Learning Navigation Skills for Legged Robots with Learned Robot Embeddings

Joanne Truong[1,*], Denis Yarats[2], Tianyu Li[2],
Franziska Meier[2], Sonia Chernova[1,2], Dhruv Batra[1,2], Akshara Rai[2]

***Abstract*— Recent work has shown results on learning navigation policies for idealized cylinder agents in simulation and transferring them to real wheeled robots. Deploying such navigation policies on legged robots can be challenging due to their complex dynamics, and the large dynamical difference between cylinder agents and legged systems. In this work, we learn hierarchical navigation policies that account for the low-level dynamics of legged robots, such as maximum speed, slipping, contacts, and learn to successfully navigate cluttered indoor environments. To enable transfer of policies learned in simulation to new legged robots and hardware, we learn dynamics-aware navigation policies across multiple robots with robot-specific embeddings. The learned embedding is optimized on new robots, while the rest of the policy is kept fixed, allowing for quick adaptation. We train our policies across three legged robots in simulation - 2 quadrupeds (A1, AlienGo) and a hexapod (Daisy). At test time, we study the performance of our learned policy on two new legged robots in simulation (Laikago, 4-legged Daisy), and one real-world quadrupedal robot (A1). Our experiments show that our learned policy can sample-efficiently generalize to previously unseen robots, and enable sim-to-real transfer of navigation policies for legged robots.**

I. Introduction

Legged robots such as quadrupeds from Boston Dynamics [1], Unitree [2] and hexapods from Hebi robotics [3] have emerged on the market as mature, robust, commercial robotic platforms. Legged robots are agile and can easily traverse uneven ground in homes like carpets and stairs, making them ideal for indoor navigation. At the same time, progress has been made in the field of learned indoor navigation using an egocentric camera input, without the use of maps [4], [5], showing real-world generalization of learned policies [6], [7]. However, learned visual navigation literature typically considers idealized virtual cylindrical agents, and does not take the dynamics of the robot into account. While such policies can be successful on wheeled robots, legged robots have significantly different dynamics than an idealized agent, or even other legged robots. These differences arise due to properties such as turning radius, motor strength, size and mass of the robot, etc. Fig. 1b shows an example where a PointGoal Navigation (PointNav) [8] policy that assumes an idealized agent is applied on the Daisy hexapod. While the policy plans to turn around an obstacle, the robot gets stuck, due to its larger size and turning radius, which were not accounted for by the PointNav policy. On the other hand, when the policy is trained directly on Daisy (Fig. 1c), and thus is aware of the dynamics of the robot, the policy starts turning sooner to accommodate the robot body and turning radius, leading to a successful navigation around the obstacle. Applying a policy that was learned on Daisy on another legged robot - the AlienGo quadruped (Fig. 1d) leads to the robot drifting away, and not reaching the goal in maximum allowed steps. This example highlights the importance of incorporating robot-specific dynamics when learning navigation policies for legged robots and the challenges of generalizing such learned policies to new robots. If navigation policies learned on one legged robot do not directly generalize to new systems, the experimental cost of learning navigation policies for legged robots can be prohibitively high, especially on hardware, reducing the overall scalability of such an approach. Learning navigation policies for complex legged robots, as well robustly transferring such learned policies to hardware and new robots are important challenges in the field of autonomous navigation.

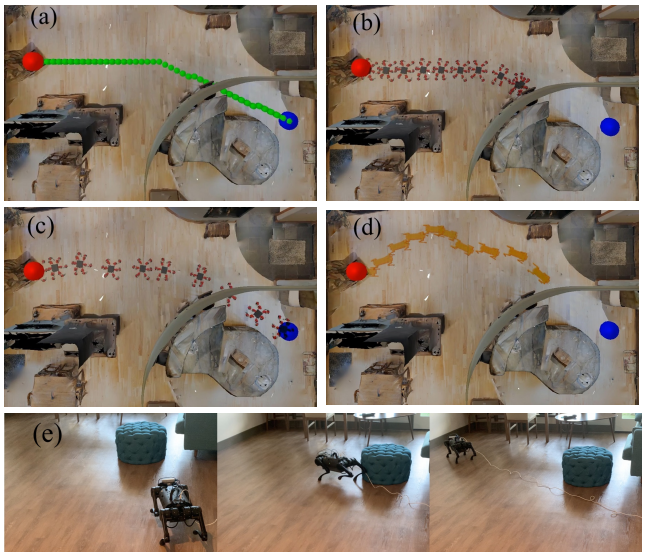

Fig. 1: Challenges of indoor navigation with legged robots. While an idealized spherical agent (a) can navigate to the goal with a PointNav policy, the same policy on Daisy (b) results in the robot getting stuck around an obstacle. A navigation policy trained on Daisy, with just egocentric images and no maps, can navigate to the goal successfully (c), but the same policy on the AlienGo quadruped (d) results in AlienGo drifting, and not reaching the goal in maximum allowed steps. We learn navigation policies across multiple legged robots and transfer to a real A1 robot (e).

In this work, we take a step towards learning visual navigation policies for legged robots, which use egocentric

*Work done while at FAIR
[1]Georgia Institute of Technology, {truong.j, dbatra, chernova}@gatech.edu
[2]Facebook AI Research {akshararai, fmeier, tianyul, denisy}@fb.com

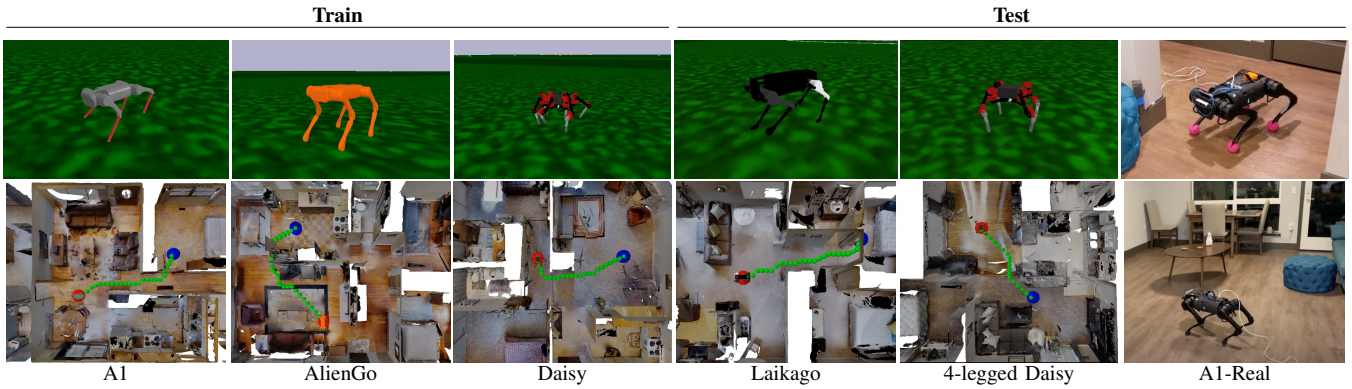

Fig. 2: Legged robots and environments used for training and testing. We train PointNav policies in cluttered environments on A1, AlienGo, and Daisy. At test-time, we study generalization to Laikago and 4-legged Daisy in simulation, and A1 in the real-world.

camera inputs and no maps, and generalize to hardware and new legged systems. We learn a navigation policy across three legged robots – two quadrupeds (A1, AlienGo) and one hexapod (Daisy), in simulated indoor home environments (Fig. 2), and test on previously unseen simulated legged robots (Laikago, 4-legged Daisy) and a real quadruped robot (A1-Real) in new environments. We develop hierarchical navigation policies that divide the control of the robot into a high-level policy that reasons about the center of mass motion and a low-level policy that converts high-level (velocity) commands into desired footsteps. The high-level policy consists of two components: (1) a universal navigation policy (shared across multiple robots) and (2) a learned robot-specific embedding used to specialize the universal policy per robot. During training, we collect data from multiple robots in separate environments which is collectively used to update a universal navigation policy, thereby improving sample-efficiency and leading to a policy that is inherently robust to variations in robot dynamics. Once trained, this high-level policy can be used on new legged robots in unseen environments, by searching in the learned embedding space, keeping the universal navigation policy fixed. To the best of our knowledge, this is the first work that (1) demonstrates learning navigation policies purely from visual input across multiple legged robots; and (2) generalizes learned policies to hardware and previously unseen simulated robots.

The core contribution of this work is a sample-efficient approach for teaching legged robots to navigate unknown indoor environments. Our simulation results are presented in the iGibson [9] photo-realistic simulation, with a Pybullet [10] physics engine. We also present zero-shot sim-to-real generalization hardware experiments on an A1 robot (Fig. 1e), where the robot navigates a new home using a policy learned purely in simulation.

II. Related work

Navigation in unknown environments has been widely studied in robotics. We mention some closely related works.

Indoor Navigation: Classical navigation approaches typically decompose the problem into mapping the environment, localizing in the map, and planning a path to the goal [11], [12], [13]. However, such approaches are highly dependent on the quality of the map. Mappers that only utilize RGB-D images are prone to failure in texture-less regions, and may require the use of expensive LiDAR sensors [14]. Incomplete maps with errors often lead to poor performance when used by planners [6]. Additionally, the map building process can be time-consuming, and require user-designed semantics for more robust navigation [15]. In this work, we use a learned policy that learns to extract relevant information from depth images, without explicitly building a map.

Recent works have shown learning-based approaches to be a robust at navigating unseen environments without explicit mapping (purely from egocentric RGB-D images and localization sensors) [4], [5], [6]. They can outperform traditional planning approaches [6], and achieve near-optimal performance in unknown houses [5]. However, these approaches have only been demonstrated on simple, cylindrical mobile base robots [7], and drones [16]. In contrast to these robots, legged robots have significantly more complex dynamics, arising from contacts and interactions between the robot and the environment, and policies learned in simulation do not directly generalize to hardware, due to the sim-to-real gap. In this work, we learn hierarchical navigation policies for legged robots that generalize to hardware and new robots.

RL for planning in legged robots: There is a significant body of work on teaching legged robots to navigate rough terrain, and robustly respond to disturbances in the environment [17], [18], [19], [20], [21]. Some of these works build policies that can navigate uneven terrain without any visual input [17], [21], [19], while others use LiDAR sensors or depth cameras to map the terrain, and use it to choose footstep locations [18], [22], [23]. However, these works do not address challenges associated with indoor navigation, where the robot has to navigate around obstacles, go through narrow hallways, and reach far away goals, without access to a map. In our experiments, the robot is placed in a new house that it has never seen before, and has to navigate to new goals using only onboard RGB-D cameras, and localization sensors, without a LiDAR, or a map. While previous work has used a LiDAR sensor for mapping the environment and using the map for navigating confined spaces [24], we only use an onboard RGB-D camera in our experiments, making our system cheaper and more scalable.

We diverge from classical planning literature [24], [25], [22] by not building maps. Instead, we use a reactive policy

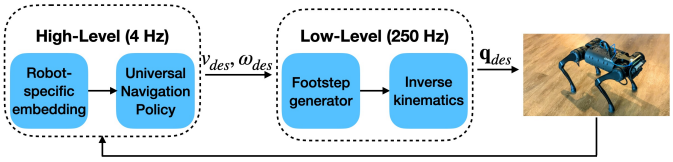

Fig. 3: A flowchart describing our hierarchical controller. The high-level policy, described in Section III-B, commands a desired CoM linear (v_{des}) and angular velocity (ω_{des}). The low-level controller generates a foot trajectory followed using Inverse Kinematics, which commands desired joint angles q_{des} to the robot.

trained using RL to reason about the space around the robot, avoid obstacles as it observes them, and navigate to a goal. The policy is trained in photo-realistic simulations of scans of different homes, and can be applied to a real robot operating in a new home. This helps with challenges such as building expensive maps [26], planning errors from inaccurate maps [27], and inflating obstacles for trajectory optimization [28].

Context-aware transfer learning: To adapt to new settings such as sim-to-real transfer, several works propose a context-aware learning setup, where a latent representation of context is learned over a large range of training tasks. At test time, the learned latent representation is used to infer the new context [29], [30], or optimized for generalization [31], [32], [33]. [31], [32] identify and encode dynamics parameters of a robot into a latent representation and learn a policy conditioned on this latent input. However, these approaches require expert knowledge to determine which dynamics parameters to encode, which can be difficult to choose when learning policies across robots of different morphologies. Instead, we propose to fully learn an embedding space for each robot, obviating the need for expert domain knowledge.

III. Dynamics-aware navigation policies

In this section, we introduce our approach which learns a dynamics-aware navigation policy that can enable sim-to-real transfer and generalize to unseen robots and environments.

A. Task: PointGoal Navigation

We consider the task of PointGoal Navigation (PointNav), as defined by [8], in which a robot is initialized in an unseen environment and must navigate to a goal location without being provided a map. An episode is considered successful if the robot can reach within 0.36m of the goal location (approximately half the width of our largest robot) in less than 150 robot steps. In our experiments, the robot is equipped with an egocentric depth sensor and an egomotion estimator for localization. The robot has access to the goal coordinates relative to the robot, specified as polar coodinates (r, θ), where r is the Euclidian distance to the goal, and θ is the relative angle to the goal. The environments consist of cluttered indoor spaces and the robot has to sense and reason about moving around obstacles to reach a randomly selected goal up to 7m away. For evaluation, we consider the success rate of our policy at reaching random new goals, as well as, Success inversely weighted by Path Length (SPL) [8] to measure the efficiency of the path taken.

B. Dynamics-aware navigation policy

Our dynamics-aware navigation policy consists of a two-layer hierarchical controller that divides the control of legged robots into a high-level policy that commands linear and angular center of mass (CoM) velocities, and a low-level policy that achieves these desired commands (Fig. 3).

High-level policy: The high-level policy consists of (1) a universal navigation policy shared across robots, and (2) a robot-specific embedding. The universal navigation policy takes as input the observed state s consisting of an egocentric depth image, the goal coordinates (in robot's reference frame), and a robot-specific embedding z (Section III-C). The output of the policy, or action a is a 2-dimensional vector containing the desired CoM linear (forward or backward) and angular (yaw) velocities (v_{des}, ω_{des}) for the robot to follow. Note that though our policy operates in a horizontal plane with 3 degrees of freedom (x-y and yaw), it only commands a 2-dimensional velocity vector, making the problem under-actuated and non-holonomic. To move laterally, the robot first needs to turn and then command a linear velocity along its new heading. This makes learning the high-level policy more challenging, but once learned, the policy is easier to generalize across multiple robot morphologies.

We use soft-actor critic (SAC) with an Autoencoder (AE) (SAC+AE) [34] as the off-policy training approach of our choice (training details described in Section III-E). However, unlike [34], we use egocentric depth images from the robot's camera, instead of a physically implausible floating 3rd-person camera. Since the camera is mounted on the robot, the image input changes as the robot moves through space, for example as the robot base tilts, the policy has to be robust to such dynamic camera movements.

Low-level policy: Our low-level policy converts the desired CoM velocities (v_{des}, ω_{des}) commanded by the high-level policy into desired footstep locations using an expert designed feedback policy, similar to [35], followed using inverse kinematics (IK). The stance legs maintain a fixed CoM height and achieve CoM velocity (v_{des}, ω_{des}) using joint position control in simulation, and whole-body MPC [36] on hardware.

One physical robot step takes $\Delta t = 0.25$s; during this time, the robot's foot moves from current location (x_{cur}, y_{cur}) to a desired foot placement location (x_{des}, y_{des}). Given (v_{des}, ω_{des}) from the high-level policy, we calculate the desired change in footstep position $(\delta x_f, \delta y_f)$ using the desired change in CoM position $\delta x_{com} = v_{des}\Delta t$ and orientation $\delta\gamma_{com} = \omega_{des}\Delta t$. For turning, the foot moves in a circle of radius r_f, changing the current angle between the foot and the robot heading γ_{cur} to $\gamma_{des} = \gamma_{cur} + \delta\gamma_{com}$ at the end of the step.

$$\delta x_f = \delta x_{com} + r_f \cdot (\cos(\gamma_{des}) - \cos(\gamma_{cur})) \quad (1)$$

$$\delta y_f = r_f \cdot (\sin(\gamma_{des}) - \sin(\gamma_{cur})) \quad (2)$$

$$x_{des} = x_{cur} + \delta x_f \quad y_{des} = y_{cur} + \delta y_f \quad (3)$$

where r_f is the distance between the foot and CoM.

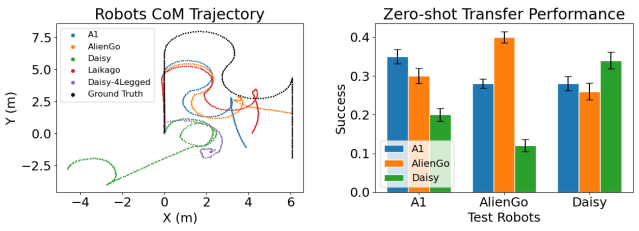

Fig. 4: (left): Center of Mass (CoM) trajectories for different robots following the same high-level commands. Due to the differences in the low-level controller, each robot ends up taking a different trajectory, with A1, AlienGo, Laikago very different from Daisy and 4-legged Daisy. (right) Zero-shot transfer of policies trained on a single robot onto other robots. Each robot's success rate is highest with a policy trained on itself, and success rate deteriorates when using a policy trained on a different robot.

Our footstep planner is shared across all our robots, including hexapods and quadrupeds. The footstep trajectory is then followed using robot-specific IK in swing, while stance legs maintain a fixed CoM height and desired velocity (v_{des}, ω_{des}). This enables us to experiment with 5 different legged robot designs, which is otherwise cumbersome due to robot-specific controllers. As our low-level policy structure is shared across multiple robots, it is not perfect at following the high-level commands for all robots. In fact, our robots can fall when commanded high velocities, and don't follow the commands well when there is a large jump in the desired velocities. Fig. 4a shows the CoM trajectories that different robots exhibit given the same high-level action sequence using our low-level controller. While A1, AlienGo and Laikago have similar trajectories in the start, they diverge later. On the other hand, Daisy and 4-legged Daisy have significantly different trajectories, and tend to turn to the left using our low-level controller. Fig. 4b shows the success rate of reaching new goals, when pairing different low-level controllers of each robot with high-level policies trained on other robots. Each robot achieves the highest success rate when using a high-level policy that was trained with its own low-level controller, and success rate deteriorates when using high-level policies trained on other robots; the largest drop is seen in AlienGo's performance – from 0.40 success down to 0.26 success when using its own policy, vs. Daisy policy, due to the largest dynamical shift in the robots. These results empirically confirm our hypothesis that each legged robot has different dynamics that need to be accounted for by the high-level policy in order to achieve good navigation performance, even when the low-level controllers share a common structure. In the next section, we will describe how to learn such a policy, while sharing data across robots.

C. Learning the high-level policy

To account for the differences between legged robots, we propose to learn a robot-specific embedding, along with a universal navigation policy across multiple robots. This allows for robot-specific specialization, while sharing data, and improved sample-efficiency.

We formulate our high-level policy π_θ to take as input both the observed state s and a latent embedding z_i per robot $i = 1, 2, \cdots N$, where N is the total number of robots used during training. The output action (desired CoM velocities) is a function of the state and the robot-specific embedding: $a \sim \pi_\theta(a \mid s, z_i)$. The critic is formulated similarly, where the predicted Q-value is a function of the state-action pair as well as z_i: $Q_\phi(s, a, z_i) = r(s, a) + \bar{V}(s', z_i)$. Q_ϕ is the critic, r is the reward, s' is the next state after taking action a, and $\bar{V}(s', z_i)$ is the robot-specific target value function [37].

Robot specific embeddings: We formulate the robot-specific embedding as a feed-forward network g with 3 layers and 100 hidden units, called the *z-network*. The 'input' to the z-network is a scalar, initially randomly sampled from a uniform distribution in $[0, 1)$, and learned alongside the parameters ψ of the z-network. Its output is a 1-dimensional latent embedding $z = g_\psi$, where g is the z-network MLP, and ψ are its learned parameters. We learn N z-networks for N robots, resulting in learned parameters $\psi_1, \psi_2, \cdots \psi_N$, and corresponding robot embeddings $z_1, z_2, \cdots z_N$.

Shared universal policy: We also train a shared actor π_θ and critic Q_ϕ across all robots. At the start of training, N robots are initialized in different environments, and tasked with navigating to a goal location. At every step, each robot receives an egocentric depth observation, the goal vector relative to its current position, and its current robot-specific embedding $z_1, \ldots, z_N$, which is added to individual replay buffers $\mathcal{D}_i = \{(s, a, r, s', z_i)\}$; combined to create a shared replay buffer $\mathcal{D} = \{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_N\}$.

Learning high-level navigation policy: The z-network parameters ψ are learned alongside the critic parameters ϕ, by minimizing $\mathcal{L}_{critic}$, a variant of the update from [37]:

$$\mathcal{L}_{critic} = \mathbb{E}_{(s,a,s',r,z\sim g_\psi)}\Big[Q_\phi(s, a, z) - \big(r + \bar{V}(s', z)\big)\Big]^2 \tag{4}$$

$$\psi_1^*, \ldots, \psi_N^*, \phi^* = \underset{\psi_1,\ldots,\psi_N,\phi}{\operatorname{argmin}} \; \mathcal{L}_{critic}(\mathcal{D}) \tag{5}$$

The partial derivative of $\mathcal{L}_{critic}$ with respect to ϕ uses samples from all robots, while when updating ψ_i the partial derivative w.r.t $\mathcal{D}_{j\neq i}$ is 0.

$$\frac{\partial \mathcal{L}_{critic}}{\partial \phi} = \frac{\partial}{\partial \phi}\mathbb{E}_{\mathcal{D}}\Big[Q_\phi(s, a, z) - \big(r + \gamma \bar{V}(s', z)\big)\Big]^2$$

$$\frac{\partial \mathcal{L}_{critic}}{\partial \psi_i} = \frac{\partial}{\partial \psi_i}\mathbb{E}_{\mathcal{D}_i}\Big[Q_\phi(s, a, z_i \sim g_{\psi_i}) - \big(r + \gamma \bar{V}(s', z_i)\big)\Big]^2$$

This results in an update law that pools data across multiple robots to update the critic parameters ϕ, but individual robot datasets $\mathcal{D}_i$ for updating robot-specific embedding z_i. The actor π_θ is updated using the common dataset $\mathcal{D}$, without updating robot embedding ψ_i, by minimizing KL divergence between π_θ and the predicted optimal Q-value, as in [37]. Our training pipeline is summarized in Figure 5.

In practice, we leverage Pytorch's `multiprocessing` to create parallel processes for multiple robots to collect experience in their own individual environments. By collecting data across multiple robots and environments for training both π_θ and Q_ϕ, our training is more sample-efficient than

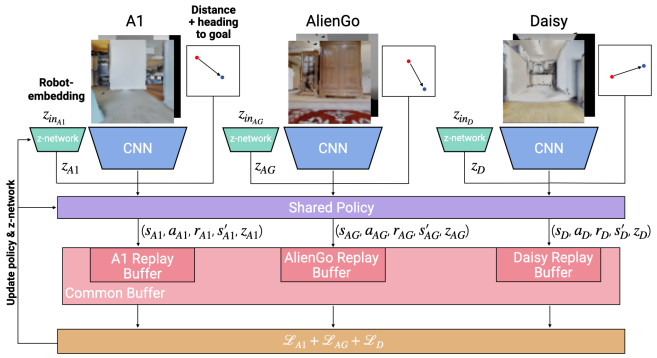

Fig. 5: We train a high-level navigation policy for A1, AlienGo, and Daisy, along with robot-specific embeddings. The robots receive an egocentric RGB or Depth observation, the goal relative to the robot's current position, and a robot-specific embedding. Each robot contributes to a replay buffer, used for learning robot embeddings, and all robot buffers contribute to the update of the shared policy.

training per robot, as well as robust to differences in robot dynamics and environments.

Learned embeddings have been explored in literature before, for example in [29], [20], [31], [30]. However, none of these works generalize to multiple robot morphologies or use a high-dimensional image input as state. [29], [31], [30] use dynamics randomization to create perturbed environments; [31] input the dynamics parameters to learn the embedding, while [29], [30] use robot state trajectories of past time steps as input. In contrast, we present a fully-learned embedding which does not need hand-defined dynamics features to learn an embedding for significantly different robot morphologies. We also compare our approach against [29] and [31] in Section IV, and show that we outperform both.

Learning a shared high-level policy allows robots to share data, significantly improving the sample-efficiency of learning. Our hierarchical controller makes training of the high-level policy even more expensive as for every action commanded by the high-level policy, the low-level policy takes multiple simulation steps to achieve the desired command. Hence, the number of the data points collected per high-level action is much smaller than the actual simulation steps. By pooling data across N robots, we effectively get N times as much data as we would if we collected data per robot. Our experiments show that our hierarchical structure with off-policy learning can learn navigation policies for multiple legged robots, and generalize to unseen robots and hardware.

Generalization to unseen robots. Once learned, our navigation policies can be adapted to new robots by searching in the space of the learned 1-dimensional embedding z_i which is input to the policy π_θ. We aim to find a robot embedding z^*_{test} that maximizes the long-term reward over trajectories τ of length T induced on the test platform, using the learned policy $\pi_\theta(\cdot, z^*_{test})$: $z^*_{test} = \text{argmax}_{z_{test}} \mathbb{E}_\tau[\sum_{t=1}^{T} r_t]$. We conduct a grid-search over z_{test} in the interval $[-1, 1]$ to find the globally-optimal z^*_{test} (within discretization error) in the space of learned embeddings. Approximate techniques like [31], [38] can be used for higher dimensional z.

For successful navigation on new robots, the learned embedding should capture different properties that the robots demonstrate. For example, some robots might be slower at turning while others might be faster. We expect different values of z to result in different motions of robots around obstacles, etc. Fig. 6 illustrates CoM trajectories in an environment with obstacles for different values of z. For $z = -0.7$, we see that 4-legged Daisy takes a relatively direct path to the goal, while for other z values, the robot gets stuck on obstacles.

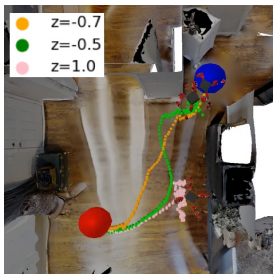

Fig. 6: Different values of z result in different CoM trajectories.

D. Reward function

Our reward function incentivizes path efficiency and safety of the robot:

$$r_t(a_t, s_t) = R_{geo} + R_{coll} + r_{fall} + r_{success} \quad (6)$$

$$R_{geo} = \Delta_{geo\_dist} \cdot k_{geo\_dist},\ R_{coll} = -1.0 \text{ if collision} \quad (7)$$

$$r_{fall} = -5.0 \text{ if fall}, \quad r_{success} = 10.0 \text{ if success} \quad (8)$$

R_{geo} is the change in geodesic distance to the goal (shortest obstacle-free path to goal) from the previous step to the current step, scaled by a geodesic reward weight k_{geo_dist}, a hyperparameter to encourage efficient paths. R_{coll} penalizes the robot if there was a collision with the environment. r_{fall} is the penalty for falling, applied if the robot falls in the step, at which point the episode is terminated. $r_{success}$ is the terminal reward given to the robot, if it reaches the goal.

E. Training details

We use SAC+AE, a robust and sample-efficient method for learning from high-dimensional images for training our navigation policy and robot embeddings. The actor, which serves as the universal policy, consists of a 3-layer multi-layer perceptron (MLP) with a 1024-dimensional hidden layer, and outputs a vector of means and a covariance matrix that parameterize a Gaussian action distribution. The Q-function is represented by a 3-layer MLP with 1024 units.

A convolutional autoencoder is used to provide the policy with visual features extracted from the input depth image. The encoder consists of 4 convolutional layers that map the input depth image to a 50-dimensional latent representation. The decoder consists of a fully connected layer followed by 4 deconvolutional layers that reconstructs the latent representation back to the original image. This reconstruction loss is used as an auxiliary objective during training; it improves sample efficiency and aids in training stability [34].

IV. Experimental Evaluation

We evaluate our proposed approach on two experimental setups: a set of cart-pole variants, and a set of legged robots. In both cases, we aim to learn generalizable policies on certain morphologies and test on new morphologies that were not seen during training. We first choose cart-pole environments as a simplified, but illustrative example of our framework, and compare with other approaches from

literature. Our results show that our approach is faster at learning on a set of cart-poles, and better at generalization to unseen cart-poles than other approaches, like [29], [31].

Next, we tackle the much more challenging task of learning navigation policies on legged robots, and generalization to unseen robots. Our experiments on legged robots show that our approach can learn navigation policies across multiple legged robots, and generalize to hardware and unseen robots sample-efficiently. It outperforms baselines from literature like [31], [29], ablations of our approach, and an oracle point-agent policy (without dynamics) that uses A* on a map.

A. Experimental setup

We compare our approach (**Learned-z**) to the following:
– RL^2 with DDPG (**Meta-RL**): We use RL^2 [29] as a baseline that learns robot-specific contexts during training. At test time, context is inferred from state trajectories on the test robot. This baseline compares our learned robot-specific embedding to meta-learned embeddings from literature.
– Informed embedding (**Informed-z**): [31] learns a robot-specific embedding by providing the z-network with the robot's dynamics parameters. At test time, z is optimized for the new robot, similar to our approach. This experiment compares our learned embedding, which does not require any prior knowledge of the dynamics parameters, to an informed embedding that has access to more privileged information.
– Semi-Informed embedding (**Semi-Informed-z**): In this experiment, we only give a sub-set of the varied dynamics parameters as inputs to the z-network during training (mass and offset of the cart-poles). Deciding the 'right' input parameters can be challenging, especially when working with different robot morphologies. This experiment shows the sensitivity of [31] to an incorrect set of parameters.
– Fixed robot embedding (**Fixed-z**): In this ablation of our approach, we fix z during training, giving each robot a distinct value. At test time, z is then optimized for the new robot, similar to our approach. This experiment demonstrates that a learned embedding captures the landscape of the robot's dynamics better than fixed initialized embeddings.
– No robot-specific embedding (**No-z**): In this ablation of our approach, we learn a shared universal policy across different environments, with no robot-specific embeddings. At test time, the whole policy is fine-tuned on the new robot. This experiment demonstrates the need for robot-specific embeddings for sample-efficient generalization.

B. Cart-pole experiments

We create a family of cart-poles in Mujoco [39] by changing the mass of cart, length of pole, and offset of pole from center (Fig. 7, Table I). Test robots consist of one cart-pole that lies 'between' the training cart-poles (CP5) and one that lies outside of the training distribution (CP4). Our aim is to test generalization of all approaches to both in and out-of-distribution robots.

We visualize the learned latent space by sweeping over different values of z and measuring performance (Fig. 7, bottom). All train cart-poles have a singular maximum, near the learned z, except CP1. Visualizing the latent space also shows that CP4 is close to CP3 in learned latent space, but out of distribution from train robots, while CP5 is between CP2 and CP3, as expected. This demonstrates that the learned embedding has captured dynamics properties of the robots.

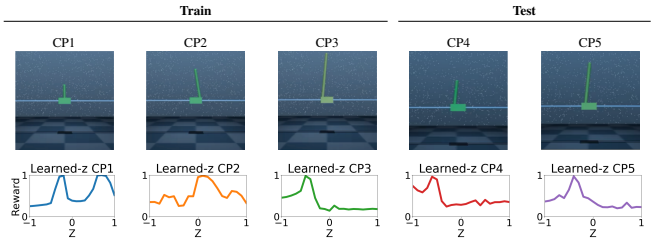

Fig. 7: (Top) Cart-pole robots used for training and testing. (Bottom) Performance of cart-poles for different z in the learned latent space.

TABLE I: We create 5 variants of cart-poles with dynamics parameters shown below. We train on CP1-3, and test on CP4-5.

Parameter	CP1	CP2	CP3	CP4	CP5
Mass	0.1	1	2	1.5	1.5
Pole Length	0.5	1	1.5	0.75	1.25
Pole Offset	0	0.15	-0.15	-0.1	-0.1

The training curves of our cart-pole experiments on CP3 are shown in Fig. 8, and Table II condenses results for all cart-poles. **Meta-RL** does not learn to control CP3, and performs poorly on test cart-poles CP4 and CP5 (0.66 reward on CP4 and 0.83 reward on CP5), showing that **Meta-RL** is unable to generalize to new unseen robots, with poorer performance on CP4 which is out of the training distribution.

Informed-z is given the dynamics parameters of the robots (Table I) during training, yet we notice that **Learned-z** generalizes to test robot CP4 better (0.94 using **Learned-z** vs. 0.79 using **Informed-z**). This indicates that for **Informed-z**, the information of dynamics parameters deteriorates generalization to out-of-distribution robots. **Semi-Informed-z** performs poorly at training and test robots, highlighting the sensitivity of [31] to the 'right' set of dynamics parameters.

Among the ablation experiments of our approach, we observe that **Learned-z** generalizes better to the two test robots than **Fixed-z** and **No-z**. While both **Learned-z** and **Fixed-z** learn to control the training cart-poles, generalization to CP4 is improved when using learned robot embeddings (0.94 with **Learned-z** vs 0.77 with **Fixed-z** and 0.68 with **No-z**). **Informed-z** and **Fixed-z** are close to **Learned-z** in generalization to CP5, but **Learned-z** performs best when testing on out-of-distribution robot (CP4).

C. Simulation legged robot experiments

Next, we present experiments on legged robots in simulation (Fig. 9). In this setting, we use the hierarchical control

TABLE II: Cart-pole Evaluation (Episode reward)

	Train			Test	
Experiment	CP1	CP2	CP3	CP4	CP5
Semi-Informed-z	0.97$\pm$0.02	0.74$\pm$0.03	0.71$\pm$0.04	0.42$\pm$0.04	0.75$\pm$0.01
Meta-RL	0.99$\pm$0.01	0.93$\pm$0.02	0.70$\pm$0.09	0.66$\pm$0.11	0.83$\pm$0.06
Informed-z	0.99$\pm$0.01	0.90$\pm$0.04	0.93$\pm$0.02	0.79$\pm$0.16	0.94$\pm$0.04
No-z	0.99$\pm$0.01	0.76$\pm$0.08	0.80$\pm$0.05	0.68$\pm$0.10	0.85$\pm$0.06
Fixed-z	0.99$\pm$0.02	0.86$\pm$0.13	0.87$\pm$0.19	0.77$\pm$0.21	**0.97$\pm$0.03**
Learned-z	**1.00$\pm$0.00**	**0.98$\pm$0.00**	**0.98$\pm$0.01**	**0.94$\pm$0.03**	**0.97$\pm$0.03**

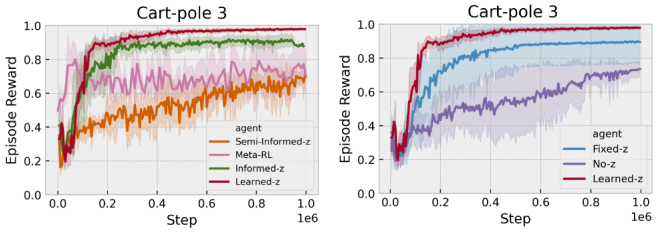

Fig. 8: Learning speed of different approaches, visualized on CP3. Our approach learns faster, and achieves a higher performance than all the other approaches. (Left) Comparison against context-aware approaches; (Right) Ablations of our approach.

structure described in Section III-B and learn a high-level navigation policy. Robots are trained from scratch in the iGibson [9] simulator for a cumulative 1 million simulation steps across all robots. For all approaches, we train on 3 train robots (A1, Aliengo, Daisy), and evaluate generalization to 2 new robots (Laikago, Daisy-4-legged) in simulation. To compare the different approaches, we report the mean and standard deviation of SPL [8] over 5 random seeds when the high-level policy is applied to both train and test robots.

TABLE III: Legged robots' evaluation performance. Learned-z achieves the highest SPL when tested on previously unseen robots.

	Train			Test	
Experiment	**A1**	**AlienGo**	**Daisy**	**Laikago**	**Daisy4**
A* policy	0.49±0.07	0.45±0.15	0.22±0.10	0.38±0.15	0.18±0.06
Meta-RL	0.08±0.06	0.14±0.12	0.03±0.06	0.02±0.03	0.07±0.06
Informed-z	0.57±0.06	0.52±0.05	0.43±0.09	0.37±0.11	0.40±0.06
No-z	0.48±0.12	0.42±0.05	0.33±0.07	0.33±0.13	0.22±0.08
Fixed-z	**0.59±0.07**	**0.59±0.14**	0.40±0.10	0.47±0.09	0.38±0.11
Learned-z	**0.59±0.07**	0.54±0.04	**0.48±0.14**	**0.48±0.08**	**0.49±0.08**

Zero-shot transfer of A* policies without dynamics information: We create an oracle point-based policy which uses a map of the environment to find a near-optimal collision-free path to goal. The policy samples a path to goal using A*, with near perfect performance (0.92 SPL) on an idealized cylinder agent. However, when applied on the legged robots, the performance drops significantly (Table III), with the most significant drop seen on 4-legged Daisy (0.18 SPL). This is because the oracle policy does not account for robot dynamics, and the robots often fall, get stuck around an obstacle, or exceed maximum allowed steps before reaching the goal. This emphasizes that knowledge of the low-level dynamics of legged robots is crucial for effective navigation.

Context-aware comparisons: Table III shows that relative to other context-aware approaches, our approach is able to achieve a higher SPL on the test robots while retaining high SPL on the train robots. **Meta-RL** achieves poor performance on all robots (<0.15 SPL), with the robots falling over in most episodes. This result highlights the difficulty in generalizing **Meta-RL** to more complex dynamical systems such as legged robots. Although **Informed-z** is given the dynamics parameters of the robots (Table IV) during training, it does not generalize to Laikago and 4-legged Daisy as well as **Learned-z** (0.37 SPL vs. 0.48 SPL for Laikago, 0.40 SPL vs. 0.49 SPL for 4-legged Daisy). This illustrates the difficulty in selecting the 'right' set of dynamics parameters

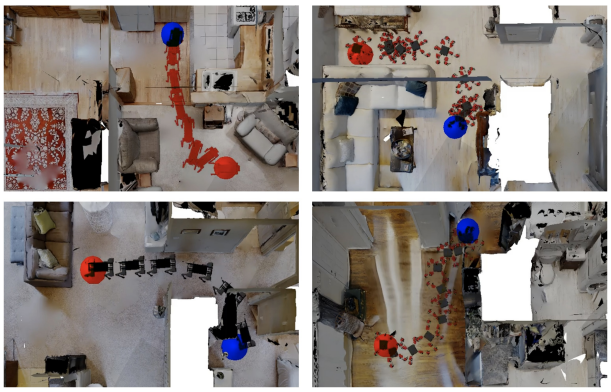

Fig. 9: Our approach learns policies capable of successfully navigating on different train (top) and test (bottom) robots.

across robots of different morphologies.

Learned-z and **Fixed-z** slightly outperform **No-z**. However, on generalization robots we see an improvement when using **Learned-z** especially on 4-legged Daisy (0.49 SPL with **Learned-z** vs 0.38 SPL with **Fixed-z** and 0.22 SPL with **No-z**). Since 4-legged Daisy is the most 'different' robot from the train robots, we again observe that our approach performs best at out-of-distribution generalization.

TABLE IV: Dynamics parameters of the robots used as input to the z-network for **Informed-z**.

Parameter	**A1**	**AlienGo**	**Daisy**	**Laikago**	**Daisy4**
Mass (kg)	12.46	20.64	24.76	20.74	17.62
Leg Length (m)	0.4	0.5	0.45	0.5	0.54
Number of Legs	4	4	6	4	4

D. Hardware experiments

Finally, we transfer our learned navigation policy from simulation to a real A1 quadrupedal robot. The robot is equipped with a RealSense depth and tracking camera for capturing depth images of the environment, as well as localizing itself with respect to its start location. The goal is defined with respect to the start point of the robot, and transformed to the current robot position at each robot step.

On hardware, we do not optimize z, and instead use the learned z from simulation for the A1 robot. The high-level policy operates at 4Hz, taking as input the depth camera image as seen by the robot, relative goal location, and learned z from simulation. It commands a desired CoM velocity to the low-level controller, which operates at 250Hz. The stance controller for the A1 robot is a whole-body MPC controller, similar to [31], and the swing legs use IK (Section III-B).

Our hardware experiments show the A1 robot moving through hallways to reach the next room, and avoiding obstacles to reach a goal (Figure 10). These experiments show that our proposed approach can learn robust high-level navigation policies that can transfer from simulation to real world, without any fine-tuning. During our experiments, we noticed that the A1 robot would turn very close to obstacles, and sometimes hit corners. This was caused by the hardware having different dynamics than the simulation, for example, the real robot turns slower than the simulation, and hence can hit obstacles even when the policy planned

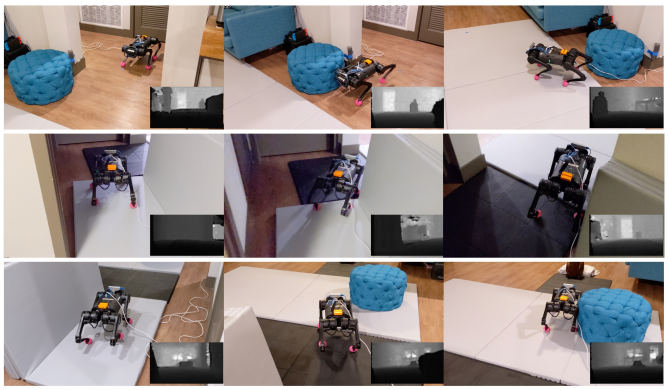

Fig. 10: Snapshots of the A1 robot navigating a new home. The robot goes through hallways, and avoids obstacles to reach goals. Depth images, as seen by the robot, are shown in the bottom right corner of each frame.

not to. This performance can be improved by fine-tuning the learned embedding on hardware, and searching for a z that results in wider motions on the real robot. Sample-efficiently optimizing z that can generalize to multiple goals on hardware is challenging, and we leave this to future work.

V. Conclusion

In this work, we present a framework for learning hierarchical navigation policies for legged robots. Our policy shares data across multiple robots during learning, and learns a high-level navigation policy with a shared universal navigation policy across robots, and robot-specific embeddings, used for specialization. The learned embedding captures different dynamical properties, such as turning radius and walking speed of different robots, and shows promising results on hardware, and robots that were never seen during training. This work opens up future avenues for large-scale research on legged platforms for indoor navigation.

VI. Acknowledgements

The Georgia Tech effort was supported in part by NSF, AFRL, DARPA, ONR YIPs, ARO PECASE. JT was supported by an NSF Graduate Research Fellowship under Grant No. DGE-1650044 and a Google Women Techmaker's Fellowship. The views and conclusions are those of the authors and should not be interpreted as representing the U.S. Government, or any sponsor.

License for dataset used Gibson Database of Spaces. License at `http://svl.stanford.edu/gibson2/assets/GDS_agreement.pdf`

References

[1] "Boston dynamics," https://www.bostondynamics.com/spot.
[2] "Unitree robotics," https://www.unitree.com/, accessed: 2020-10-10.
[3] "Hebi robotics," https://www.hebirobotics.com/robotic-kits.
[4] M. Savva, A. Kadian, O. Maksymets, Y. Zhao, E. Wijmans, *et al.*, "Habitat: A Platform for Embodied AI Research," in *ICCV*, 2019.
[5] E. Wijmans, A. Kadian, A. Morcos, S. Lee, I. Essa, *et al.*, "Dd-ppo: Learning near-perfect pointgoal navigators from 2.5 billion frames," *arXiv*, pp. arXiv–1911, 2019.
[6] S. Bansal, V. Tolani, S. Gupta, J. Malik, and C. Tomlin, "Combining optimal control and learning for visual navigation in novel environments," in *Conference on Robot Learning*. PMLR, 2020, pp. 420–429.
[7] A. Kadian, J. Truong, A. Gokaslan, A. Clegg, E. Wijmans, *et al.*, "Sim2real Predictivity: Does Evaluation in Simulation Predict Real-World Performance?" *IEEE RAL*, 2020.
[8] P. Anderson, A. Chang, D. S. Chaplot, A. Dosovitskiy, S. Gupta, *et al.*, "On Evaluation of Embodied Navigation Agents," *arXiv preprint arXiv:1807.06757*, 2018.
[9] F. Xia, W. B. Shen, C. Li, P. Kasimbeg, M. E. Tchapmi, *et al.*, "Interactive gibson benchmark: A benchmark for interactive navigation in cluttered environments," *IEEE RAL*, 2020.
[10] E. Coumans and Y. Bai, "Pybullet," http://pybullet.org.
[11] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping," *IEEE robotics & automation magazine*, 2006.
[12] S. Thrun, "Probabilistic robotics," *ACM*, 2002.
[13] J. Fuentes-Pacheco, J. Ruiz-Ascencio, and J. M. Rendón-Mancha, "Visual simultaneous localization and mapping: a survey," *Artificial intelligence review*, vol. 43, no. 1, pp. 55–81, 2015.
[14] W. Hess, D. Kohler, H. Rapp, and D. Andor, "Real-time loop closure in 2d lidar slam," in *ICRA*, 2016.
[15] S. L. Bowman, N. Atanasov, K. Daniilidis, and G. J. Pappas, "Probabilistic data association for semantic slam," in *ICRA*, 2017.
[16] F. Sadeghi and S. Levine, "Cad2rl: Real single-image flight without a single real image," *arXiv preprint arXiv:1611.04201*, 2016.
[17] W. C. Martin, A. Wu, and H. Geyer, "Experimental evaluation of deadbeat running on the atrias biped," *IEEE RAL*, 2017.
[18] S. Feng, E. Whitman, X. Xinjilefu, and C. G. Atkeson, "Optimization-based full body control for the darpa robotics challenge," *JFR*, 2015.
[19] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning quadrupedal locomotion over challenging terrain," *Science Robotics*, 2020.
[20] T. Li, R. Calandra, D. Pathak, Y. Tian, F. Meier, and A. Rai, "Planning in learned latent action spaces for generalizable legged locomotion," *arXiv preprint arXiv:2008.11867*, 2020.
[21] C. Hubicki, A. Abate, P. Clary, S. Rezazadeh, M. Jones, *et al.*, "Walking and running with passive compliance: Lessons from engineering: A live demonstration of the atrias biped," 2018.
[22] D. Kim, D. Carballo, J. Di Carlo, B. Katz, G. Bledt, *et al.*, "Vision aided dynamic exploration of unstructured terrain with a small-scale quadruped robot," in *ICRA*, 2020.
[23] V. Tsounis, M. Alge, J. Lee, F. Farshidian, and M. Hutter, "Deepgait: Planning and control of quadrupedal gaits using deep reinforcement learning," *IEEE RAL*, 2020.
[24] R. Buchanan, L. Wellhausen, M. Bjelonic, T. Bandyopadhyay, N. Kottege, and M. Hutter, "Perceptive whole-body planning for multilegged robots in confined spaces," *Journal of Field Robotics*, vol. 38, 2021.
[25] P. Fankhauser, M. Bloesch, and M. Hutter, "Probabilistic terrain mapping for mobile robots with uncertain localization," *RAL*, 2018.
[26] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "Octomap: An efficient probabilistic 3d mapping framework based on octrees," *Autonomous robots*, vol. 34, no. 3, 2013.
[27] A. A. Al Makdah, V. Katewa, and F. Pasqualetti, "Accuracy prevents robustness in perception-based control," in *ACC 2020*.
[28] M. X. Grey, A. D. Ames, and C. K. Liu, "Footstep and motion planning in semi-unstructured environments using randomized possibility graphs," in *ICRA 2017*.
[29] Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel, "Rl ^{2}: Fast reinforcement learning via slow reinforcement learning," *arXiv preprint arXiv:1611.02779*, 2016.
[30] K. Rakelly, A. Zhou, C. Finn, S. Levine, and D. Quillen, "Efficient off-policy meta-reinforcement learning via probabilistic context variables," in *International conference on machine learning*, 2019.
[31] X. B. Peng, E. Coumans, T. Zhang, T.-W. Lee, J. Tan, and S. Levine, "Learning agile robotic locomotion skills by imitating animals," *arXiv preprint arXiv:2004.00784*, 2020.
[32] W. Yu, V. C. Kumar, G. Turk, and C. K. Liu, "Sim-to-real Transfer for Biped Locomotion," *arXiv preprint arXiv:1903.01390*, 2019.
[33] W. Yu, J. Tan, C. K. Liu, and G. Turk, "Preparing for the unknown: Learning a universal policy with online system identification," *arXiv preprint arXiv:1702.02453*, 2017.
[34] D. Yarats, A. Zhang, I. Kostrikov, B. Amos, J. Pineau, and R. Fergus, "Improving sample efficiency in model-free reinforcement learning from images," *CoRR*, vol. abs/1910.01741, 2019.
[35] M. H. Raibert, *Legged robots that balance*. MIT press, 1986.
[36] D. Kim, J. Di Carlo, B. Katz, G. Bledt, and S. Kim, "Highly dynamic quadruped locomotion via whole-body impulse control and model predictive control," *arXiv preprint arXiv:1909.06586*, 2019.
[37] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, *et al.*, "Soft actor-critic algorithms and applications," *arXiv preprint arXiv:1812.05905*, 2018.
[38] W. Yu, J. Tan, Y. Bai, E. Coumans, and S. Ha, "Learning fast adaptation with meta strategy optimization," *IEEE RAL*, 2020.
[39] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," *IROS*, 2012.