SoMoGym: A toolkit for developing and evaluating controllers and reinforcement learning algorithms for soft robots

Moritz A. Graule[1†], Thomas P. McCarthy[1], Clark B. Teeple[1], Justin Werfel[1], and Robert J. Wood[1†]

***Abstract*—Soft robots offer a host of benefits over traditional rigid robots, including inherent compliance that lets them passively adapt to variable environments and operate safely around humans and fragile objects. However, that same compliance makes it hard to use model-based methods in planning tasks requiring high precision or complex actuation sequences. Reinforcement learning (RL) can potentially find effective control policies, but training RL using physical soft robots is often infeasible, and training using simulations has had a high barrier to adoption. To accelerate research in control and RL for soft robotic systems, we introduce SoMoGym (Soft Motion Gym), a software toolkit that facilitates training and evaluating controllers for continuum robots. SoMoGym provides a set of benchmark tasks in which soft robots interact with various objects and environments. It allows evaluation of performance on these tasks for controllers of interest, and enables the use of RL to generate new controllers. Custom environments and robots can likewise be added easily. We provide and evaluate baseline RL policies for each of the benchmark tasks. These results show that SoMoGym enables the use of RL for continuum robots, a class of robots not covered by existing benchmarks, giving them the capability to autonomously solve tasks that were previously unattainable.**

***Index Terms*—Soft Robot Applications, Modeling, Control, and Learning for Soft Robots, Reinforcement Learning**

I. Introduction

SOFT robots have received much attention in recent years as new developments in simulation, actuation, and control have shown the feasibility and utility of soft systems in applications including grasping [1]–[3], in-hand manipulation [4], [5], surgical tools [6], locomotion [7], [8], and wearable assistive devices [9]. Continuum manipulators in particular show a great deal of promise for manipulation tasks in unpredictable settings, as their compliance and conformability provide resilience to imperfect control policies and unexpected contact interactions with the environment. However, soft robots can be difficult to control effectively due to unmodeled effects of material fatigue and manufacturing variations, and their many degrees of freedom make it hard to plan trajectories with traditional approaches. Therefore, their ability to accomplish tasks requiring complex actuation sequences is still relatively underdeveloped.

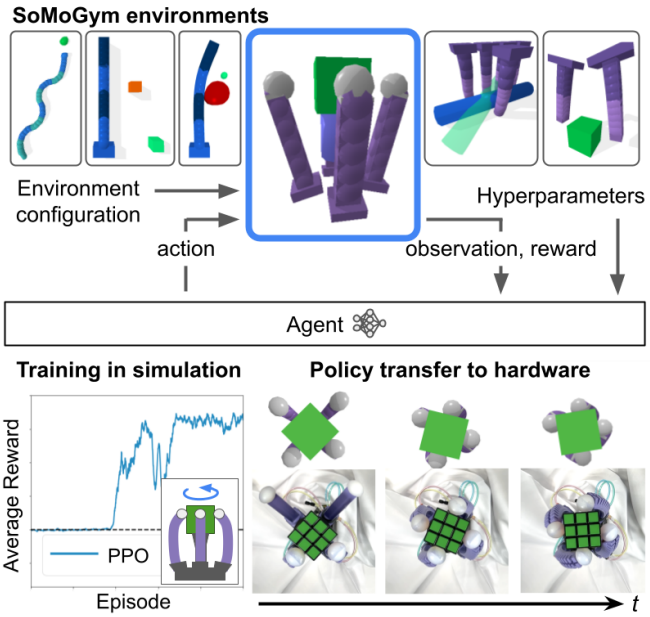

Fig. 1: **(top)** SoMoGym provides a collection of simulated environments in which soft robots execute locomotion, reaching, or manipulation tasks. It readily interfaces with standard reinforcement learning frameworks, enabling the training of performant control policies. We demonstrate how SoMoGym can extend the capabilities of soft robots by training baseline policies on all environments. **(bottom)** In one of the environments, a robot hand is rewarded for rotating a cube. We used proximal policy optimization (PPO) to train a policy on this task and successfully deploy it on hardware.

One tool that could accelerate progress in the field is a standardized, intuitive platform providing a fixed set of benchmark tasks for designing and comparing different control approaches. Moreover, the ability to use reinforcement learning (RL) with such systems would make it possible to develop and optimize continuum robot behaviors beyond the capabilities of hand-tuned control commands [10]. While it can be infeasible to manually prescribe suitable behaviors for every task and environment, an RL policy can adapt to novel situations, making it more useful for deployment across variable and diverse applications. This adaptability can include, for example, adjusting behavior to account for fatigue and partial failure in hardware [11]. However, accessible, scalable, and easily adaptable tools for RL with a broad range of soft robots currently do not exist.

In this paper, we present SoMoGym, an open-source toolkit for developing and evaluating control policies for soft robots. SoMoGym, unlike existing frameworks for simulating soft robotic control [12], [13], emphasizes easily configurable environments that support interaction with objects. The framework builds on the well-known and widely used OpenAI Gym

Manuscript received: September 9, 2021; Revised: December 14, 2021; Accepted: February 3, 2022.

This paper was recommended for publication by Editor Cecilia Laschi upon evaluation of the Associate Editor and Reviewers' comments.

This work was supported by the National Science Foundation (award number EFMA-1830901) and a Space Technology Research Institutes grant (number 80NSSC19K1076) from NASA's Space Technology Research Grants Program. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the funding organizations.

[1]All authors are with John A. Paulson School of Engineering and Applied Sciences, Harvard University, 150 Western Avenue, Allston MA 02134, USA.

[†]Correspondence to {graulem,rjwood}@g.harvard.edu

Digital Object Identifier (DOI): see top of this page.

environment class, and on our previous work developing the SoMo simulator [14]. SoMoGym permits experiments on the effects of varying control and robot design parameters, and also enables the use of RL for generating and tuning controllers for such systems. We provide a set of standard tasks which serve as a complete benchmarking suite, along with a set of baseline controllers found by RL for each of those tasks. Like OpenAI Gym, SoMoGym also supports the easy addition of custom tasks and environments.

The benchmarking suite comprises six distinct tasks (four of which provide two versions varying in difficulty or scope), together constituting a wide range of different types of activities encompassing reaching, manipulation, and locomotion. *Planar Reaching (Basic/With Obstacle)* requires the agent to move the tip of a four-actuator, planar manipulator to a goal position. *Planar Block Pushing* introduces object manipulation, where the same planar manipulator is tasked with moving a box to a goal position. Additional, more difficult, manipulation tasks and their variations (*Antipodal Gripper (Basic/Advanced)*, *In-Hand Manipulation (Basic/Inverted)*, and *Pen Spinning*) involve moving an object to a goal state using a system of multiple soft manipulators within a three-dimensional workspace. Finally, *Snake Locomotion (Basic/Advanced)* requires an untethered eight-actuator robot to move across a plane to a specified goal point.

We also present preliminary RL-generated policies for our benchmark tasks. These serve as a performance baseline for the benchmarks and demonstrate the framework's utility. While higher performance on these tasks is surely possible (especially after extensive hyperparameter tuning), our results provide examples upon which to base future work.

In summary, we make the following contributions:

- We release SoMoGym, an open-source framework to train and test soft robot control policies on complex tasks where robots interact with their environments, frequently making and breaking contact.
- We provide a set of standard benchmark tasks, spanning challenges of manipulation, locomotion, and reaching in environments both with and without obstacles.
- We demonstrate the utility of SoMoGym for RL by training performant control policies for each of these tasks, and provide a set of effective training and simulation parameters for several RL algorithms.
- We show that in-hand manipulation policies trained in SoMoGym simulations readily transfer to the real world by successfully executing them on physical hardware.

II. Related Work

A. Reinforcement Learning Benchmarks

In the RL paradigm, an agent learns a policy that prescribes what action to take for any given situation, based on rewards received during interactions with its environment (for a detailed review, see [15]). With variations in environments and their reward structures, an RL agent's performance can be highly task-dependent. It is therefore best to evaluate RL algorithms in a range of different environments.

OpenAI presented one of the first comprehensive collections of standardized test problems for RL when they introduced *OpenAI Gym* [16]. OpenAI Gym includes a range of control tasks for simulated robots and other agents, in environments that share a standardized and intuitive interface.

Driven by the framework's utility, recent projects have introduced additional robotic RL benchmarking toolkits built on OpenAI Gym. Notable projects include *Assistive Gym* (common robot platforms assist humans in activities of daily living) [17], *SoftGym* (rigid robots manipulate deformable objects like cloth and rope) [18], *Air Learning* (for training and evaluating control policies for aerial vehicles in photorealistic environments) [19], *Isaac Gym* (providing GPU implementations of common RL algorithms) [20], *SURREAL* (focusing on traditional robot manipulation tasks like block stacking and peg insertion) [21], and an extension of OpenAI Gym for ROS and the Gazebo simulator (providing the flexibility of those systems) [22].

Together, the environments in these frameworks cover a diverse set of robotic challenges. However, their tasks are executed exclusively by traditional rigid robots. A noteworthy exception is *Elastica*, which couples a simulator for Cosserat rods with a typical RL framework and achieves impressive results on several tasks for continuum robots [13]. However, Elastica is limited to the simulation of rod-like structures. Therefore, it remains hard to develop, select, and fine-tune suitable RL algorithms for soft robotic systems that undergo rich contact interactions with complex environments.

B. Physics Simulators

At the heart of every benchmark for RL on simulated robot agents lies a physics engine that computes how the agent interacts with its environment. RL benchmarks involving traditional robots commonly rely on DART [23], Bullet/PyBullet [24], or MuJoCo [25] for the physics simulation.

Some systems implement physics simulation de novo rather than using an engine as above. Elastica [13] and a recent framework by Huang et al. [26] are well-suited for the simulation of environments that exclusively contain rod-like structures. ChainQueen is a simulator for fully soft systems [27]. SOFA is well-suited for systems with a wide range of material properties, but is computationally expensive, as it employs a full finite element analysis [12], [28].

SoMo, a thin wrapper around PyBullet, is designed specifically for the simulation of soft/rigid hybrid systems [14]. It facilitates the simulation of continuum manipulators by approximating them through rigid-link systems with spring-loaded joints and has been shown to accurately capture soft manipulator behavior. It is well-suited as the underlying engine for an RL benchmarking suite for soft/rigid hybrid tasks since: (i) its high-level interface enables the construction of varied environments that contain soft robots and other objects with complex contact interactions; (ii) its rigid-link approximations make it fast—a crucial feature given the large amounts of data required in state-of-the-art RL algorithms.

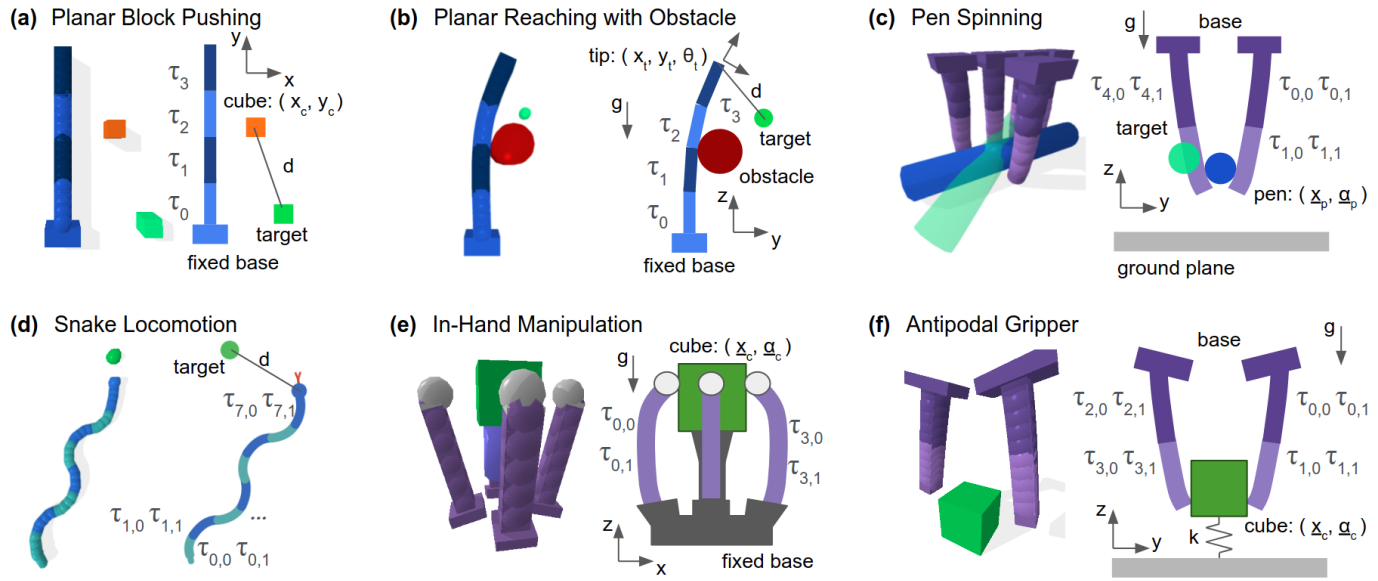

Fig. 2: SoMoGym provides a suite of benchmark tasks with soft continuum manipulators. The environments have a standardized interface similar to the one employed in OpenAI Gym, allowing for straightforward integration with common reinforcement learning frameworks. These tasks include: **(a)** In *Planar Block Pushing*, a planar continuum manipulator is tasked to move a cube to a goal position. The manipulator has four independently controlled actuators with torques τ_i for $i \in [0,1,2,3]$. **(b)** In *Planar Reaching with Obstacle*, the same manipulator must move its tip to a goal pose in the presence of a cylindrical obstacle. **(c)** In *Pen Spinning*, an anthropomorphic soft hand holds a cylindrical object that needs to be moved to the desired pose. The hand has five fingers, each of which has two actuators with two independently controlled torques per actuator. **(d)** In *Snake Locomotion*, a snake-like robot has to move towards a goal. The snake robot has eight independently controlled actuators with two active axes each with torques $(\tau_{i,0}, \tau_{i,1})$ for $i \in [0,1,2,...,7]$. **(e)** In *In-Hand Manipulation*, a soft hand with four fingers is tasked to rotate a cube around the z-axis. Each of the fingers has two axes with bending torques ($\tau_{i,0}$ and $\tau_{i,1}$). **(f)** In *Antipodal Gripper*, a two-finger manipulator is tasked to grasp and lift a block connected to the ground by a spring with stiffness k. Each of the fingers has two actuators with two bending axes per actuator (controlled by torques ($\tau_{i,0}$ and $\tau_{i,1}$)).

III. SoMoGym

A. Overview

SoMoGym provides a modular and straightforward way to define complex environments as control tasks for soft robots, allowing for easy evaluation of control policies and learning frameworks. It contains a set of predefined environments with prespecified settings for use as a policy benchmarking suite. The gym, built in Python3, consists of environments that inherit from the environment (or 'Env') class in OpenAI Gym. The format of OpenAI Gym environments is a standard for defining RL tasks, making SoMoGym's structure easy to understand within the RL community. During initialization, a SoMoGym environment defaults to its provided benchmark configuration. To initialize an environment with custom settings, a configuration dictionary can be provided with custom general simulation parameters (PyBullet time step, controller update rate, maximum torque for each actuator, etc.) and task-specific parameters (reward components and weights, observation components, object properties, etc.). SoMoGym also provides a variety of observation and reward functions, which can be selected, combined, or modified easily.

SoMoGym environments contain soft manipulators, rigid obstacles and/or freely moving objects, and, optionally, traditional robots. Internally, SoMoGym relies on the SoMo framework [14] to represent and control soft robots. As such, each continuum robot is described in a human-readable configuration file, making it easy to vary the number, poses, and properties of soft robots in each SoMoGym environment. As SoMo simulation results translate to physical systems with high fidelity [14], [29], [30], control policies generated with SoMoGym readily transfer to the physical world.

B. Environments

SoMoGym includes a suite of simulation environments in which continuum robots execute reaching, manipulation, and locomotion tasks (Fig. 2). The standard configurations of these environments form the SoMoGym benchmarking suite. Summaries of these benchmark tasks follow (detailed descriptions can be found in the SoMoGym repository [31]).

Planar Block Pushing: A planar continuum robot must move a cube from an initial position to a goal position. The robot consists of four serial actuators with one actively controlled bending axis per actuator. The reward at each action step is proportional to the distance between the cube and the goal position.

Planar Reaching: The same robot as above has to move its tip to a randomly chosen goal pose (i.e., position and orientation). The robot is rewarded for bringing its tip close to the goal pose.

Planar Reaching with Obstacle: identical to *Planar Reaching* but with a cylindrical obstacle present.

Pen Spinning: A downward-facing, anthropomorphic hand holds a long cylinder within a gravitational field and is required to reorient it to align with a random goal pose. The hand consists of a rigid, fixed palm, and five continuum manipulator fingers. Each of the fingers consists of two actuators, each of which has two actively controlled bending axes. The robot is rewarded for manipulating the cylinder such that its final pose matches the goal pose.

Snake Locomotion: A robot consisting of eight actuators with two independently actuated bending axes each, exhibiting anisotropic friction with the ground plane, is rewarded for successfully moving towards a (fixed) goal.

Snake Locomotion Advanced: identical to *Snake Locomotion* but with the goal location sampled uniformly at random from a square on the ground plane.

In-Hand Manipulation: A rigid-soft (hybrid) robotic hand pointing upward holds a cube within a gravitational field and aims to rotate the cube around the z-axis. The hand consists of a rigid palm and four soft fingers. Each finger consists of one actuator with two independently controlled bending axes. The robot is rewarded for rotating the cube counterclockwise without dropping it, and can optionally be rewarded for keeping the cube centered above the center of the palm. Hardware instantiations of this system have been presented previously [4], [14], [29].

Inverted In-Hand Manipulation: identical to *In-Hand Manipulation* but inverted, such that the cube lies between the ground plane and the robot's rigid palm. This reduces the likelihood of catastrophic failures (e.g., dropping the cube).

Antipodal Gripper: A downward-facing antipodal gripper is tasked with picking up a cuboid or cylindrical object with given width or diameter. The gripper consists of a rigid palm connected to two continuum manipulator fingers, each consisting of two actuators with two actively controlled bending axes per actuator. The gripper assembly can move along the z-axis. An external spring force in the z-direction is applied to the object, proportional to the object's height above the ground plane. The grip strength required to retain the object accordingly increases with increasing object height. At each action step, the robot earns a reward proportional to the object's height.

Antipodal Gripper Advanced: identical to *Antipodal Gripper* but the size and position of the gripped object are varied for each trial.

IV. Control and Policy Learning

In SoMoGym, soft actuators are represented using the SoMo framework. As such, they are approximated by serially linked rigid segments connected by spring-loaded joints. These actuators are controlled via torque control, i.e., the torque applied to each joint is the sum of spring reaction force and actuator control torque (for details, see [14]). In our RL setting, the action space usually represents normalized torque for each actively controlled degree of freedom (DOF) of each actuator. In *Antipodal Gripper* environments, a final action dimension is reserved for controlling change in the height of the manipulator's base position. Each dimension of the action space can take a value from -1 to 1, and we map from action values to actuator torques using a constant multiplier (provided in the run configuration). This value can be specified along with manipulator physical characteristics to reflect the dynamics and capabilities of a given physical system.

In controlling robots in SoMoGym, we consider two distinct time step durations: action time and simulation time. Simulation time represents how often the PyBullet simulation is updated; action time represents how often a new policy input is applied and is the inverse of the controller update rate. The simulation time has to be small enough to avoid numerical instabilities; the magnitude of action time determines the precision over time a policy can exhibit, and depends on the update rate of the real-world controller. We found that a simulation time of 0.2 milliseconds works well in our examples, and selected an action time of 0.01 seconds.

Instantaneously applying a new actuator torque that is significantly different from the previous torque is (i) commonly not possible in physical systems, and (ii) can lead to numerical instability in simulated systems. In order to achieve stable and smooth motion as actions are applied, we limit the change in torque per simulation step. This also helps us achieve realistic system behavior; hardware systems have actuation rate limits (e.g., caused by fluidic resistance/capacitance in fluidic actuators). In the simulation, applied torques are ramped linearly according to a torque rate limit (defined in the run configuration) until the commanded torque is achieved or the next torque command is issued.

SoMoGym permits a selection of state observation options for each environment. These observations are chosen based on what may be available on a hardware system or useful in evaluating the impact of providing more complete state information to an agent. For the continuum robots in our environments, we commonly observe backbone positions, joint curvatures, joint angles, link velocities, and applied torques. If an external object is involved, we commonly observe object position, orientation, velocity, and distance between object and robot tip. Run configurations define which observation options to use during training as the observation function. This allows for easy comparison of the performance of different observation spaces, helping us pursue useful questions like optimal sensor placement.

Reward functions are similarly separated into components selectable by run configurations. In *Planar Block Pushing*, for example, the object's orientation, position, and distance from the manipulator tip might each contribute to the reward. The run configuration specifies the weight of each component in the cumulative reward. This structure allows for the comparison of different reward functions, and makes it easy to evaluate the performance of policies along common axes in the case that reward functions vary across an experiment.

A. Training Baselines

RL algorithms vary in their prioritization of exploration versus exploitation, and in their treatment of previously collected data. Based on how they treat this data, algorithms can be grouped into *on-* and *off-policy* learners. On-policy learners use the data to estimate the expected return for the current policy, using only actions chosen by that policy. Off-policy algorithms estimate the return for an unknown optimal policy, using actions chosen under the current and previously considered policies. Algorithms of the latter type can have better sample efficiency, but typically lack convergence guarantees. The optimal exploration-exploitation tradeoff and ideal treatment of collected data are task dependent, and different

RL algorithms can develop disparate behaviors to solve a given task. Thus, we provide baseline controllers for three algorithms spanning the generally-accepted state-of-the-art in deep RL: on-policy algorithm Proximal Policy Optimization (PPO); and off-policy algorithms Twin Delayed DDPG (TD3) and Soft Actor Critic (SAC), which learn deterministic and stochastic policies, respectively.

Developed by OpenAI to be both performant and easy to use, PPO is an on-policy algorithm that has been shown to outperform competing algorithms while having a simpler implementation [32]. It allows multiple workers to run concurrently for improved efficiency. During training, optimization is performed to maximize an objective function that encourages actions with a positive advantage (meaning that the selected action performs better than simply executing the current policy) while limiting policy changes per update.

TD3, an improved version of Deep Deterministic Policy Gradient (DDPG) [33], is an off-policy algorithm training a deterministic policy. DDPG, inspired by the Q-learning algorithm [34], simultaneously learns a Q-function (which outputs the expected future reward given a state and action) and an optimal policy; off-policy data is used to learn the Q-function, and the Q-function is used to optimize the policy. TD3 makes this process more robust in three ways: (1) in order to prevent overestimating Q-values, two Q-functions are learned and the smaller Q-value is chosen for each update; (2) the policy estimator is updated less frequently than the Q-function; (3) noise is added when computing the target action, reducing the impact of Q-function errors.

Like TD3, SAC is an off-policy algorithm that learns two Q-functions alongside its policy [35]. Uniquely, SAC learns a stochastic policy while optimizing the tradeoff between expected reward and entropy. Increased entropy allows better exploration of the state-action space, which can prevent the algorithm from getting caught in local optima. SAC automatically tunes the ratio of expected return versus entropy using two Q-functions (similar to the double Q-learning in TD3), directly controlling the explore-exploit tradeoff.

We use the Stable Baselines3 [36] implementation of these algorithms, and represent policies and action-value functions with the algorithm-specific default multilayer perceptron network architecture. As these experiments are mainly intended to show the utility of SoMoGym, hyperparameter variations were not explored exhaustively. Better performance may be achieved with further hyperparameter tuning, and the improvement may be more pronounced in TD3, as it is more sensitive to hyperparameter selection.

V. Reinforcement Learning Evaluation

In our experiments, we ran several RL training runs to build baseline control policies for all of the benchmark tasks presented in Section III-B using each of the three algorithms (PPO, SAC, and TD3). For each task, we trained policies for 10^7 action steps with three different random seeds. Each training run used four modern CPU cores. Each task required between two and 72 hours of training for the rewards per episode to stabilize across all algorithms, depending greatly on task and simulation complexity. For PPO training runs, we ran four environments in parallel.

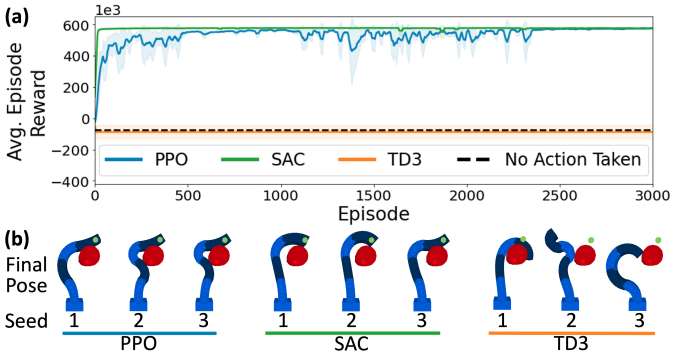

Fig. 3: *Planar Reaching with Obstacle* training results. **(a)** Reward history for the first 3000 training episodes for each algorithm. Curves are the average smoothed reward (window size 20) across three random seeds, and shaded regions indicate plus/minus one standard deviation. **(b)** Final pose from rollouts of the best policy developed by each seed of each algorithm. PPO and SAC learned similar policies, each reaching the (green) goal in a reliable and stable manner. Some of these policies brace against the (red) obstacle, while others reach around it. The policies trained with TD3 do not succeed in reaching the goal.

At least one algorithm achieved reasonable success on each task, demonstrating the feasibility of RL for soft robots with SoMoGym; in many cases these baseline policies exhibited interesting multi-step manipulation behaviors like re-grasping and finger gaiting. As expected, on-policy algorithm PPO generally took longer to converge than off-policy algorithms SAC and TD3. SAC succeeded in all tasks; PPO solved most tasks well, but typically achieved lower rewards than SAC. TD3 often did not achieve a relevant improvement throughout training and would likely require more extensive hyperparameter tuning for good performance.

Below, we present RL results for selected tasks; complete baseline results and detailed information for all environments are available with the SoMoGym codebase [31]. For each task, we visualize the training performance of each algorithm by plotting the reward history (i.e., the average reward accumulated per episode versus episode number) averaged across seeds. For each seed and algorithm, we filter the reward history (moving average with a window size of 20 episodes), and plot the mean and standard deviation of these smoothed values across seeds. Our truncated plots show the intervals during which the reward changed significantly.

Planar Reaching with Obstacle: In this task, the agent observes manipulator backbone positions, link velocities, vector d from tip to goal, latest applied actuation torques, and goal position. Reward is calculated as $R = -10|d| + \psi(d)$, where $\psi(d)$ is a three-level bonus achieved when the tip is brought into close proximity of the goal.

The training runs on this task converged after approximately 800 episodes for PPO, and 25 episodes for SAC and TD3. PPO and SAC both developed successful policies, while TD3 did not succeed (Fig. 3a). Performance across TD3 random seeds was highly variable, with most seeds developing behaviors that fail completely. SAC and PPO converged at a similar rate initially, but SAC more quickly learned behaviors that earn higher proximity bonuses, and more variance was present

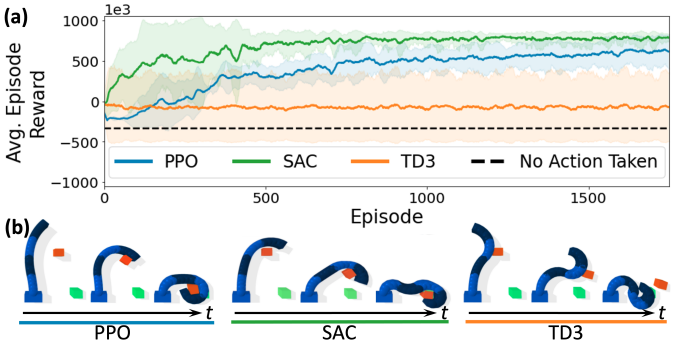

Fig. 4: *Planar Block Pushing* training results. **(a)** Reward history for the first 2000 training episodes for each algorithm. Curves are the average smoothed reward (window size 20) across three random seeds, and shaded regions indicate plus/minus one standard deviation. **(b)** Snapshots of policy rollouts over one episode representative of the best reliable behaviors developed by each algorithm. TD3 does not move the object (orange) all the way to the green goal position, while both PPO and SAC grasp and position the box successfully.

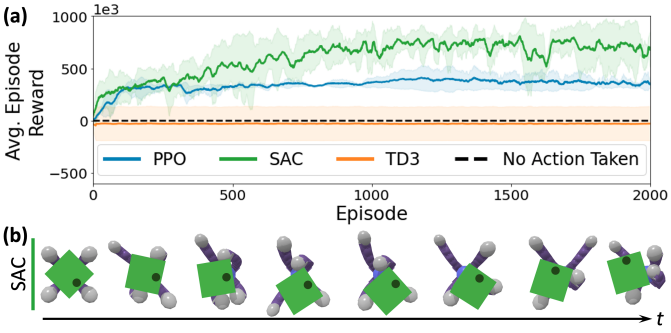

Fig. 5: *Inverted In-Hand Manipulation* training results. **(a)** Reward history for the first 3000 training episodes for each algorithm. Curves are the average smoothed reward (window size 20) across three random seeds, and shaded regions indicate plus/minus one standard deviation. **(b)** Snapshots of an SAC policy rollout over one episode looking up at the block from underneath the table's surface. This shows the hand successfully reorienting the block nearly 180° using a series of grasps and re-grasps.

in the performance of PPO over episodes. Qualitatively, PPO and SAC achieved similar results, reliably reaching the target position with the manipulator tip across all policy rollouts.

Planar Block Pushing: This task requires complex behaviors that use part of the manipulator to grasp the box while the other part maneuvers it into place. The agent observes manipulator backbone positions, link velocities, and latest applied actuation torques; vector d_{bt} from box to tip; box position, orientation, and velocity; vector d from box to goal; and goal position. Reward is given by $R = -10|d| - 10|d_{\text{bt}}| + \psi(d)$, where the first term encourages contact and $\psi(d)$ is a three-level bonus achieved when the box is close to the goal.

The training runs on this task converged after approximately 1600 episodes for PPO, 800 episodes for SAC, and 25 episodes for TD3. TD3 plateaued quickly but achieved sub-optimal rewards in two of three seeds. SAC and PPO took longer to stabilize, but developed more successful behaviors across all seeds. All successful policies learned to grasp or cup the box with the distal segments of the manipulator. This behavior was incentivized by reward shaping, as we penalize the distance between manipulator tip and target position. Where the policies of TD3 and PPO seeds learned to carry out the task in a single motion, some SAC seeds learned to do so in two distinct steps. As seen at the bottom of Fig. 4, the manipulator first hook-grasps the box with the two distal actuators and pushes it down, and then pulls it back up to reach the target using the other segments.

Inverted In-Hand Manipulation: One of SoMoGym's most complex benchmark tasks, this rewards a hand with soft robotic fingers for rotating a box counterclockwise on a table as far as possible (up to 180°). Reward is $R = 100(\phi - \phi_0) - |\psi - \psi_0| - |\theta - \theta_0| - 10|x - x_0|$, where $\psi - \psi_0$, $\theta - \theta_0$, and $\phi - \phi_0$ are the differences in Euler angles between the box's start and current orientations, and $x - x_0$ is the change in position between the box's start and current positions. The observation function returns manipulator backbone positions and link velocities; latest applied actuation torques; and box position, velocity, and orientation.

TD3 runs plateaued after approximately 25 episodes, though most seeds did not learn successful behaviors. Conversely, most policies from PPO and SAC achieved reliable rotation of the box by at least 45°. PPO's learning progress decreased significantly after approximately 100 episodes, and SAC's reward gradually surpassed it until plateauing after approximately 800 episodes with significantly higher rewards earned per episode.

Early on during training, SAC and PPO policies exhibited a single-step behavior: grasping the box with at least two fingers and rotating until reaching the torque limit. While stable and repeatable, this approach has its limitations as box rotation is restricted to approximately ±45°. SAC and PPO eventually developed more complex, multi-step manipulation behaviors (Fig. 5b). SAC policies showing this behavior rotated the box further than those of PPO. After an initial grasp and box rotation, the box is re-grasped and rotated by second and third sets of fingers to continue rotation. While this is not always successful in every policy rollout, it achieves far superior performance compared to simpler behaviors. Though these high-performing policies achieve a box rotation of nearly 180°, hyperparameter tuning, potentially on memory-related parameters like replay buffer size, could likely lead to the development of more complex or periodic gaiting behaviors that succeed more elegantly.

Snake Locomotion: This task requires highly coordinated behavior between actuators; if actuation is improperly sequenced, the robot does not move towards the target. The goal position for the robot tip is approximately ten body-lengths from the initial head position, and the most effective policy for covering that distance is to carry out one motion primitive several times (e.g., periodic slithering). TD3 and PPO did not learn successful policies; SAC reliably learned successful policies across all seeds, plateauing after approximately 2000 episodes. These policies produced a slithering-like motion, enabling the snake to reach the target and stop after arrival (Fig. 6b). This behavior was especially pronounced early in episodes; as the snake approaches the goal, its motion becomes more erratic and less snake-like. The likely cause is more-thorough exploration of states closer to the start.

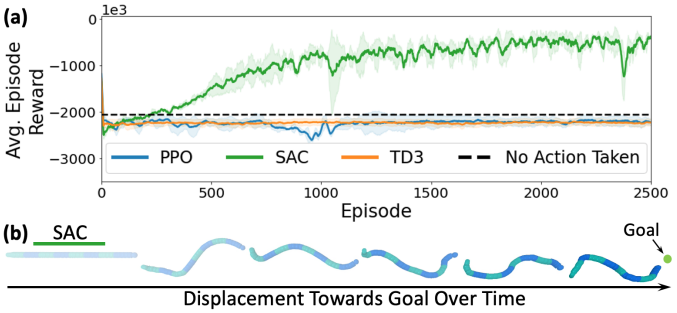

Fig. 6: *Snake Locomotion* training results. **(a)** Reward history for the first 2000 training episodes for each algorithm. Curves are the average smoothed reward (window size 20) across three random seeds, and shaded regions indicate plus/minus one standard deviation. **(b)** Snapshots of snake over SAC policy rollout showing *slithering* behavior.

VI. Hardware Validation

In order to verify that policies trained with SoMoGym are directly useful in solving tasks in the physical world, we selected a policy trained on the *In-Hand Manipulation* environment and applied it to a hardware instantiation of said task. This hardware system consists of four soft pneumatic fingers arranged symmetrically around a flat cylindrical palm, as described in [4]. Each finger is actuated using differential pressure in two parallel chambers spanning its length.

When building the simulation for this system, we first calibrated the simulation parameters (i.e., link mass and joint stiffness). The following outlines this process for a planar physical continuum actuator (an extension to two bending axes is trivial). For an actuator with length L, one must first choose the number of rigid segments (N) used to approximate the actuator. The mass of each segment can be obtained from measurements, finite element analysis (FEA), or design parameters. Given N, the stiffness k of each of the spring-loaded joints can be obtained from the actuator's flexural rigidity (i.e., the product of Young's modulus E and second moment of area I) as $k = EIN/L$ (for sufficiently large N). The actuator's flexural rigidity, in turn, can be obtained either from design variables, FEA, or experimental characterization. For a clamped-free Euler-Bernoulli beam without gravity, the flexural rigidity can be identified experimentally by measuring the tip deflection δ_t under known tip load F_t, as $EI = F_t L^3/(3\delta_t)$. Finally, as the control input in SoMo and SoMoGym is the internal bending moment for each actuator, the relationship between physical control input (e.g., cable tension or fluid pressure) to internal bending moment (and vice versa) needs to be characterized. This step is actuator-specific; we typically do so through tip deflection and blocked-force experiments [14].

Following this calibration, we can instantiate a simulated system equivalent to our hardware setup. In simulation, our action space then represents actuation torques in each controllable bending direction of the soft fingers; each finger has a control input for forward/backward motion (grasping torque), and a control input for lateral motion (side-to-side torque). Using the calibration procedure described in [14] and outlined above, we find the following linear map between real differential actuation pressures p_{diff} and simulated orthogonal actuation torques τ (with $w_0 = 1.0$ and $w_1 = 0.31$):

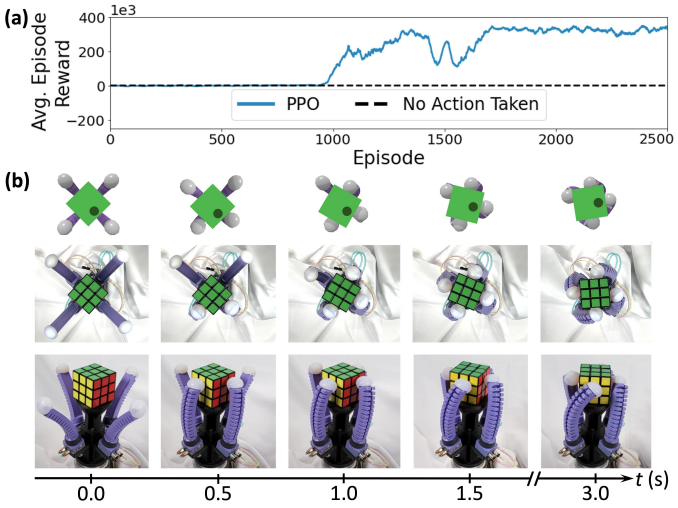

Fig. 7: **(a)** Reward history for the first 2,500 episodes of an *In-Hand Manipulation* training run with PPO. **(b)** Rollout of the final PPO-trained *In-Hand Manipulation* policy in simulation and hardware.

$$\begin{bmatrix} \tau_{\text{grasp}} \\ \tau_{\text{side}} \end{bmatrix} = \begin{bmatrix} p_{\text{diff},0} & p_{\text{diff},1} \\ p_{\text{diff},0} & -p_{\text{diff},1} \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \end{bmatrix} \tag{1}$$

With this mapping, we can transfer a simulated trajectory to the hardware system. We additionally restrict applied torques in hardware to the torque range of the fingers (to prevent finger failure or excessive fatigue), so any torque commanded by a policy that is outside of that range will be applied as the nearest valid torque value.

Fig. 7 shows the reward history of a selected training run, along with snapshots of the resulting policy executed in simulation and hardware. The system's states in simulation and hardware match closely throughout the policy rollout (Supplemental Video).

VII. Conclusion and Outlook

We introduced SoMoGym, a tool for controller evaluation and RL training for continuum robots performing reaching, manipulation, and locomotion tasks in nontrivial environments. We provided a predefined collection of tasks with parameter presets, forming the SoMoGym benchmarking suite to train, evaluate, and compare RL agents for soft robots. We integrated SoMoGym with a common RL framework and presented learned baseline policies for all tasks within the benchmarking suite. The environments in SoMoGym are straightforward to customize, with a wide range of possible observation spaces and reward functions provided for each task. They also provide a well-documented set of examples and utilities that users can build upon when creating simulated RL environments for their soft robots. Finally, we showed that policies learned in SoMoGym simulations can be transferred to hardware systems, achieving good performance in hardware with only minor calibration.

In future work, we will conduct a more extensive study on the performance of various RL algorithms on the SoMoGym benchmark tasks under careful hyperparameter tuning. We will combine this with a rigorous exploration of the effect of different observation spaces, action spaces, and reward shaping

on the performance achieved. Summarizing the insights gained in these explorations, we will leverage SoMoGym to provide a set of recommendations on how to set up RL tasks with continuum robots to achieve effective results in simulation and hardware. Together with the utility of SoMoGym, these advances will drive the adoption of RL in soft robotics, enabling soft robots to autonomously complete previously unattainable tasks.

Finally, we aim to further expand the capabilities of soft robots through a holistic and integrative approach to hardware and controller design. As part of this work, we will extend SoMoGym to allow for the algorithmic variation of robot design parameters (in addition to control policies), and explore how we can optimize the design of continuum robots to create systems that are easier to control and more amenable to training with RL.

Software Availability

All source code for SoMoGym is available online [31].

References

[1] C. Choi, W. Schwarting, J. DelPreto, and D. Rus, "Learning object grasping for soft robot hands," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 2370–2377, 2018.

[2] D. M. Vogt, K. P. Becker, B. T. Phillips, M. A. Graule, R. D. Rotjan, T. M. Shank, E. E. Cordes, R. J. Wood, and D. F. Gruber, "Shipboard design and fabrication of custom 3d-printed soft robotic manipulators for the investigation of delicate deep-sea organisms," *PLoS One*, vol. 13, no. 8, p. e0200386, 2018.

[3] R. Deimel and O. Brock, "A novel type of compliant and underactuated robotic hand for dexterous grasping," *The International Journal of Robotics Research*, vol. 35, no. 1-3, pp. 161–185, 2016.

[4] S. Abondance, C. B. Teeple, and R. J. Wood, "A dexterous soft robotic hand for delicate in-hand manipulation," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 5502–5509, 2020.

[5] A. Bhatt, A. Sieler, S. Puhlmann, and O. Brock, "Surprisingly robust in-hand manipulation: An empirical study," *Robotics: Science and systems (RSS) Conference*, 2021.

[6] M. McCandless, A. Perry, N. DiFilippo, A. Carroll, E. Billatos, and S. Russo, "A soft robot for peripheral lung cancer diagnosis and therapy," *Soft Robotics*, 2021.

[7] R. F. Shepherd, F. Ilievski, W. Choi, S. A. Morin, A. A. Stokes, A. D. Mazzeo, X. Chen, M. Wang, and G. M. Whitesides, "Multigait soft robot," *Proceedings of the national academy of sciences*, vol. 108, no. 51, pp. 20 400–20 403, 2011.

[8] N. W. Bartlett, M. T. Tolley, J. T. Overvelde, J. C. Weaver, B. Mosadegh, K. Bertoldi, G. M. Whitesides, and R. J. Wood, "A 3d-printed, functionally graded soft robot powered by combustion," *Science*, vol. 349, no. 6244, pp. 161–165, 2015.

[9] P. H. Nguyen, I. I. Mohd, C. Sparks, F. L. Arellano, W. Zhang, and P. Polygerinos, "Fabric soft poly-limbs for physical assistance of daily living tasks," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 8429–8435.

[10] S. Bhagat, H. Banerjee, Z. T. Ho Tse, and H. Ren, "Deep reinforcement learning for soft, flexible robots: Brief review with impending challenges," *Robotics*, vol. 8, no. 1, p. 4, 2019.

[11] A. Cully, J. Clune, D. Tarapore, and J.-B. Mouret, "Robots that can adapt like animals," *Nature*, vol. 521, no. 7553, pp. 503–507, May 2015. [Online]. Available: https://doi.org/10.1038/nature14422

[12] F. Faure, C. Duriez, H. Delingette, J. Allard, B. Gilles, S. Marchesseau, H. Talbot, H. Courtecuisse, G. Bousquet, I. Peterlik, and S. Cotin, "SOFA: A Multi-Model Framework for Interactive Physical Simulation," in *Soft Tissue Biomechanical Modeling for Computer Assisted Surgery*, ser. Studies in Mechanobiology, Tissue Engineering and Biomaterials, Y. Payan, Ed. Springer, Jun. 2012, vol. 11, pp. 283–321. [Online]. Available: https://hal.inria.fr/hal-00681539

[13] N. Naughton, J. Sun, A. Tekinalp, T. Parthasarathy, G. Chowdhary, and M. Gazzola, "Elastica: A compliant mechanics environment for soft robotic control," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 3389–3396, 2021.

[14] M. A. Graule, C. B. Teeple, T. P. McCarthy, R. S. Louis, G. Kim, and R. J. Wood, "SoMo: Fast and accurate simulations of continuum robots in complex environments," in *2021 IEEE International Conference on Intelligent Robots and Systems (IROS), p. In Review, IEEE*, 2021.

[15] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[16] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[17] Z. Erickson, V. Gangaram, A. Kapusta, C. K. Liu, and C. C. Kemp, "Assistive gym: A physics simulation framework for assistive robotics," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 10 169–10 176.

[18] X. Lin, Y. Wang, J. Olkin, and D. Held, "Softgym: Benchmarking deep reinforcement learning for deformable object manipulation," *arXiv preprint arXiv:2011.07215*, 2020.

[19] S. Krishnan, B. Boroujerdian, W. Fu, A. Faust, and V. J. Reddi, "Air learning: a deep reinforcement learning gym for autonomous aerial robot visual navigation," *Machine Learning*, vol. 110, no. 9, pp. 2501–2540, 2021.

[20] NVIDIA, "Isaac gym," *https://developer.nvidia.com/isaac-gym*, 2020.

[21] L. Fan, Y. Zhu, J. Zhu, Z. Liu, O. Zeng, A. Gupta, J. Creus-Costa, S. Savarese, and L. Fei-Fei, "Surreal: Open-source reinforcement learning framework and robot manipulation benchmark," in *Conference on Robot Learning*. PMLR, 2018, pp. 767–782.

[22] I. Zamora, N. G. Lopez, V. M. Vilches, and A. H. Cordero, "Extending the openai gym for robotics: a toolkit for reinforcement learning using ros and gazebo," *arXiv preprint arXiv:1608.05742*, 2016.

[23] J. Lee, M. X. Grey, S. Ha, T. Kunz, S. Jain, Y. Ye, S. S. Srinivasa, M. Stilman, and C. K. Liu, "Dart: Dynamic animation and robotics toolkit," *Journal of Open Source Software*, vol. 3, no. 22, p. 500, 2018.

[24] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning, 2016," *URL http://pybullet. org*, 2016.

[25] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 5026–5033.

[26] W. Huang, X. Huang, C. Majidi, and M. K. Jawed, "Dynamic simulation of articulated soft robots," *Nature communications*, vol. 11, no. 1, pp. 1–9, 2020.

[27] Y. Hu, J. Liu, A. Spielberg, J. B. Tenenbaum, W. T. Freeman, J. Wu, D. Rus, and W. Matusik, "Chainqueen: A real-time differentiable physical simulator for soft robotics," in *2019 International conference on robotics and automation (ICRA)*. IEEE, 2019, pp. 6265–6271.

[28] E. Coevoet, T. Morales-Bieze, F. Largilliere, Z. Zhang, M. Thieffry, M. Sanz-Lopez, B. Carrez, D. Marchal, O. Goury, J. Dequidt, and C. Duriez, "Software toolkit for modeling, simulation, and control of soft robots," *Advanced Robotics*, vol. 31, no. 22, pp. 1208–1224, 2017. [Online]. Available: https://doi.org/10.1080/01691864.2017.1395362

[29] C. B. Teeple, G. R. Kim, M. A. Graule, and R. J. Wood, "An active palm enhances dexterity of soft robotic in-hand manipulation," in *2021 IEEE International Conference on Robotics and Automation (ICRA). IEEE*, 2021.

[30] C. B. Teeple, R. C. S. Louis, M. A. Graule, and R. J. Wood, "The role of digit arrangement in soft robotic in-hand manipulation," in *2021 IEEE International Conference on Intelligent Robots and Systems (IROS), p. In Review, IEEE*, 2021.

[31] M. A. Graule and T. P. McCarthy, "SoMoGym," https://github.com/GrauleM/SoMoGym, 2021.

[32] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[33] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *International Conference on Machine Learning*. PMLR, 2018, pp. 1587–1596.

[34] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

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

[36] A. Raffin, A. Hill, M. Ernestus, A. Gleave, A. Kanervisto, and N. Dormann, "Stable baselines3," *GitHub repository*, 2019.