <inertial>
      <!-- CENTER OF MASS -->
      <origin rpy="0 0 0" xyz="0 0 0"/>
      <mass value="0"/>
      <inertia ixx="0" ixy="0" ixz="0" iyy="0" iyz="0" izz="0"/>
    </inertial>
  </link>
  <joint name="body_leg_0" type="revolute">
    <parent link="base_link"/>
    <child link="leg_0_1"/>
    <limit effort="1.961" lower="-2.0944" upper="2.0944" velocity="5.236"/>
    <origin rpy="0 0 0.785" xyz="-0.119 0.0585 -0.025"/>
    <!-- <xacro:if value="${index == 5 or index == 4 or index == 3}">
                <axis xyz="0 0 1"/>
            </xacro:if>
            <xacro:if value="${index == 2 or index == 1 or index == 0}"> -->
    <axis xyz="0 0 1"/>
    <!-- </xacro:if> -->
    <dynamics damping="0" friction="0"/>
  </joint>
  <link name="leg_0_1">
    <visual>
      <origin rpy="1.57079632679 0 0" xyz="0 0.0225 0"/>
      <geometry>
        <box size="0.055 0.02 0.045"/>
        <!-- <cylinder length="0.04" radius="0.02"/> -->
      </geometry>
      <material name="Red"/>
    </visual>
    <collision>
      <origin rpy="1.57079632679 0 0" xyz="0 0.0225 0"/>
      <geometry>
        <box size="0.055 0.02 0.045"/>
      </geometry>
    </collision>
    <inertial>
      <!-- CENTER OF MASS -->
      <origin rpy="1.57079632679 0 0" xyz="0 0.0225 0"/>
      <mass value="0.01"/>
      <!-- box inertia: 1/12*m(y^2+z^2), ... -->
      <inertia ixx="2.021e-6" ixy="0" ixz="0" iyy="4.208e-6" iyz="0" izz="2.854e-6"/>
    </inertial>
  </link>
  <joint name="leg_0_1_2" type="revolute">
    <parent link="leg_0_1"/>
    <child link="leg_0_2"/>
    <limit effort="1.961" lower="-2.0944" upper="2.0944" velocity="5.236"/>
    <origin rpy="0 0 0" xyz="0 0.045 0"/>
    <axis xyz="1 0 0"/>
    <dynamics damping="0" friction="0"/>
  </joint>
  <link name="leg_0_2">
    <visual>
      <origin rpy="1.57079632679 0 0" xyz="0 0.045 0"/>
      <geometry>
        <box size="0.044 0.02 0.09"/>
        <!-- <cylinder length="0.09" radius="0.02"/> -->
      </geometry>
      <material name="Blue"/>
    </visual>
    <collision>
      <origin rpy="1.57079632679 0 0" xyz="0 0.045 0"/>
      <geometry>
        <box size="0.044 0.02 0.09"/>
      </geometry>
    </collision>
    <inertial>
      <!-- CENTER OF MASS -->
      <origin rpy="1.57079632679 0 0" xyz="0 0.045 0"/>
      <mass value="0.124"/>
      <!-- box inertia: 1/12*m(y^2+z^2), ... -->
      <inertia ixx="0.00008783" ixy="0" ixz="0" iyy="1.037e-4" iyz="0" izz="2.414e-5"/>
    </inertial>
  </link>
  <joint name="leg_0_2_3" type="revolute">
    <parent link="leg_0_2"/>
    <child link="leg_0_3"/>
    <limit effort="1.961" lower="-2.0944" upper="2.0944" velocity="5.236"/>
    <origin rpy="0 0 0" xyz="0 0.09 0"/>
    <axis xyz="-1 0 0"/>
    <dynamics damping="0" friction="0"/>
  </joint>
  <link name="leg_0_3">
    <visual>
      <origin rpy="0.785 0 0" xyz="0 0.0441941738 -0.0441941738"/>
      <geometry>
        <box size="0.055 0.04 0.125"/>
        <!-- <cylinder length="0.125" radius="0.025"/> -->
      </geometry>
      <material name="Red"/>
    </visual>

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

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

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

  <link name="dummy_eef_0">
  <visual>
      <geometry>
        <sphere radius="0.005"/>
      </geometry>
      <material name="Green"/>
    </visual>
    <inertial>
      <!-- CENTER OF MASS -->
      <origin rpy="0 0 0" xyz="0 0 0"/>
      <mass value="0"/>
      <inertia ixx="0" ixy="0" ixz="0" iyy="0" iyz="0" izz="0"/>
    </inertial>
  </link>

  <joint name="body_leg_5" type="revolute">
    <parent link="base_link"/>
    <child link="leg_5_1"/>
```