Appendix

1 Robot Details

Table 1: Robot specific parameters used for training and evaluation. The maximum number of steps and velocity limits for each robots are set in proportion to the robot's leg length.

	Parameter		A1		Aliengo		Spot
1	Success radius (m)		0.24		0.32		0.425
2	Maximum number of steps		326		268		150
3	Linear velocity limits (m/s)	$\pm$	0.23	$\pm$	0.28	$\pm$	0.50
4	Angular velocity limits (rad/s)	$\pm$	0.14	$\pm$	0.17	$\pm$	0.3
5	Leg length (m)		0.2		0.25		0.44

2 Additional Evaluation Results

We present additional results using the Raibert controller for evaluation in Figure 1 (row 4). The policies are evaluated across 3 seeds, using the HM3D + Gibson validation split which consists of 1,100 episodes from 110 unique scenes. Our results are consistent with evaluation using the MPC controller– kinematic trained policies still outperform the dynamic trained policies, *even when evaluated using dynamic control* [1] (68.9 % SR for Aliengo in Habitat, Kinematic vs. 45.4 % SR in Habitat, Dynamic, Fig. 1, middle).

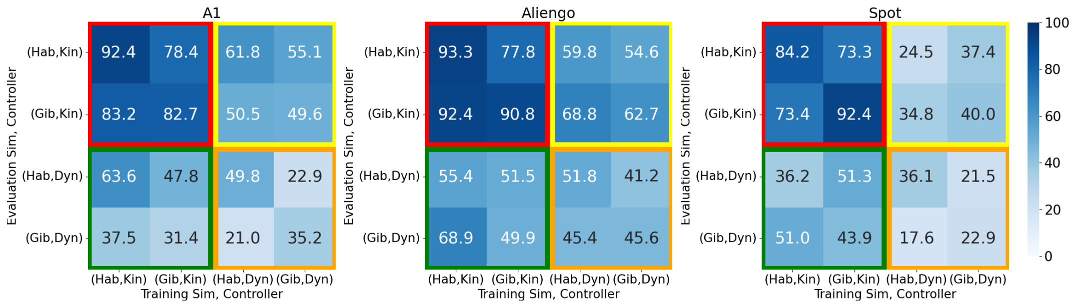

Figure 1: Average success rates for sim2sim and kinematic2dynamic transfer for A1, Aliengo and Spot. Dynamic evaluation in iGibson is performed using the Raibert controller [1]. We see that the kinematic trained policies still perform the best overall (red quadrants), and also often outperforms the dynamic trained policies, even when evaluated using dynamic control (green quadrants vs. orange quadrants).

3 Actuation Noise Modeling Details

We collect actuation noise (difference between the commanded and true velocity of the robot) on the Boston Dynamics Spot robot by commanding the robot at a random velocity for 1Hz in an empty room and measuring the final velocity. Noise is collected in a decoupled and coupled manner described below:

1. **Decoupled**: Random velocities ($\sim \mathcal{U}(-0.5, 0.5)$) are commanded in the forward, lateral, and angular directions *separately*. When collecting data for the forward direction, the side-

[1]For all robots and training sim/controller except A1, iGibson-Kinematic

ways direction velocity is commanded zero velocity; the opposite is true when collecting data for the sideways direction. We collect 2,000 datapoints for each direction.

2. **Coupled**: Random velocities ($\sim \mathcal{U}(-0.5, 0.5)$) are commanded in the forward, lateral, and angular directions *at the same time*.

Each dataset contains 6,000 data points, with decoupled data containing 2000 data points for each direction. We choose to model the uncertainty in the robot's actuation with a standard bivariate Gaussian with a diagonal variance similar to [2]. The collected data is used to generate mean and variance parameters for a Gaussian distribution describing the noise in each dimension, as shown in Table 2. The Gaussian models are then used to inject noise into the the kinematic simulation during training time through the following method: 1) the policy predicts a velocity, 2) the Gaussian distributions for each direction are sampled, 3) the sampled noise is added to the policy's predicted velocity, and 4) the robot's state is updated according to the noisy velocity.

The two different noise collection approaches aim to study the effects data collection has on the resulting noise model. Our experiments show that both noise models perform better in the real-world than no noise modeling, and coupled noise performs slightly better than decoupled.

Noise	μ_x (m/s)	μ_y (m/s)	μ_ω (deg/s)	σ_x (m/s)	σ_y (m/s)	σ_ω (deg/s)
Coupled	0.002	-0.004	0.081	0.054	0.065	2.599
Decoupled	0.002	-0.001	-0.029	0.036	0.044	1.468

Table 2: We fit a bivariate Gaussian to the actuation noise collected on a real Spot robot. During kinematic training, we sample from the noise models and inject the realistic actuation noise to the robot's desired final state.

It is also important to note that while Spot (and other legged robots) can move in all directions, these robots are not necessarily omnidirectional platforms, since they cannot move in all directions *equally well*. To illustrate this, we collect displacement errors on Spot in forward and lateral direc-

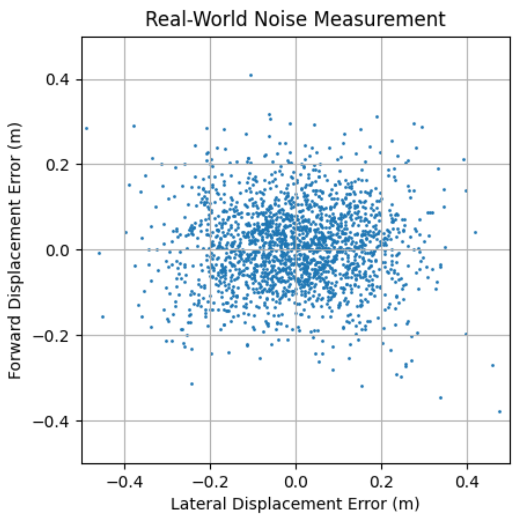

Figure 2: We collect displacement errors on Spot in forward and lateral directions. The standard deviation for the displacement errors between the forward and lateral directions are large and asymmetric, demonstrating that the robot is not perfectly omnidirectional.

tions while commanding random desired CoM movements (Figure 2). If the robot were perfectly omnidirectional, we would expect the means and variances for the forward and lateral direction to be small and the same. While the mean error in both directions is close to 0, the standard deviations in the forward and lateral directions are significantly larger and asymmetric. In the forward direction, the standard deviation is 0.097 meters, and in the lateral direction it is 0.139 meters. This tracking error, which increases with commanded velocity, motivated the choice of saturating commanded desired velocity at 0.5 m/s. This behavior is observed on the Spot robot using Boston Dynamics walking controllers, which is a very good, highly tuned controller for the robot. We would expect any open-sourced controller which is not tuned for a particular robot to only be worse.

4 Additional Low-level Controller Details

We use two different kinds of low-level controllers in our work– an expert-designed Raibert controller from [1] (modified to allow for lateral movement), and a model-predictive control (MPC) controller from [3]. The Raibert controller takes in desired CoM velocities $(v_{x\_des}, v_{y\_des}, \omega_{des})$ from the high-level policy to calculate the desired foot placement location, following equations 1-3 from [1]. The footstep trajectory is followed using inverse kinematics. The MPC controller uses a contact schedule to determine each leg's contact state and compute the optimal joint torque for each leg.

5 Additional Policy Details

5.1 High-level policy parameters.

We use PPO with Generalized Advantage Estimation (GAE). We use a discount factor of 0.99, and GAE parameter of 0.95. We use the Adam optimizer, with a learning rate of 2.5e-4. We run 8 agents in parallel (in different environments) per GPU, and each agent collects a rollout of 128 frames of experience. We use 8 GPUs, for a total of 64 parallel workers.

5.2 Reward function.

Our reward function is derived from [1], with an added penalty for backward velocities, which can lead to collisions and hurts performance. Specifically, our reward function is defined as:

$$r_t(a_t, s_t) = R_{geo} + R_{coll} + R_{fall} + R_{success} + R_{slack} + R_{backward} \tag{1}$$

R_{geo} is a shaped reward, denoting the change in geodesic distance to the goal between two timesteps.
R_{coll} is a penalty for collisions. We set the collision penalty to -0.03.
R_{fall} is a penalty if the robot falls over. We set the falling penalty to -5.0, and terminate the episode.
$R_{success}$ is the terminal reward for completing the episode. We set the terminal reward to 10.0.
R_{slack} is a slack penalty used to encourage the robot to reach the goal as fast as possible. We set the slack penalty to -0.002.
$R_{backward}$ is a penalty for moving backwards, as moving backwards can lead to collisions. We set the backwards penalty to -0.03.

5.3 Dynamic Simulation Overfitting Details.

We define overfitting as the drop in performance when testing on a different controller and/or simulator than training. This is a natural generalization of the standard definition of overfitting in supervised learning (accuracy on IID training dataset - accuracy on IID testing dataset). We train dynamic policies for all three robots to congergence (Figure **??**)
In Table 3, we show the success rate on Habitat-Dynamic (training scenario) - success rate on iGibson-Dynamic (testing scenario) for all 3 robots. Note that these are all evaluations on the same houses/scenes/environments (from a held-out evaluation set) and the only factor changing is the simulator. We can clearly see that the gap is always positive, indicating that policies trained on Habitat-Dynamic perform worse when evaluated on iGibson-Dynamic compared to evaluation on Habitat-Dynamic. As can be expected, in all but one case, the performance gaps are increasing with more RL training, though this is not strictly necessary. A well-trained high-level policy can learn to reason intelligently about navigation (even with dynamic controllers), and then perform well across simulators.

Robot	Steps of experience	Performance gap (%)
A1	12M	7.8
	25M	26.7
	50M	27.2
AlienGo	12M	5.7
	25M	5.4
	50M	8.1
Spot	12M	14.0
	25M	21.6
	50M	15.6

Table 3: We measure the performance gap for dynamic policies between evaluations in Habiat-Dynamic and iGibson-Dynamic. The gap is always postive, and in all cases but one, the performance gaps increase with more RL training, demonstrating that the dynamic policies overfit to the simulator and controller it was trained on.

References

[1] J. Truong, D. Yarats, T. Li, F. Meier, S. Chernova, D. Batra, and A. Rai. Learning navigation skills for legged robots with learned robot embeddings. In *International Conference on Intelligent Robots and Systems (IROS)*, 2020.

[2] A. Murali, T. Chen, K. V. Alwala, D. Gandhi, L. Pinto, S. Gupta, and A. Gupta. Pyrobot: An open-source robotics framework for research and benchmarking. *arXiv preprint arXiv:1906.08236*, 2019.

[3] X. B. Peng, E. Coumans, T. Zhang, T.-W. Lee, J. Tan, and S. Levine. Learning agile robotic locomotion skills by imitating animals. *Robotics: Science and Systems (RSS)*, 2020.