Learning to Navigate by Pushing

Cornelia Bauer*, Dominik Bauer*, Alisa Allaire, Christopher G. Atkeson and Nancy Pollard

Abstract— **In this work, we investigate a form of dynamic contact-rich locomotion in which a robot pushes off from obstacles in order to move through its environment. We present a reflex-based approach that switches between optimized hand-crafted reflex controllers and produces smooth and predictable motions. In contrast to previous work, our approach does not rely on periodic movements, complex models of robot and contact dynamics, or extensive hand tuning. We demonstrate the effectiveness of our approach and evaluate its performance compared to a standard model-free RL algorithm. We identify continuous clusters of similar behaviours, which allows us to successfully transfer different push-off motions directly from simulation to a physical robot without further retraining.**

I. INTRODUCTION

Humans exploit highly dynamic interactions with the environment to reach states that would have otherwise been inaccessible. For instance, to move to an out-of-reach handhold, a rock climber jumps to the new handhold by pushing against the wall with their legs. Swinging from the current handhold before the jump can help them gain momentum for larger jumping distances. In parkour, a popular technique for scaling high walls is to run towards the wall and then jump onto and push off the wall with a foot to reach the top. In contrast to these highly dynamic interactions, robotic systems typically attempt to avoid obstacles and only navigate and operate in clear, structured settings such as labs and empty hallways. If robots can use their arms to dynamically push off obstacles, they could move more freely, flexibly, and rapidly in cluttered environments. However, this particular skill is difficult to learn because of the intermittent contact events we encounter during a dynamic pushing motion.

In this work, we explore this scenario with the PushBot, a freely hovering robot, that can only move by interacting with its environment. To solve pushing tasks we develop a reflex-based approach that switches between optimized reflex controllers triggered by specific observations. We show that our approach can solve push-off tasks in simulation, and achieves success rates comparable to a standard model-free RL algorithm. Furthermore, our approach allows us to identify continuous clusters of similar behaviors, and we demonstrate how these motion families can be utilized to directly transfer push-off motions from simulation to a real robot without further reward engineering or retraining.

II. RELATED WORK

Control of robotic whole-body locomotion. Designing robots that can achieve dynamic, contact-rich, whole-body locomotion similar to human and animal capabilities remains challenging. Dynamic legged locomotion has often been considered within the context of walking, running, hopping, and jumping which involve periodic or semi-periodic movements [1], [2], [3], [4], [5]. Our interest lies in non-periodic behaviors that allow robots to exploit dynamic interactions for locomotion in unstructured environments. For most mobile robots with arms, the arms are typically only engaged for static or quasi-static tasks such as balance and support, object manipulation, or climbing [6], [7], [8], [9], [10]. Zhao et al. [11] demonstrate non-periodic humanoid locomotion on challenging and unpredictable simulated terrain, where both the arms and legs are used to move dynamically through the environment.

Previous approaches for legged locomotion have used simplified template models, such as those based on inverted pendulums [12], for model-based planning and control; such models can also be defined for highly dynamic tasks. For instance, a simplified model was developed for the ParkourBot and demonstrated on a chute climbing, or "vertical running", task [13]. However, reduced template models are more difficult to design for unstructured environments and robots capable of more complex movements. Additionally, they usually require more hand tuning than learned models to adapt to new robots and environments.

Learning and optimization techniques, such as quadratic programming [3], [8], [9], [10], black-box optimization (BBO) [14], [15], [16], [17], [18], and reinforcement learning (RL) [19], [20] have been applied for automatic optimization and tuning of locomotion controllers. There has been increased interest in learning controllers for humanoid locomotion using model-free RL which does not require expert domain knowledge or strict assumptions about the policy [21], [22]. BBO methods can be similarly used in a model-free way using an approach called direct policy search. Salimans et al. [23] demonstrated that a black-box evolutionary algorithm is competitive with popular policy gradient RL algorithms on a simulated 3D walking task.

Reflex control modules. Our approach is motivated by prior works related to biologically inspired, whole-body humanoid locomotion. Human locomotion and the neural circuitry involved in generating locomotion behaviors have been studied extensively in previous works [24] [25]. Experiments have shown that neural networks along the spinal cord make a significant contribution to generate these behaviors, and various approaches have been proposed to model the neural circuitry. In this context, Song et al. [15] demonstrate that a control structure composed of reflex modules can generate diverse

*equal contribution

Robotics Institute, Carnegie Mellon Univ, 5000 Forbes Ave, Pittsburgh, PA 15213, USA, {cornelib,dominikb,aallaire,cga,nsp}@andrew.cmu.edu

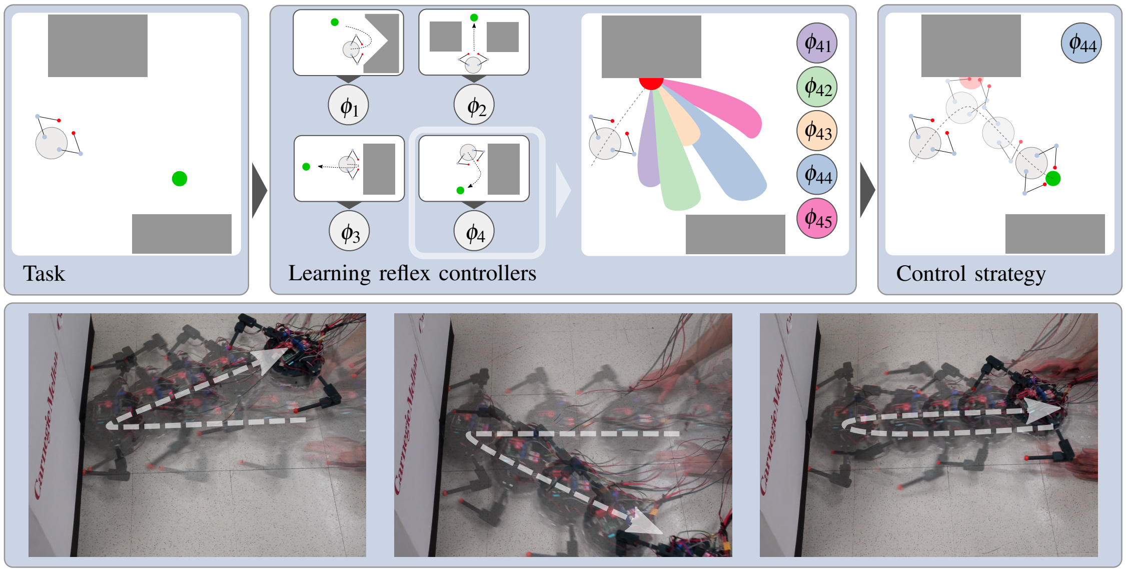

Fig. 1. We propose a reflex-based controller approach to learn push-off tasks, which switches between optimized low-level controllers triggered by specific observations. Each reflex controller is trained on an individual task, e.g. a front push reflex, and within these low-level controllers we identify different motion families which represent clusters of similar behaviors (colored patches). This approach allows us to transfer different push-off motions directly from simulation to a physical robot without further retraining. The bottom row shows time-lapses of pushing motions to the right, left and center, respectively. In each image we accelerate the robot towards the obstacle, contact triggers the pushing motion, and the robot moves to its final position (opaque).

and robust locomotion. These reflex modules are ”simple” decentralized control units, which map sensory feedback onto activation of one or multiple muscles. Combined with a higher level control layer, their model is able to generate periodic behaviors ranging from walking and running to stair climbing and obstacle avoidance. Zhao et al. [11] achieve dynamic, contact-rich humanoid locomotion by combining whole-body locomotion behaviors using a high level reactive planner. Similarly, we focus on the development of reactive controllers that define simple dynamic pushing behaviors.

III. Overview and Contributions

Pushing and pulling are two of the most basic motions that humans use to navigate through clutter. While it is intuitive for us to align our arms in anticipation of contact, react to contact, and adjust our arm stiffness, it is difficult for a robot to learn these behaviors. We investigate a form of dynamic locomotion where an omnidirectional robot base with a pair of two degree of freedom (DoF) arms pushes against nearby obstacles to move through its environment. Unlike previous works applied to running, hopping, or climbing, neither the applied force nor desired net movement direction is aligned with gravity. Without gravity to help generate momentum, the robot must utilize contact-rich dynamic interaction with the environment to reach the desired goal.

Model-free reinforcement learning has shown promising results for learning contact-rich tasks in simulation, but comes with the drawbacks of poor sample efficiency and challenges of sim2real transfer. In this work, we present an approach that only relies on a kinematic robot model and the general ability to adjust stiffness, damping and timing of an end effector trajectory. Our **reflex-based controller** features a hierarchical structure and optimized sub-controllers (fig. 1, top). A high-level controller switches between relatively simple sub-controllers, triggered by specific observations. We build the structure of these sub-controllers, the *reflex controllers*, from reasoning about human pushing motions, and find suitable values for the controller parameters through optimization. We identify clusters of similar motions within the resulting behaviors, resembling the underlying patterns observed in human locomotion. We apply our approach to multiple tasks in simulation and compare the resulting motions to a baseline RL algorithm (Proximal Policy Optimization, PPO [26]). We further show that we can successfully perform real world pushing tasks by directly transferring control policies learned in simulation to a real robot.

PushBot – A hovercraft with arms: To demonstrate that our approach generalizes to the real world we build a physical robot platform (fig. 1, bottom). To introduce compliance in the arms, we choose a direct drive configuration using brushless DC motors in each joint. Additionally, we reduce friction between the floor and the robot by building a hovercraft-like lift system. This creates a low friction air cushion that allows the robot to move freely in all directions. The robot uses piezo-electric vibration sensors [27] in each end effector to detect contacts.

IV. Model and Methods

A. *Robot states and action spaces*

We define the robot **state** at a time t as

$$s_t = \left(v_t \quad \theta_{r,t} \quad c_{r,t} \quad \theta_{l,t} \quad c_{l,t} \quad \xi_t \quad \zeta_t\right)^T$$

where v_t describes x and y component of the velocity of the trunk. $\theta_{r,t}$ and $\theta_{l,t}$ are the joint angles of the right and left arm, and $c_{r,t}, c_{l,t}$ are binary contact variables for each end effector. ξ_t describes the current pose of the robot relative to the goal in terms of distance and heading. We define the robot heading error as the difference between the goal heading and the current robot heading. Finally, an obstacle to interact with is described by ζ_t, which encodes the distance from the spatially closest obstacle, and the relative heading between that obstacle and the current robot heading. For the scope of this work we consider the state to be fully observable and there is only a single obstacle.

We define **actions** as

$$u = (\theta_r, \theta_l, k_p, k_d)^T$$

with goal angles θ_r, θ_l, and variables k_p and k_d specifying the gains of the arm joint PD-controllers. A behavior is considered **successful** if the absolute distance between the robot trunk and goal $d_{g,r} = |p_r - p_g| < 0.1m$ at the end of the push (time $t = N$). A **reward** is given at each timestep t to encourage movement towards the goal $r_{tot} = \gamma_v r_d - 1$ with distance-based reward $r_d = 1 - \hat{d}_{g,r}^{0.4}$ and a velocity-based discount factor $\gamma_v = (1 - \max\{|\hat{v}_t|, 0.1\})^{\max\{\hat{d}_{g,r}, 0.4\}^{-1}}$ with normalized $\hat{d}_{g,r}, |\hat{v}_t| \in [0,1]$ to discourage overshooting.

B. Modeling pushing motions

We intuitively know that we can generate the momentum required for pushing off by extending our arms and exerting a force onto an obstacle. By varying the distance along which we push and how fast we execute the motion, we can scale the intensity of a push. A quick and full extension of our arms will lead to stronger pushes. Instead of learning an optimal trajectory from scratch, we incorporate our innate knowledge about this task into a reflexive controller that constrains the kinematic motion of the arms to retraction and extension along a fixed direction. For a push, we allow the end effector to move along a straight line of action as shown in fig. 2 *left*.

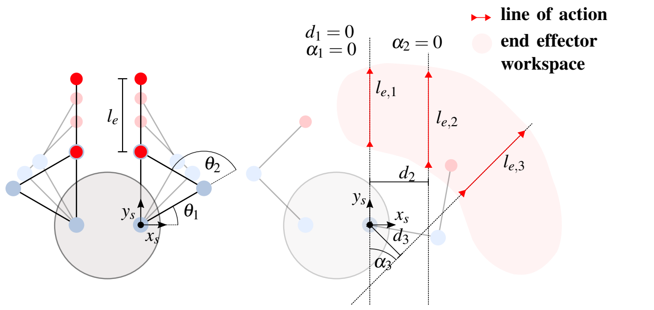

Fig. 2. Left: Pushing model, described by the effective length of the push. Right: Qualitative examples of right-arm pushes along different lines of actions, parameterized by distance d_i from the shoulder joint origin and angle α_i between line of action and shoulder joint y-axis.

Each possible linear motion is parameterized by the angle α between the *line of action* and the y-axis of the shoulder joint reference frame, and an offset d between the *line of action* and the shoulder joint reference frame fig. 2 *right*. The resulting kinematic configuration of the robot arm for a push can be described by $\kappa = (\alpha, d)$. Each position along a *line of action* L_i is parameterized by the coordinate $l_i \in [0,1]$, with $l_i = 0$ specifying the fully retracted and $l_i = 1$ the fully extended arm position. Since this representation parameterizes end effector motions along a linear trajectory in cartesian space it also applies to robot arms with more joints and DoF given that an inverse kinematics model exists.

C. Reflex-based controller for pushing tasks

We propose a hierarchical controller structure, which is composed by a high-level controller that, based on the current observation, switches between different *reflex controllers*. Each reflex controller is dedicated for controlling specific parts of the overall motion, such as placing the arms in front in anticipation of the impact, or extending the arms to push off while in contact with the obstacle.

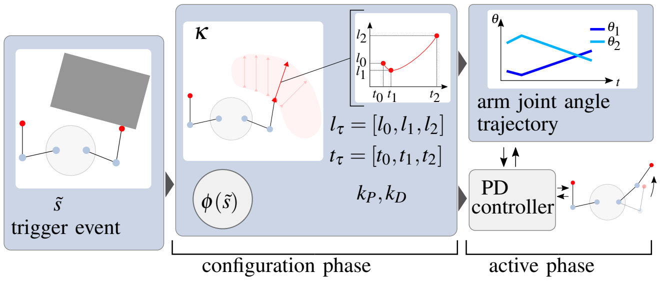

Fig. 3. Reflex controller at runtime: Contact at end effector triggers reflex controller (*trigger event*), configuration policy selects arm configuration setpoints l_τ, timesteps t_τ, kinematic pushing configuration κ, and PD gains k_p, k_d (*configuration phase*). Arm joint angle trajectory is executed (*active phase*).

1) High-level controller: During a pushing task, the robot experiences changes in contact modes. Based on the current robot state, the high-level controller switches between different control modes. We implement this as a finite state machine with explicit transition rules, where the switching mechanism is dictated by a set of handcrafted rules (e.g. if a contact occurs at either end effector, switch on front-push reflex controller).

2) Low-level reflex controller: Whenever a reflex controller is triggered by the high-level controller, it runs through two separate phases as described in fig. 3. Before the reflex controller starts to execute an arm motion, the parameters required for a successful push need to be selected based on the current observation $\tilde{s}$ (*configuration phase*). The controller then stays active for a fixed period of timesteps and executes a specific dynamic motion (*active phase*).

Configuration phase: Each reflex controller constitutes a configuration policy

$$\phi : \tilde{s} \rightarrow \{l_\tau, t_\tau, \kappa, k_p, k_d\} \qquad (1)$$

that maps the state $\tilde{s}$ observed at configuration time to a sequence of arm configuration setpoints l_τ, timesteps t_τ, kinematic pushing configuration $\kappa = (\alpha, d)$, and low-level controller gains k_p, k_d. Depending on the type of the reflex controller, it can either produce a single target position l_0 or a sequential trajectory along the line of action L for the end effector to reach. This corresponds to either a single desired arm configuration (e.g. $l_\tau = 0 \rightarrow$ retracted, $l_\tau = 1 \rightarrow$ extended), or a desired motion (e.g. $l_\tau = [0,1] \rightarrow$ first fully retract, then fully extend the arms). A trajectory along a line of action L is encoded by a sequence of setpoints $l_\tau = [l_0, l_1, \ldots, l_n]$ and desired number of timesteps to reach each l, given by $t_\tau = [t_0, t_1, \ldots, t_n]$. Given an arbitrary setpoint l we can calculate the corresponding joint angles directly from the kinematics model described in section IV-B.

In order to execute the desired kinematic motion plan, the reflex controller uses low-level PD joint controllers to move the end effector. In addition to l_τ, t_τ, κ, the configuration policy also selects k_p, k_d, effectively adjusting the stiffness of each arm during the active phase. We find a suitable configuration policy for a task through optimization, as described in section IV-D.

Active phase: After a reflex controller has been triggered, it stays active for $N_t = \sum t_\tau$ timesteps, and executes the configured trajectory. The kinematics model maps the arm configuration setpoints to joint angles θ_i based on the desired configuration, and the final trajectory in joint angle space is obtained though linear interpolation. Finally, these target angles, along with the desired gains k_p, k_d, are passed to the low-level PD-controllers, which generate the required joint torques. Configuration phase and active phase at runtime are shown in fig. 3.

D. Learning reflex controllers

To learn the trajectory and controller gains for a reflex controller, we formulate the following learning problem. Our goal is to find a configuration policy $\phi(\tilde{s})$, such that given the current state $\tilde{s}$ at configuration time (when the reflex controller is triggered) the resulting controller creates a dynamic motion that accelerates the robot trunk towards the goal.

We can define the optimization problem

$$\phi(\tilde{s}) = \arg\max_{\phi} \sum_{t=0}^{N} r_{tot}(s_t)$$

where we seek to maximize the sum of total rewards over N timesteps, with total reward r_{tot} defined in section IV-A.

Initial grid search: We organize the space of possible configuration states $\tilde{s}$ into a discrete space Ω, such that each point $\omega \in \Omega$ represents a distinct value of $\tilde{s}$. In the push task setting, different values of $\tilde{s}$ correspond to different initial conditions, such as goal position and initial position and velocity of the robot. For instance, goal positions $g_n \in [g_{min}, g_{max}]$ and initial robot velocities $v_{0,m} \in [v_{0,min}, v_{0,max}]$, result in a grid space $\Omega \in \mathbb{R}^{n \times m}$.

For each grid point ω, we seek to find the optimal configuration $\phi^*(\omega)$, i.e. the optimal set of PD-controller gains k_p, k_d, motion setpoints l_τ, timesteps t_τ, and kinematic pushing configuration κ, that maximize the task reward. Finding $\phi^*(\omega)\ \forall\ \omega \in \Omega$ constitutes a non-convex problem. However, given the nature of the task locally convex regions exist, which represent families of motions.

In order to find such families we start an initial global search to find the optimal set of parameters $\phi^*(\omega_0)$ for the first point in the discrete space. Many classes of BBO algorithms work well to solve this problem. In our implementation, we use Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [18] due to its ability to handle ill-conditioned and rugged landscapes. $\phi^*(\omega_0)$ is then used as initial guess in a local search to find $\phi^*(\omega_1)$. We use the SciPy [28] implementation of L-BFGS-B [29]. We continue this process iteratively for neighboring points in Ω until the local search does not converge to an optimal solution, meaning the robot does not successfully complete the task. This implies that $\phi^*(\omega_{i+1})$ and effectively the dynamic motion required to complete the task is substantially different from previous $\phi^*(\omega_i)$. We then re-run a global search to find a different locally convex region for ω_{i+1}, and continue with the local searches until we exhaustively found solutions for all $\omega \in \Omega$. The search process is shown for a 2D grid example space in fig. 4 *A-C*.

Iterative process to refine families of motions: To further improve the controller performance, we refine the locally convex regions, which we term motion families. For each grid point, we take solution candidates from neighboring grid points as initial guesses for local searches. If a solution candidate obtains a higher reward, we update the current solution for the grid point. We repeat this process until convergence. As shown in fig. 4 *C-D*, this leads to changes in shape and area of the solution families, or shrinking and extension of individual families.

The set of optimal reflex controller configurations $\phi^*(\Omega)$ found by our search and refinement process is then used to train a k-Nearest-Neighbor regression model. This model serves as configuration policy $\phi(\tilde{s})$ for the reflex controller; given a state $\tilde{s}$ it predicts a suitable pushing configuration.

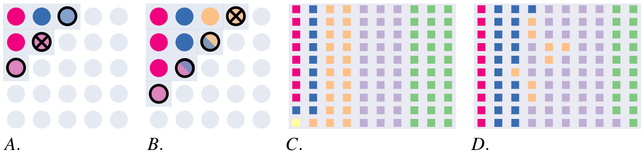

Fig. 4. Grid search: Starting with an initial global solution (*A* top left), we apply local search to neighboring grid points, using the initial solution as starting point for each local search (*A*,*B*). If a local optimum is not successful at completing the task ('x' in *A*,*B*), a new global search is initiated, creating a new starting point for neighboring local searches. We continue this iterative search until solutions have been found for all grid points. *C*, *D* show the resulting grid from *2D push* (section V-B), before and after our refinement process, respectively. Same colors indicate same families of motion.

V. Experiments and Results

A. Simulation environment

To evaluate the performance of our reflex-based approach, we set up a 2D simulation environment using the Box2D simulation engine [30]. We model a simple dual arm planar robot, which consists of a circular trunk and two 2DOF arms, as shown in fig. 5 *left*. Each task is implemented as an environment using the OpenAI Gym ecosystem [31].

B. Simulated pushing tasks

Learning to push off along one direction - 1D: We create two environments in each of which the robot has to learn to push off from an obstacle along a fixed direction (x-axis) in order to reach its goal.

In the first environment (**static 1D push**) the robot is placed in front of the obstacle with arms positioned in a pre-push configuration (flexed elbows). The initial position of the robot is sampled along the x-axis, from a uniform distribution $x_{trunk} \sim \mathscr{U}(0.77, 0.8)$, such that the obstacle surface is always

TABLE I

PERFORMANCE OF PPO AND REFLEX-BASED APPROACH, EVALUATED ON 10K EPISODES WITH RANDOM INITIAL CONFIGURATIONS. REFLEX-BASED APPROACH REPORTED FOR 5X5 AND 10X10 GRID SIZES.

Env	Metric	Reflex-10x10	Reflex-5x5	PPO
1D static	success rate	98.85%	93.77%	98.97%
	mean reward	-5.47 ± 2.12	-7.4 ± 3.2	-4.8 ± 2.3
	train time [s]	556.28	60.52	1444
1D	success rate	96.10%	92.94%	97.72%
	mean reward	-16.4 ± 5.9	-17.7 ± 7.3	-14.7 ± 5.8
	train time [s]	839.15	994.98	3449
2D	success rate	98.65%	93.74%	97.52%
	mean reward	-15.6 ± 3.7	-17.9 ± 6.0	-16.0 ± 4.8
	train time [s]	3361.93	1013.67	2876

located inside the reachable workspace of both arms. We further sample the goal position along the x-axis, such that $x_{goal} \sim \mathscr{U}(0.35, 0.6)$. Initially the robot is *static*, such that it needs to generate the entire momentum needed for reaching the goal through interaction with the obstacle.

For the second environment (**1D push**) the robot is approaching the obstacle with varying initial velocities. We apply a force $F_{init,x} \sim \mathscr{U}(6.0, 12.0)$ which accelerates the robot and sample the goal position from $x_{goal} \sim \mathscr{U}(0.0, 0.4)$.

For training a pushing configuration policy $\phi(\tilde{s})$ (eq. (1)) we restrict the kinematic configuration $\kappa = (\alpha = 0.0, d = 0.1)$ and arm configuration setpoints $l_\tau = [0, 1]$. This corresponds to a straight push to the front (fig. 2), and allows for a large effective length. We optimize for the remaining stiffness (k_p, k_d) and timing $(t_\tau = [t_0, t_1])$ parameters.

As described in section IV-D, we create a n-by-m grid of different initial conditions for each environment. For *static 1D push*, the grid axes represent possible goal locations and robot positions, and for *1D push* the axes represent possible goal locations and robot velocities.

Learning to push off along two directions - 2D: For this task, the robot needs to learn to push off from the obstacle surface at an oblique angle. It is accelerated by a constant force and we vary the goal position along both x and y-axis from $x \sim \mathscr{U}(0.2, 0.4)$, $y \sim \mathscr{U}(-0.2, 0.2)$. We optimize for configuration parameters $[\alpha_l, \alpha_r, l_{1,l}, l_{1,r}, t_0, t_1, k_p, k_d]$, and fix $d = 0.1$. Results are reported in table I.

C. Results

Table I shows the success rate, mean reward and train time for the three simulated tasks. We compare our reflex-based approach to PPO (stable-baselines PPO2 with MlpPolicy [32]) using the same OpenAI Gym environments. As shown, both methods achieve comparable results with respect to success rate and mean reward; this indicates that both techniques can generate diverse and flexible motions which generalize to previously unseen states. Train times generally increase with rising complexity of the tasks, and grid size in the case of the reflex-based approach.

For the 2D push task we find the solution grid shown in fig. 4-*D*. The corresponding robot arm motions are visualized in fig. 6 and fig. 7. The solution grid depicts motion families each marked by a distinct color and the corresponding lines of action (fig. 6) and joint angle trajectories (fig. 7) are colored accordingly. For comparison, we show the joint angle trajectories generated by PPO in the same gridpoints in fig. 8.

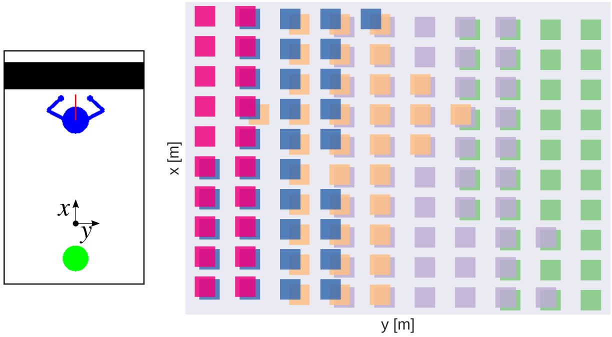

Fig. 5. Left: PushBot in Box2D. The robot's task is to reach the goal (green circle) by interacting with the obstacle. The front trunk orientation is marked by a red line. Right: Overlay of motion families for 10x10 solution grid for *2D push* (section V-B). Colors indicate different motion families. Points marked in the color of a motion family indicate that the robot successfully completed the task (reached the target) using a push generated by this family.

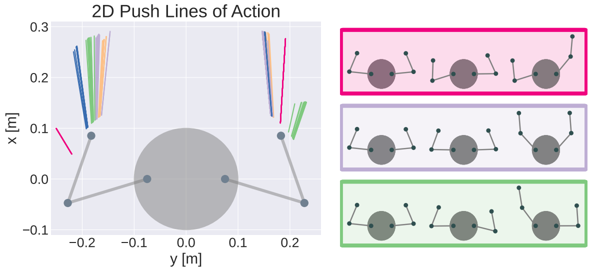

Fig. 6. Lines of actions for 10x10 solution grid for *2D push* (section V-B). Colors indicate different motion families, and three exemplary resulting arm movements are shown on the right.

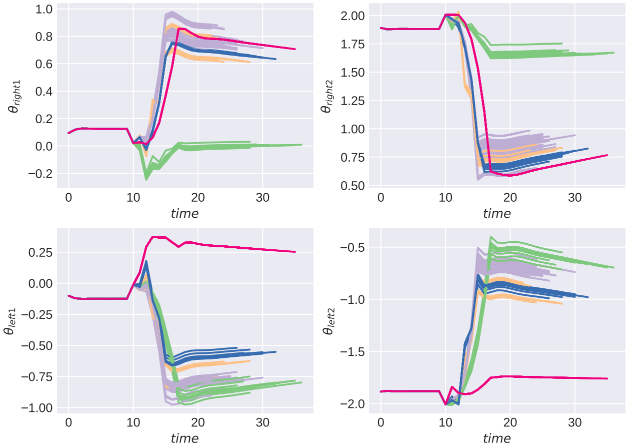

Fig. 7. 2D push: Joint angle trajectories produced by reflex approach. Colors correspond to grid points in fig. 4-D and lines of action in fig. 6.

VI. DISCUSSION

As shown in fig. 4-*D*, for the 2D task motion families are primarily distributed along the y-axis which for this task corresponds to goal locations ranging from left to right. We note that in fig. 4-*D* not all families are continuous across the grid. However, the underlying distribution of successful

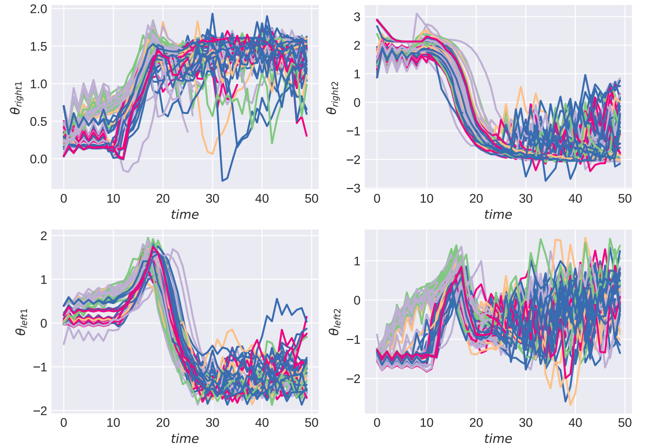

Fig. 8. 2D push: Joint angle trajectories produced by PPO. Colors correspond to the same grid points as the trajectories in fig. 7.

motions, visualized in fig. 5 *right*, shows that for most grid points multiple families can produce successful motions. For the final policy we select the family resulting in the highest reward for each grid point, which can lead to discontinuities. Alternatively, the space could be segmented into continuous families based on the results shown in fig. 5 *right*.

The observed joint angle trajectories are tightly clustered within each motion family as depicted in fig. 7. This indicates that arm motions are similar within the same motion family and that each family produces distinct behaviors. For instance, goal locations to the left require pushes using primarily the right arm (fig. 6 *magenta*). For central goal locations the robot uses both arms (*light purple*) and for locations to the right mainly the left arm is used (*green*). This can be especially useful for motion planning to solve tasks that require multiple pushes in sequence.

To compare our approach to PPO, we evaluate pushing behaviors generated by PPO for the same grid points as discussed previously, and plot the corresponding observed joint angle time series in fig. 8. Overall, PPO is able to generate diverse motions across the grid, however, as can be seen in fig. 8 the policy exploits the simulation physics resulting in unnatural and jerky motions. For example, the robot oscillates the arms to generate momentum towards the goal. Contrary to the reflex based approach, there is no distinct correlation between joint angle trajectories and goal locations. In addition to the typical caveats of sim2real (small changes in the environment or unknown obstacles could cause failure), this behavior is undesirable as it creates unpredictable and unsafe motions. Reward engineering and dynamics or domain randomization [33] could help to create more transferable RL policies, but would further decrease sample efficiency and significantly increase training time.

VII. Real robot experiments

To validate that we can directly transfer a learned control policy from simulation to the real robot, we use the PushBot robot described in section III to run the reflex controller learned for the *2D push* simulation task. In the experiment, we accelerate the robot towards an obstacle, and trigger the push off reflex controller when either piezo vibration sensor returns a binary contact signal. We select three goal locations to the right, left and center of the robot, and for each goal, we choose a corresponding pushing configuration from the motion families found in simulation. The resulting push off motions are shown in fig. 1 and in the supplemental video.

VIII. Conclusion and Future Work

In this work we investigate the problem of learning highly dynamic interactions. We present a reflex-based approach, and show that we are able to learn to push off in a simulated environment. Our approach achieves success rates comparable to state-of-the-art model-free RL, but produces distinct clusters of smooth and predictable arm trajectories. We show that these can directly be deployed on a real robot without further retraining and hand tuning.

In future, we hope to compare sim2real results from the reflex controller to those from PPO. The challenge is that sim2real with PPO requires that the real robot observes its full state at all times. In contrast, for the reflex controller, it is sufficient for the robot to know its initial velocity (fixed for our examples) and the goal position relative to contact. No observations were required for the reflex controller except to recognize when the contact event occurred, which was accomplished with sensors at the end effectors.

A second area of future work would be to explore inverse dynamics as an alternative to reflex control. The challenge is that inverse dynamics control requires the robot to maintain desired contact forces during the push. In contrast, our reflex controller learns and optimizes the effect of simple kinematic controls that are easy for the robot to achieve.

Finally, we look forward to scaling these results to more complex systems and scenarios. For the reflex controller, the state information required may be represented as velocity of the robot prior to contact and location of the goal in the local coordinate frame of the surface. Although the required state information has greater dimension than the examples shown in this paper, the number of additional parameters is not large. In contrast, unless similar abstractions are employed, the PPO algorithm must scale up to the full state space of the more complex robot and scenarios, and will have many more additional parameters. To further improve scalability of the reflex approach, we intend to explore adaptive sampling — the presence of continuous families allows us to make good estimates within large regions using simple interpolation of parameters. An adaptive sampling approach would evaluate random samples based on their interpolation result and only optimize a new solution if the interpolated result was inadequate. We look forward to testing these ideas with the CMU ballbot and its 7 DoF arms [34].

Acknowledgment

This research was supported by the National Science Foundation award CMMI-1925130.

References

[1] R. Tajima, D. Honda, and K. Suga, "Fast running experiments involving a humanoid robot," in *2009 IEEE International Conference on Robotics and Automation*, 2009, pp. 1571–1576.

[2] P. M. Wensing and D. E. Orin, "High-speed humanoid running through control with a 3d-slip model," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013, pp. 5134–5140.

[3] ——, "Development of high-span running long jumps for humanoids," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 2014, pp. 222–227.

[4] Y. Hong and B. Lee, "Evolutionary optimization for optimal hopping of humanoid robots," *IEEE Transactions on Industrial Electronics*, vol. 64, no. 2, pp. 1279–1283, 2017.

[5] A. H. Chang, C. M. Hubicki, J. J. Aguilar, D. I. Goldman, A. D. Ames, and P. A. Vela, "Learning terrain dynamics: A gaussian process modeling and optimal control adaptation framework applied to robotic jumping," *IEEE Transactions on Control Systems Technology*, pp. 1–16, 2020.

[6] T. Bretl, "Motion planning of multi-limbed robots subject to equilibrium constraints: The free-climbing robot problem," *The International Journal of Robotics Research*, vol. 25, no. 4, pp. 317–342, 2006. [Online]. Available: https://doi.org/10.1177/0278364906063979

[7] L. Sentis, J. Park, and O. Khatib, "Compliant control of multicontact and center-of-mass behaviors in humanoid robots," *IEEE Transactions on Robotics*, vol. 26, no. 3, pp. 483–501, 2010.

[8] J. Carpentier and N. Mansard, "Multicontact locomotion of legged robots," *IEEE Transactions on Robotics*, vol. 34, no. 6, pp. 1441–1460, 2018.

[9] J. Vaillant, A. Kheddar, H. Audren, F. Keith, S. Brossette, A. Escande, K. Bouyarmane, K. Kaneko, M. Morisawa, P. Gergondet, E. Yoshida, S. Kajita, and F. Kanehiro, "Multi-contact vertical ladder climbing with an hrp-2 humanoid," *Autonomous Robots*, vol. 40, 02 2016.

[10] M. Posa, S. Kuindersma, and R. Tedrake, "Optimization and stabilization of trajectories for constrained dynamical systems," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 1366–1373.

[11] Y. Zhao, U. Topcu, and L. Sentis, "High-level planner synthesis for whole-body locomotion in unstructured environments," in *2016 IEEE 55th Conference on Decision and Control (CDC)*, 2016, pp. 6557–6564.

[12] S. Kajita and K. Tani, "Study of dynamic biped locomotion on rugged terrain-derivation and application of the linear inverted pendulum mode," in *Proceedings. 1991 IEEE International Conference on Robotics and Automation*. Los Alamitos, CA, USA: IEEE Computer Society, apr 1991, pp. 1405,1406,1407,1408,1409,1410,1411. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/ROBOT.1991.131811

[13] A. Degani, S. Feng, H. B. Brown, K. M. Lynch, H. Choset, and M. T. Mason, "The parkourbot - a dynamic bowleg climbing robot," in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 795–801.

[14] D. Urieli, P. Macalpine, S. Kalyanakrishnan, Y. Bentor, and P. Stone, "On optimizing interdependent skills: A case study in simulated 3d humanoid robot soccer," vol. 2, 01 2011, pp. 769–776.

[15] S. Song and H. Geyer, "A neural circuitry that emphasizes spinal feedback generates diverse behaviours of human locomotion," *The Journal of physiology*, vol. 593, no. 16, pp. 3493–3511, 2015.

[16] R. Calandra, A. Seyfarth, J. Peters, and M. P. Deisenroth, "Bayesian optimization for learning gaits under uncertainty," *Annals of Mathematics and Artificial Intelligence*, vol. 76, no. 1–2, p. 5–23, Feb. 2016. [Online]. Available: https://doi.org/10.1007/s10472-015-9463-9

[17] A. Rai, R. Antonova, S. Song, W. Martin, H. Geyer, and C. Atkeson, "Bayesian optimization using domain knowledge on the atrias biped," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 1771–1778.

[18] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evolutionary computation*, vol. 9, no. 2, pp. 159–195, 2001.

[19] J. Morimoto, G. Cheng, C. G. Atkeson, and G. Zeglin, "A simple reinforcement learning algorithm for biped walking," in *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, vol. 3, 2004, pp. 3030–3035 Vol.3.

[20] G. Endo, J. Morimoto, T. Matsubara, J. Nakanishi, and G. Cheng, "Learning cpg-based biped locomotion with a policy gradient method: Application to a humanoid robot," *The International Journal of Robotics Research*, vol. 27, no. 2, pp. 213–228, 2008. [Online]. Available: https://doi.org/10.1177/0278364907084980

[21] X. B. Peng, G. Berseth, K. Yin, and M. Van De Panne, "Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning," *ACM Trans. Graph.*, vol. 36, no. 4, Jul. 2017. [Online]. Available: https://doi.org/10.1145/3072959.3073602

[22] N. Heess, D. TB, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, S. M. A. Eslami, M. A. Riedmiller, and D. Silver, "Emergence of locomotion behaviours in rich environments," *CoRR*, vol. abs/1707.02286, 2017. [Online]. Available: http://arxiv.org/abs/1707.02286

[23] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever, "Evolution strategies as a scalable alternative to reinforcement learning," 2017.

[24] S. Grillner, "Biological pattern generation: the cellular and computational logic of networks in motion," *Neuron*, vol. 52, no. 5, pp. 751–766, 2006.

[25] E. Bizzi and V. C. Cheung, "The neural origin of muscle synergies," *Frontiers in computational neuroscience*, vol. 7, p. 51, 2013.

[26] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017. [Online]. Available: http://arxiv.org/abs/1707.06347

[27] "Minisense 100 vibration sensor," https://www.sparkfun.com/datasheets/Sensors/Flex/MiniSense_100.pdf, accessed: 2021-09-12.

[28] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[29] R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu, "A limited memory algorithm for bound constrained optimization," *SIAM Journal on scientific computing*, vol. 16, no. 5, pp. 1190–1208, 1995.

[30] E. Catto, "Box2d: A 2d physics engine for games," *URL: http://www. box2d. org*, 2011.

[31] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *CoRR*, vol. abs/1606.01540, 2016. [Online]. Available: http://arxiv.org/abs/1606.01540

[32] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, "Stable baselines," https://github.com/hill-a/stable-baselines, 2018.

[33] 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," 2018. [Online]. Available: https://arxiv.org/pdf/1804.10332.pdf

[34] R. Shu and R. Hollis, "Development of a humanoid dual arm system for a single spherical wheeled balancing mobile robot," in *2019 IEEE-RAS 19th International Conference on Humanoid Robots (Humanoids)*, 2019, pp. 499–504.