2023 IEEE International Conference on Robotics and Automation (ICRA 2023)
May 29 - June 2, 2023. London, UK

Learning Modular Robot Visual-motor Locomotion Policies

Julian Whitman and Howie Choset

***Abstract*— Control policy learning for modular robot locomotion has previously been limited to proprioceptive feedback and flat terrain. This paper develops policies for modular systems with vision traversing more challenging environments. These modular robots can be reconfigured to form many different designs, where each design needs a controller to function. Though one could create a policy for individual designs and environments, such an approach is not scalable given the wide range of potential designs and environments. To address this challenge, we create a visual-motor policy that can generalize to both new designs and environments. The policy itself is modular, in that it is divided into components, each of which corresponds to a type of module (e.g., a leg, wheel, or body). The policy components can be recombined during training to learn to control multiple designs. We develop a deep reinforcement learning algorithm where visual observations are input to a modular policy interacting with multiple environments at once. We apply this algorithm to train robots with combinations of legs and wheels, then demonstrate the policy controlling real robots climbing stairs and curbs.**

I. Introduction

Vision can help a robot learn to locomote through challenging terrain [1]–[4]. Recent work has trained visual-motor control policies, which react in real-time to both exteroceptive (externally-measured/visual) and proprioceptive (internal actuator and IMU) feedback, for legged robots with reinforcement learning (RL) [1]–[4]. However, these prior methods must be re-trained any time a different robot mechanism design is introduced, making it computationally expensive to add new designs. To address this limitation, this paper develops an RL method to train one policy for multiple designs and environments, such that the policy can generalize (i.e., zero-shot transfer, apply without additional optimization/learning) to new designs and environments.

We focus on modular robots, where a set of modules, e.g., legs, wheels, and a body, can be easily re-combined to construct many designs. But a design on its own is not a functional robot; each new design needs a controller. It would be inefficient to create a new control policy from scratch for every new design and environment. Instead, a policy can be trained to control a variety of designs at once. Prior work [5], [6] found that with a modular learning architecture, one policy can control many designs, and even generalize to new designs outside of those seen in training. But, those policies operated only in flat obstacle-free environments using proprioceptive sensor feedback.

This paper develops "Visual-motor MBRL for modular robots," an algorithm that builds on our prior learning methods [5] to traverse more challenging terrain. This work adds exteroceptive inputs, multiple terrains during training, and alters the RL pipeline to support discovery of climbing behaviors. We train one policy with a set of three designs and two environments, and show the policy can generalize to both new designs and environments. Then, we investigate how including more than one design and environment during training impacts policy performance after convergence. Finally, we instrument our robot with onboard vision, and demonstrate that our policy can control real robots to climb stairs and curbs.

Fig. 1: A time-lapse of our hexapod robot climbing stairs using the onboard vision system and our learned modular visual-motor policy. The steps in the stairs shown here are 14 cm high, which at the robot's nominal resting stance, is near the distance between the robot body and the ground. A video of this behavior is in the supplementary material and at `https://youtu.be/2tCj34zY6kI`

II. Background

A. Reinforcement learning

RL methods collect data from an agent's interaction with its environment to train a policy [7]. RL has been used in

Carnegie Mellon University, Pittsburgh PA, USA. jwhitman@alumni.cmu.edu, choset@cs.cmu.edu

recent years with deep neural networks to control a variety of locomoting robots, both in simulation and in reality [7]–[13]. Recent work has shown the benefit of including vision as an input to a policy trained with RL, i.e., those that take exteroceptive inputs in addition to proprioceptive inputs over those that take only proprioceptive inputs [1]–[4].

RL methods can broadly be categorized as model-free RL (MFRL) or model-based RL (MBRL). MFRL treats the robot-environment interactions as a black box, and has been used to train visual-motor policies for quadrupeds [3], [4] that run in real-time. In contrast, MBRL explicitly learns an approximate model of the robot-environment dynamics. MBRL has been shown to be significantly more sample-efficient [14]–[17], that is, requires fewer trajectories to reach convergence. Though most recent MBRL work operates in uniform environments without exteroceptive inputs, [18] added exteroceptive measurements to the model input, and trained on rough terrain. But, [18] was limited to wheeled robots, and did not produce a real-time reactive policy, instead using the approximate model to optimize control actions off-line.

B. Modular policies

The *modular policy* learning framework, introduced in [5], is geared toward systems where a large number of designs are derived from a set of modules. Modular policies can learn to control robots composed of various combinations of modules, then transfer without additional training to new robots with different designs. In these policies, each type of module has a neural network associated with it– i.e. there is one network used to control all of the "leg" modules, and a different one used to control all of the "wheel" modules. The structure of a modular robot is represented as a design graph, with nodes as modules and edges as connections between them. A design graph is used to create a policy graph with the same structure. Each node in the policy graph uses the neural network associated with its corresponding module type's node to convert the inputs from that module's observations into the outputs for that module's actions. For example, the part of the neural network for leg-type modules is used to compute actions for each of the legs on a hexapod, one leg at a time.

Modular policies are implemented with a graph neural network (GNN) architecture [19], [20], which enables modules to learn to modulate their outputs via a communication procedure in which they send information over the graph edges. As a result, the policy can produce different behaviors for the same module depending on its location and neighboring modules within the robot. In this work, we use the policy architecture of [5], with additional observations for local terrain measurements included in the body node input.

III. Modular visual-motor policy learning

A. Problem formulation

The designs to be trained D_{train} are first chosen by the user. The objective we optimize maximizes the reward a policy receives when applied to multiple designs and environments,

$$\theta^* = \underset{\theta}{\operatorname{argmax}}\, \mathbb{E}_{e\sim\mathcal{E}}\Bigg[\overbrace{\frac{1}{|D_{\text{train}}|}\sum_{d\in D_{\text{train}}}\underbrace{\mathbb{E}_{a\sim\pi_\theta}\sum_{t=1}^{T} r(s_t,a_t,d)}_{\substack{\text{Expected reward}\\ \text{for each design}}}}^{\text{Average over designs}}\Bigg] \tag{1}$$

$$\text{s.t.}\quad s_{t+1} = f(s_t, a_t, e, d) \qquad \text{(Dynamics)} \tag{2}$$

$$a \sim \pi_\theta(a|o,d) \qquad \text{(Policy)} \tag{3}$$

$$o \sim O(s) \qquad \text{(Observation function)} \tag{4}$$

$$s_o \sim p(s) \qquad \text{(Initial state distribution)} \tag{5}$$

Here π_θ is a stochastic policy conditioned on observations and parameters θ. The policy is applied to the training designs D_{train} on environments e sampled from a distribution of non-flat environments, $e \sim \mathcal{E}$. The reward $r : S \times A \times \mathcal{D} \rightarrow \mathbb{R}$ is computed over a time horizon T. The observation function O adds noise to, or removes, parts of the state vector. The observation function we use removes the x/y position, yaw, and the linear velocity of the body from the state, as these can be estimated by an odometry system, whereas the remainder of the state (joint positions and velocities, body orientation and angular velocity) can be readily obtained from joint and IMU sensors. During training, a low level of Gaussian noise is added to all these observations, with variance tuned to approximate the noise levels of hardware sensors. Exteroceptive observations (e.g. from a vision system) are incorporated into the observations for the body module, making π a visual-motor policy, rather than a purely proprioceptive policy.

This problem formulation is similar to that used in [5], with the addition of marginalization over multiple environments. We approach this problem with an MBRL algorithm, which learns an approximate model $\tilde{f}_\phi$ with parameters ϕ, then optimizes a policy with the model as a differentiable approximation of the dynamics.

B. Algorithm overview

Our algorithm alternates between phases of model learning, policy optimization, and data collection, within each iteration. This work differs from that of our prior and related MBRL methods because each phase gathers data from, and learns to apply to, multiple designs and environments at once. The psuedocode of the method is in Algorithm 1.

In the first phase, an approximate dynamics model is trained with supervised learning from a dataset of randomly generated trajectories in the simulation environment. Next, the model is used within policy optimization, leveraging the fact that the neural network-based model is differentiable and can be applied in large parallel batches. The policy is then applied to robots in simulation to generate more trajectory data. The dynamics model is retrained with the additional data, the policy re-optimized, and the process repeated. All data is gathered from simulation (NVIDIA IsaacGym [21]) with a simulation time step of $1/60$ s, and each action is applied for 5 time steps.

C. Initial model data acquisition

First, the designs D_{train} are randomly initialized at near their nominal joint positions on a uniform flat environment. Actions are created by sampling 10 normally distributed random joint commands, and fitting splines to create smooth joint commands over 100 time steps. These actions are applied to obtain initial trajectories stored in a dataset $\mathcal{T}_{\text{train}}$.

D. Model learning

Next, the dataset $\mathcal{T}_{\text{train}}$ is used for supervised model learning (as in [5], [22]). The modular model $\tilde{f}_\phi$ has the same structure as the modular policy, that is, it is a GNN with body, leg, and wheel module node types. The body node takes as input the state of the body (world frame orientation and velocity). In contrast to [5], we incorporate an additional local terrain observation processed through convolutional layers, flattened, batch-normalized, and appended to the body node input. The leg and wheel nodes take in joint angles and joint velocities from their respective modules. Each node outputs a change in state similarly to [5], [14], [22], [23].

E. Policy optimization

The policy is optimized to maximize reward with respect to the model. Our policy optimization method differs from that of [5], which used the model to create a dataset of trajectories, then trained a policy via behavioral cloning independently from the model. This work instead optimizes the policy network parameters end-to-end with the model. The loss function derived from (1) is

$$\mathcal{L}_{\text{RL}} = -\mathbb{E}_{e\sim\mathcal{E}}\left[\frac{1}{|D_{\text{train}}|}\sum_{d\in D_{\text{train}}}\mathbb{E}_{a\sim\pi_\theta}\sum_{t=1}^{T} r(\tilde{s}_t, a_t, d)\right]. \quad (6)$$

When optimizing this loss, the approximate state evolves according to the model $\tilde{s}_{t+1} = \tilde{f}_\phi(\tilde{s}_t, a_t, e, d)$, where the model is held fixed during policy optimization. The actions are sampled according to $a_t \sim \pi_\theta(a_t|o_t, d)$, $o_t \sim O(\tilde{s}_t)$. The initial state $\tilde{s}_t = s_0$ is taken from the simulator. Over the time horizon length T, the policy is applied to the approximate state, a reward is computed, and the state is advanced according to the model. The total reward over T steps is accumulated in $\mathcal{L}_{\text{RL}}$, and gradients $d\mathcal{L}_{\text{RL}}/d\theta$ are used in gradient descent with an Adam optimizer [24]. In other words, after a fully differentiable T-step roll-out of forward passes using the model and policy, policy parameter gradients are computed by back-propagating through the sequence of states and actions. This process is repeated with random mini-batches of initial states. We also use a one-step (1/12 s) simulated latency [25] to facilitate transfer to reality.

The reward function r is designed to reward a variety of robot designs for locomoting forward over rough terrain. The main term in the objective rewards the distance in the $+x$ direction. Additional terms with smaller relative weights penalize roll, pitch, yaw, deviation from $y = 0$, control effort, and distance from a nominal stance. The exteroceptive observations are taken by sampling the simulation environment height at the approximated state's position. The policy learns locomotive behaviors that vary depending on the design and on the environment sensed.

F. On-policy model data acquisition

After the policy training phase, the policy is used to gather additional trajectories in simulation. In early iterations, the model may be inaccurate, such that the policy optimization can exploit model bias to produce a policy that will be low-cost under the model but could be high-cost in the simulation environment [26]. Consequently, policy optimization can become unstable if the time horizon is too long initially. To iteratively reduce model bias, we apply the policy to gather trajectories, and fine-tune the model with this new data. At the next iteration of policy optimization, the model will be more accurate near the policy, causing a virtuous cycle in which the model improves near states visited by the policy, and the policy is optimized with respect to the refined model.

The policy is applied to the simulation environment deterministically using the mean output of the policy distribution π_θ. The resulting states and actions are added to $\mathcal{T}_{\text{train}}$. The policy can also be applied to the simulation environment with time-correlated noise. That is, the policy is applied to the model for T steps, $x_{t+1} = \tilde{f}_\phi(x_t, \pi_\theta(o_t, e_t))$, to obtain control actions $u_{1:T}$. These control actions are perturbed with time correlated noise similarly to the actions generated in Sec. III-C, then applied to the simulation. Trajectories across the designs and environments are added to $\mathcal{T}_{\text{train}}$. The model is then refit using supervised learning, and used for the next phase policy optimization.

G. Terrain curriculum

We created an adaptive curriculum on the terrain difficulty, inspired by [3], [4]. At the first iteration, the terrain is fully flat, presenting the easiest learning problem. The model and policy are trained initially on flat ground, and once all designs pass a threshold distance travelled, the height of terrain features in all environments is incrementally increased by 2 cm. Then, each batch in the policy training and on-policy trajectories contains some samples from each terrain difficulty level to prevent catastrophic forgetting.

H. Additional implementation details

We found that for our algorithm to succeed, a few additional training implementation details were helpful.

The recurrent policy trained more reliably than a non-recurrent policy; a potential cause for this could be vanishing gradients over the time sequence in which the policy loss was computed. We train the recurrent policy to operate for more than T time steps (the policy optimization horizon) with truncated back-propogation through time [27]. Each time a batch of states are sampled as initial states for the policy rollout, half of those states come from the simulation of the previous policy iteration, and half of those states come from an "imagination" of applying the policy to the current model. The states that come from imagination are associated with a policy hidden recurrent vector, used as the initial hidden recurrent vector for those initial states. Using a policy

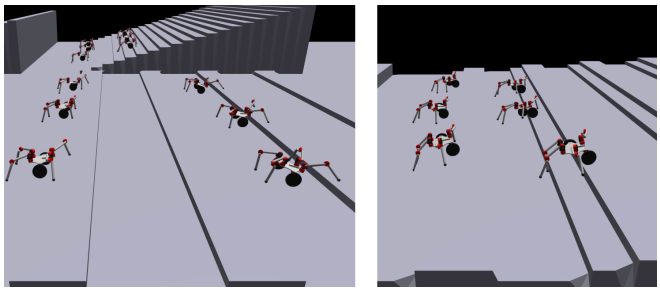

Fig. 2: Our modular visual-motor policy is trained simultaneously using multiple designs and environments at the same time. Left: One of the designs trained to locomote through stairs and curbs. Right: Generalization to a new design and a new environment not seen during training. We show instances of the same robot at two time steps, at states reached from three initial conditions.

that is recurrent over time, and not just recurrent over the internal propagation phase of each GNN forward pass, has an additional benefit. We were able to reduce the number of internal propagation steps within the GNN at each time step to only one step. As a consequence, the forward pass is slightly less computationally expensive than it would be for greater number of internal propagation steps (see [5], [20] for descriptions of GNN internal propagation steps).

Secondly, we found that with a fixed learning rate, the policy optimization would sometimes diverge. We implemented a simple adaptive learning rate to stabilize policy optimization. At the start of policy optimization, we compute the net reward from applying the policy for T steps to a large set (e.g. 10x the batch size) of initial states. This "validation reward" acts similarly to a validation loss in a supervised learning problem. If the current validation reward becomes lower than that initial validation reward, we revert the policy parameters back to the last point when the validation reward was computed, lower the learning rate, and continue.

Thirdly, we found regulating the stochastic policy entropy, by rewarding the variance output by the policy, aids in balancing exploration and exploitation. During policy optimization, a small entropy bonus is added to the RL loss (as in [28]), which prevents premature convergence to a poor local minima and aids in discovering the transition from flat-ground to climbing behavior.

IV. Generalization Experiments

We next measure the policy's capability to generalize to new designs and new environments. We created a training and test set of designs and environments, a baseline behavior as a basis of comparison, and compared the performance under conditions both seen and unseen in training.

A. Designs and environments tested

In these experiments, we limit the training designs D_{train} to three designs, each with four legs and two wheels, where the position of the wheels is left-right symmetric and either in the front, middle, or rear of the body. The test designs

Algorithm 1 Visual-motor MBRL for modular robots. Each step is conducted for multiple designs and environments.

Collect dataset $\mathcal{T}_{\text{train}}$ from random action trajectories
for $i = 1 \dots N$ **do**
  Model learning phase:
  Train model $\tilde{f}_\phi$ from $\mathcal{T}_{\text{train}}$ with supervised learning
  Policy optimization phase:
  Initialize buffer $\mathcal{B}$ with random initial robot states and zero-valued policy hidden-state vectors, i.e., $\mathcal{B} \leftarrow \{s_{0,i}, h_0\}_{i=1}^{N_{\text{batch}}}$ where $h_0 = \vec{0}$
  for $k = 1 \dots K$ **do**
    Sample a batch of initial states s_0 with hidden-state vectors from $\mathcal{B}$, and set the hidden state of the policy π_θ
    $R \leftarrow 0$, $\tilde{s}_0 = s_0$
    Roll out policy with approx. dynamics
    for $t = 1 \dots T$ **do**
      $a_t \sim \pi_\theta(a_t|O(\tilde{s}_t))$, $\tilde{s}_{t+1} = \tilde{f}_\phi(\tilde{s}_t, a_t)$
      $R \leftarrow R + r(\tilde{s}_t, a_t)$
    end for
    $\mathcal{L}_{\text{RL}} = -R$
    Gradient descent on policy parameters $d\mathcal{L}_{\text{RL}}/d\theta$
    Overwrite part of $\mathcal{B}$ with intermediate states and policy hidden-state vectors, i.e., $\mathcal{B}^{N_{\text{batch}}/2:N_{\text{batch}}} \leftarrow \{s_{T/2,i}, h_{T/2}\}_{i=N_{\text{batch}}/2}^{N_{\text{batch}}}$
  end for
  Terrain curriculum: Increase environment difficulty for designs that have met the performance threshold
  On-policy model data acquisition:
  Apply π_θ to collect data $\mathcal{T}_{\text{new}}$
  $\mathcal{T}_{\text{train}} \leftarrow \mathcal{T}_{\text{train}} \cup \mathcal{T}_{\text{new}}$
end for

D_{test} consists of three designs with four legs and two wheels, where the location of the wheels is left-right asymmetric.

We created training environments with two types of terrain. The first, "stairs," contains ascending steps. The second, "curbs" contains rectangular blocks with fixed width at a regular interval. The test environment consists of steps and flat regions staggered. Fig. 2 depicts a visualization of these environments. The terrain difficulty curriculum includes "levels" of difficulty, ranging from fully flat to higher obstacles, with the maximum step height increasing in 2 cm intervals. For the following experiments, we report the results of each policy tested on a nearly-flat terrain (2 cm steps) and a more difficult terrain (10 cm steps).

After training, for each design/environment combination, the policy was simulated from 10 perturbed initial conditions. The mean and standard deviation of the distance travelled after 200 time steps (16.7 seconds) was recorded. Each policy was trained three separate times, and the results averaged for each cell. Although the reward includes multiple terms, we use the distance travelled forward as a metric for

policy success, as the reward is dominated by this term and it is more easily interpretable than the full reward value. Training was run for 20 iterations, such that the three-design two-environment process trained for about 8.5 hours on a computer with an 8-core AMD Ryzen 7 2700X processor and a NVIDIA GTX 1070 GPU.

B. Comparison to hand-crafted baseline

We developed a hand-crafted gait applicable to the various combinations of legs and wheels tested in this work. All legs are given an alternating tripod gait, with phase and position offsets assigned as if they were in a hexapod. All wheels are given differential drive/skid-steering commands, with position offsets and wheel speeds assigned as if they were on a car. Gait speeds and amplitudes were tuned to produce steps as fast and high as could be tracked by the joint velocity limits. This baseline gait is not fully open-loop, as it reacts to steer to face forwards using the observed yaw angle. The baseline gait enables the tested designs to locomote effectively on flat ground and over small obstacles, but its performance degrades as the terrain features (steps, curbs) become larger. In the tables following, we denote the results from this gait as "tripod baseline."

C. Generalization to new designs and environments

First we test the capability of the policy when applied to new (i.e., "test," unseen in training) designs, new environments, and both simultaneously. Here the policy is trained with three designs and two environments jointly, then the average distance travelled is recorded in Table I. We found that the policy produced larger displacement than the baseline. These results indicate that the policy can generalize to new designs in new environments at the same time, and still produce behaviors better than our hand-crafted policy, though there is a small drop in performance between the train and test results for the 10 cm steps.

	Avg. dist. traveled in 200 time steps (m)	
2 cm steps	Tripod baseline	Policy
Train designs + train envs.	6.3	**7.0** $\pm$ 0.5
Test designs + train envs.	5.9	**7.0** $\pm$ 0.4
Train designs + test env.	6.6	**7.1** $\pm$ 0.3
Test designs + test env.	6.2	**7.3** $\pm$ 0.3
10 cm steps	Tripod baseline	Policy
Train designs + train envs.	2.3	**4.9** $\pm$ 0.2
Test designs + train envs.	3.5	**4.2** $\pm$ 0.4
Train designs + test env.	2.5	**4.2** $\pm$ 1.0
Test designs + test env.	3.6	**4.2** $\pm$ 0.6

TABLE I: Generalization to new designs and/or environments. "Train" indicates that the design/environment was seen during training, "test" indicates that it was not. The policy mean and standard deviation are listed after the policy is trained from three times with random initial seeds. The policy was able to pass the baseline in all cases, even when both the designs and environments were not seen during training.

D. The trade-off between specialization and generalization

Training with multiple designs/environments enables generalization to new designs/environments, but requires optimizing a more complex loss. We next test whether including multiple designs and/or environments in training effects the policy's performance.

The policy is trained with one/multiple designs and one/-multiple environments at a time. When one design/environment at a time is used, then we train policies separately for each, and average the performance of those policies on the conditions in which they were trained. For example, in the "Design ind. + Envs. joint" condition, we train three separate policies, one for each of the three designs. Each policy is trained with multiple environments, and other hyperparameters are held constant. We then measure the performance of those three policies on their corresponding designs, where each policy is applied to the environments, and the performance over all designs and environments is averaged. When multiple designs/environments are included in training, we also test whether that policy can generalize to test designs/environments. Results, averaged over three training runs from random initial seeds, are recorded in Table II.

A policy trained with multiple designs and environments comes with drawbacks alongside its benefits. On flat ground and low obstacles, the policy performance was not significantly affected by additional designs or environments in training. But as the task becomes more difficult with larger obstacles (higher steps), the policy performance was lower when trained with multiple designs and environments than when trained individually for each of those same designs or environments. We call this the "joint learning delta:" when training with more than one design or environment, the performance on the more difficult terrain decreased.

The joint learning delta is most apparent from the difference between the "Designs ind. + Envs. ind." condition and the "Designs joint + Envs. joint" condition in Table II The policy is fit to a more complex objective, rather than specializing to a single design and environment. As a consequence, though it gains the ability to generalize, we observed a decreased performance compared to a specialized policy. Jointly training with multiple designs and environments will likely require larger neural network capacities, larger batch sizes, and more training time in order to reach the performance of the policy that is trained with a single design and environment.

V. Robot demonstrations

Given that "simulations are doomed to succeed" [29], we validated our policies in reality. We constructed a modular robot with onboard power (GRIN batteries [30]), computation (4-core i7 Intel NUC), sensing (a Realsense D435 and XSens IMU), and Hebi X-series actuators [31]. The onboard vision system uses VINS-Fusion [32] for odometry, and Elevation Mapping [33], [34] to produce maps. A local terrain map of 21×21 grid of points (1.5 by 1.5 m) aligned with the robot body frame is sent as input to the policy at

	Avg. dist. traveled in 200 time steps (m)				
	Tripod baseline	Designs ind. + Envs. ind.	Designs joint + Envs. ind.	Designs ind. + Envs. joint	Design joint + Envs. joint
Train designs + train envs. (2 cm)	6.3	7.2	7.0	7.6	7.0
Test designs + train envs. (2 cm)	5.9	–	7.0	–	7.0
Train designs + test env. (2 cm)	6.6	–	–	7.1	7.1
Train designs + train envs. (10 cm)	2.3	6.4	5.1	6.2	4.9
Test designs + train envs. (10 cm)	3.5	–	4.6	–	4.2
Train designs + test env. (10 cm)	2.5	–	–	4.2	4.2

TABLE II: The effect of including multiple designs and/or environments during training ("Train" conditions) on policy performance and generalization ("Test" conditions). When designs/environments are trained individually, we do not test generalization to unseen designs/environments, and mark these conditions with a "–".

each time step. Note this robot does not have foot contact sensing, so it cannot directly sense ground contacts.

The policy outputs target joint velocity commands tracked by low-level PID loops at a higher frequency on-board the actuator. The actuators perform more accurate tracking when provided with a feed-forward (FF) torque value, in particular when they are under heavy load. Therefore in addition to control outputs, we train a torque estimation network (implemented as a GNN with the same structure as the policy) that estimates the feed-forward torque needed for the actuator to track the desired velocity. The data for this training is obtained by tracking the torques experienced in simulation.

Our policy was able to transfer from simulation to reality. Most notably, the policy was able to control a hexapod to climb outdoors human-scale stairs. These demonstrations are shown in Figures 1 and 3, in the supplementary video and at `https://youtu.be/2tCj34zY6kI`.

To test the reliability of the simulation-to-reality transfer, we conducted trials of the policy applied to a hexapod traversing a 19 cm-high roadside curb. The robot was reset one meter from the curb, then the modular visual-motor policy applied for 20 s. We conducted five trials, and in each one, the robot was able to locomote over the curb, transitioning from a flat-ground walking motion to a climbing motion and back to flat ground. We also applied the baseline alternating tripod behavior, with the step size high enough to step on to the curb. Though effective on flat ground, the alternating tripod baseline only traversed the curb one out of five trials, indicating that this obstacle is difficult enough to require a different behavior to succeed. Finally, we conducted a test in which the visual-motor policy at run-time was given a spoofed terrain map image consistent with walking on flat ground, effectively "blindfolding" the policy. The ablated policy locomoted on flat ground, but did not traverse the curb in any of five trials, indicating it uses the exteroceptive feedback to adapt its behavior.

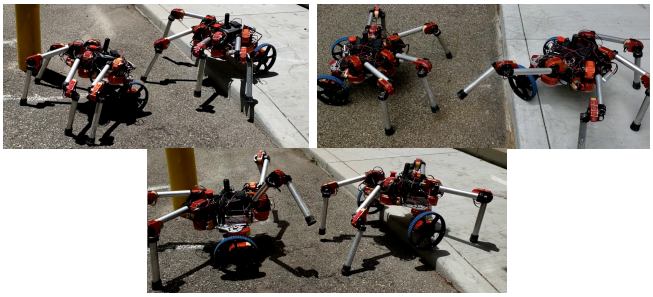

Fig. 3: Three time-lapses of robots with four legs and two wheels climbing up a curb using the onboard vision system and modular visual-motor policy.

VI. Conclusions

This work presents a deep RL algorithm that trains a modular visual-motor policy to control multiple designs in multiple environments. We showed that the policy can generalize to new designs and environments, and transfer from simulation to reality. To the best of our knowledge, this is the first demonstration of a legged robot locomoting through non-flat terrain using MBRL.

These methods also come with some limitations. We validated that our policy can transfer from simulation to reality, but on more difficult environments, the transferred behaviors degrade compared to their simulated counterparts, though we have yet to quantify the sim-to-real gap for the full variety of designs and environments used in this work. Future work will aim to adapt techniques such as domain randomization [35] or domain adaptation [36]. More accurate robot models in simulation, or learning from a mix of simulation and real data, may also improve sim-to-real. Further, we showed only a limited number of designs and environments, and ongoing work aims to increase the variety of designs and environments included.

Another direction for future work is to learn partially from expert knowledge. This paper could be combined with our concurrent work on combining imitation and reinforcement learning for modular robots [37], such that the climbing policy could either be warm-started by imitating an existing policy or hand-engineered gait, even when the demonstrations are shown for designs not contained within D_{train}.

We developed an MBRL method to train modular policies, but similar concepts could be applied to MFRL methods. Due to our focus on generalizability in this work, we have not yet directly compared how well our policy performs relative to MFRL methods, those applied to individual designs in related work, or with different policy architectures. A direction for future work is adapting methods such as [4] or [6], to modular robots with vision in varied environments.

REFERENCES

[1] A. Agarwal, A. Kumar, J. Malik, and D. Pathak, "Legged locomotion in challenging terrains using egocentric vision," in *6th Annual Conf. on Robot Learning*, 2022. [Online]. Available: https://openreview.net/forum?id=Re3NjSwf0WF

[2] W. Yu, D. Jain, A. Escontrela, A. Iscen, P. Xu, E. Coumans, S. Ha, J. Tan, and T. Zhang, "Visual-locomotion: Learning to walk on complex terrains with vision," in *5th Annual Conf. on Robot Learning*, 2021.

[3] T. Miki, J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning robust perceptive locomotion for quadrupedal robots in the wild," *Science Robotics*, vol. 7, no. 62, p. eabk2822, 2022.

[4] N. Rudin, D. Hoeller, P. Reist, and M. Hutter, "Learning to walk in minutes using massively parallel deep reinforcement learning," in *Conf. on Robot Learning*. PMLR, 2022, pp. 91–100.

[5] J. Whitman, M. Travers, and H. Choset, "Learning modular robot control policies," *arXiv preprint arXiv:2105.10049*, 2021.

[6] W. Huang, I. Mordatch, and D. Pathak, "One policy to control them all: Shared modular policies for agent-agnostic control," in *ICML*, 2020.

[7] J. Ibarz, J. Tan, C. Finn, M. Kalakrishnan, P. Pastor, and S. Levine, "How to train your robot with deep reinforcement learning: lessons we have learned," *The Int. Journal of Robotics Research*, vol. 40, no. 4-5, pp. 698–721, 2021.

[8] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter, "Learning agile and dynamic motor skills for legged robots," *Science Robotics*, vol. 4, no. 26, 2019.

[9] Z. Xie, P. Clary, J. Dao, P. Morais, J. Hurst, and M. Panne, "Learning locomotion skills for cassie: Iterative design and sim-to-real," in *Conf. on Robot Learning*, 2020, pp. 317–329.

[10] N. Heess, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, S. Eslami, M. Riedmiller *et al.*, "Emergence of locomotion behaviours in rich environments," *arXiv preprint arXiv:1707.02286*, 2017.

[11] R. Hafner, T. Hertweck, P. Kloeppner, M. Bloesch, M. Neunert, M. Wulfmeier, S. Tunyasuvunakool, N. Heess, and M. Riedmiller, "Towards general and autonomous learning of core skills: A case study in locomotion," in *Conf. on Robot Learning*. PMLR, 2021, pp. 1084–1099.

[12] B. Zhang, R. Rajan, L. Pineda, N. Lambert, A. Biedenkapp, K. Chua, F. Hutter, and R. Calandra, "On the importance of hyperparameter optimization for model-based reinforcement learning," in *Int. Conf. on Artificial Intelligence and Statistics*. PMLR, 2021, pp. 4015–4023.

[13] B. Amos, S. Stanton, D. Yarats, and A. G. Wilson, "On the model-based stochastic value gradient for continuous reinforcement learning," in *Learning for Dynamics and Control*. PMLR, 2021, pp. 6–20.

[14] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine, "Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning," in *Int. Conf. on robotics and Automation*. IEEE, 2018, pp. 7559–7566.

[15] K. Chua, R. Calandra, R. McAllister, and S. Levine, "Deep reinforcement learning in a handful of trials using probabilistic dynamics models," in *Advances in Neural Information Processing Systems*, 2018, pp. 4754–4765.

[16] B. Amos, I. D. J. Rodriguez, J. Sacks, B. Boots, and J. Z. Kolter, "Differentiable MPC for end-to-end planning and control," in *Int. Conf. on Neural Information Processing Systems*, 2018, pp. 8299–8310.

[17] A. Rajeswaran, I. Mordatch, and V. Kumar, "A game theoretic framework for model based reinforcement learning," in *Int. Conf. on Machine Learning*. PMLR, 2020, pp. 7953–7963.

[18] S. J. Wang, S. Triest, W. Wang, S. Scherer, and A. Johnson, "Rough terrain navigation using divergence constrained model-based reinforcement learning," in *Conf. on Robot Learning*. PMLR, 2022, pp. 224–233.

[19] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *Trans. on Neural Networks*, vol. 20, no. 1, pp. 61–80, 2008.

[20] T. Wang, R. Liao, J. Ba, and S. Fidler, "Nervenet: Learning structured policy with graph neural networks," in *Int. Conf. on Learning Representations*, 2018.

[21] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa *et al.*, "Isaac gym: High performance gpu based physics simulation for robot learning," in *Conf. on Neural Information Processing Systems, Datasets and Benchmarks Track*, 2021.

[22] Y. Yang, K. Caluwaerts, A. Iscen, T. Zhang, J. Tan, and V. Sindhwani, "Data efficient reinforcement learning for legged robots," in *Conf. on Robot Learning*. PMLR, 2020, pp. 1–10.

[23] A. Sanchez-Gonzalez, N. Heess, J. T. Springenberg, J. Merel, M. Riedmiller, R. Hadsell, and P. Battaglia, "Graph networks as learnable physics engines for inference and control," *arXiv preprint arXiv:1806.01242*, 2018.

[24] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *Int. Conf. on Machine Learning*, 2015.

[25] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke, "Sim-to-real: Learning agile locomotion for quadruped robots," in *Robotics: Science and Systems*, Pittsburgh, Pennsylvania, June 2018.

[26] M. Deisenroth and C. E. Rasmussen, "Pilco: A model-based and data-efficient approach to policy search," in *Int. Conf. on Machine Learning*. Citeseer, 2011, pp. 465–472.

[27] P. J. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[28] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *Int. Conf. on Machine Learning*. PMLR, 2018, pp. 1861–1870.

[29] R. A. Brooks and M. J. Mataric, "Real robots, real learning problems," in *Robot learning*. Springer, 1993, pp. 193–213.

[30] Grin Technologies, "[Online] https://ebikes.ca," 2022.

[31] Hebi Robotics, "[Online] www.hebirobotics.com," 2022.

[32] T. Qin, P. Li, and S. Shen, "Vins-mono: A robust and versatile monocular visual-inertial state estimator," *Trans. on Robotics*, vol. 34, no. 4, pp. 1004–1020, 2018.

[33] P. Fankhauser, M. Bloesch, and M. Hutter, "Probabilistic terrain mapping for mobile robots with uncertain localization," *Robotics and Automation Letters*, vol. 3, no. 4, pp. 3019–3026, 2018.

[34] P. Fankhauser, M. Bloesch, C. Gehring, M. Hutter, and R. Siegwart, "Robot-centric elevation mapping with uncertainty estimates," in *Int. Conf. on Climbing and Walking Robots (CLAWAR)*, 2014.

[35] S. J. Wang and A. M. Johnson, "Domain adaptation using system invariant dynamics models," in *Learning for Dynamics and Control*. PMLR, 2021, pp. 1130–1141.

[36] A. Kumar, Z. Fu, D. Pathak, and J. Malik, "RMA: Rapid motor adaptation for legged robots," *Robotics: Science and Systems*, 2021.

[37] J. Whitman and H. Choset, "Learning modular robot locomotion from demonstrations," *arXiv preprint arXiv:2210.17491*, 2022.