Nonprehensile Planar Manipulation through Reinforcement Learning with Multimodal Categorical Exploration

Juan Del Aguila Ferrandis

4th Year Project Report
Computer Science and Mathematics
School of Informatics
University of Edinburgh

2023

Abstract

Developing robot controllers capable of achieving dexterous nonprehensile manipulation, such as pushing an object on a table, is challenging. The underactuated and hybrid-dynamics nature of the problem, further complicated by the uncertainty resulting from the frictional interactions, requires sophisticated control behaviors. Reinforcement Learning (RL) is a powerful framework for developing such robot controllers. However, previous RL literature addressing the nonprehensile pushing task achieves low accuracy, non-smooth trajectories, and only simple motions, i.e. without rotation of the manipulated object. We conjecture that previously used unimodal exploration strategies fail to capture the inherent hybrid-dynamics of the task, arising from the different possible contact interaction modes between the robot and the object, such as sticking, sliding, and separation. In this work, we propose a multimodal exploration approach through categorical distributions, which enables us to train planar pushing RL policies for arbitrary initial and target object poses, i.e. different positions and orientations, and with improved accuracy. We show that the learned policies are robust to external disturbances and observation noise, and scale to tasks with multiple pushers. Furthermore, we validate the transferability of the learned policies, trained entirely in simulation, to a physical robot hardware using the KUKA iiwa robot arm. The behavior of the policies in the physical robot can be seen in our supplemental video: https://youtu.be/mOgibE-qv3k.

Research Ethics Approval

This project was planned in accordance with the Informatics Research Ethics policy. It did not involve any aspects that required approval from the Informatics Research Ethics committee.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Juan Del Aguila Ferrandis*)

Acknowledgements

First of all, I would like to thank my supervisor Professor Sethu Vijayakumar for his invaluable help and support throughout the past year, and for welcoming me into the SLMC group, which has provided countless opportunities for academic and personal growth. Additionally, I am deeply grateful to Dr. João Moura for his persistent and clear guidance, and for always being happy to engage in discussions and answer my numerous questions. When I started this project my understanding of robotics and reinforcement learning was quite limited. However, my time within the SLMC group has been incredibly enriching, inspiring me to pursue this project with unwavering determination, which culminated in our submission of a conference paper.

I would also like to thank my parents, José María and Teresa, my brothers, Santiago, José, and Guillermo, and my sister, Teresa. Their unconditional love and support means the world to me and has been a constant source of strength and motivation. Finally, I would like to thank my dear friend Iván, who has accompanied me in many walks and sleepless nights throughout the past four years.

Table of Contents

Chapter 1

Introduction

1.1 Motivation

With the current trend of an aging population, and the ambition to automate dangerous jobs, developing versatile robots that can interact intelligently in diverse environments is becoming ever more important. Recent advancements have greatly enhanced the capabilities of robots to execute complex tasks. However, most industrial and assistive robots are currently limited to pick-and-place tasks. In contrast, humans and animals demonstrate a broader range of contact-rich manipulation skills that extend beyond the scope of prehensile manipulation.

Nonprehensile manipulation is defined as manipulation without grasping [1]. Humans are constantly performing nonprehensile manipulation tasks in their everyday lives, for example, when pushing a box on the floor, rolling or throwing a ball, and balancing a tray. We are interested in studying robot control for nonprehensile manipulation since it endows robots with versatile behaviors, enabling them to perform a wide range of motions on objects with different properties [1]–[3].

In particular, choosing not to grasp the object allows the robot to realize a broader class of object motions that might not be feasible by the end-effector itself [2]. The end-effector is the device attached at the end of a robotic arm, such as a gripper or a pusher. Furthermore, suppose that we wish to develop a robot capable of manipulating a wide range of objects, with different sizes, shapes, and weights. Manipulating these objects exclusively through grasping would require a highly complex and dexterous gripper that can adapt to the characteristics of the objects; however, nonprehensile manipulation allows us to use simpler end-effectors, with the only requirement being to be able to apply forces to the object [2], [3]. Nonprehensile manipulation leverages features of the external environment, such as gravitational and frictional forces, to realize complex motions, and can expand the effective workspace of the robot through throwing motions [3].

Nevertheless, this versatility comes at the expense of increased complexity. Allowing the pose of the object relative to the end-effector to change requires the robot to constantly adapt the contact positions, leading to different possible contact modes in the form

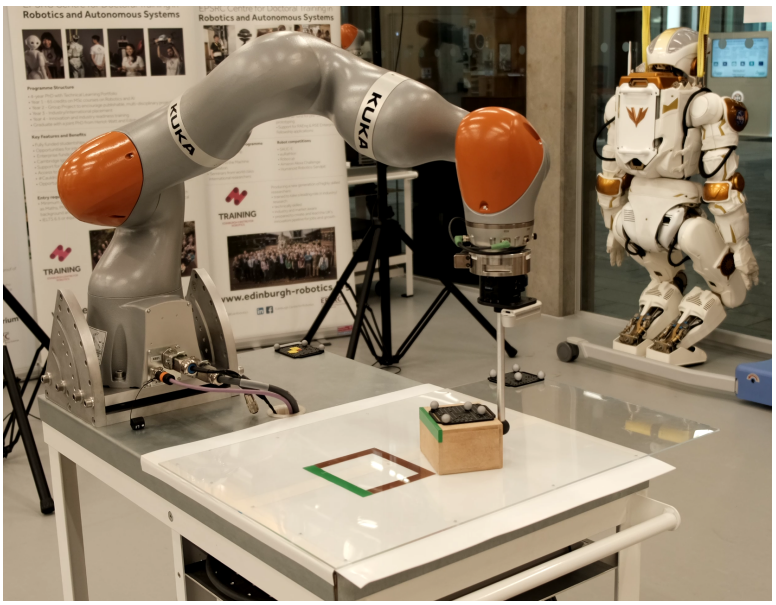


Figure 1.1: Experimental robotic hardware set-up for the planar pushing task. The robot uses a pusher to move an object to a specified target pose.

of *sticking*, *sliding*, and *separation*. *Sticking* occurs when the end-effector and the manipulated object are in contact, but not sliding relative to each other. Additionally, *sliding* occurs when the end-effector and the manipulated object are in contact and sliding relative to each other. Finally, *separation* occurs when the end-effector and the manipulated object are not in contact. We highlight three key challenges that arise in nonprehensile manipulation:

- **Underactuation**. The system is underactuated, which means that the robot is unable to realize arbitrary motions of the object [4]. Therefore, reasoning about the long-term robot behavior required to perform certain tasks with the manipulated object is complex.
- **Hybrid-dynamics**. The transition between different contact modes results in hybrid-dynamics [4]. A hybrid-dynamical system is characterized by exhibiting both continuous-time and discrete-time dynamics [5]. This leads to discontinuities in the dynamics which can be problematic for gradient-based optimizers [6].
- **Frictional Uncertainty**. The frictional interactions between the robot, the manipulated object, and the environment are hard to model [6]–[8]. This exacerbates the uncertainty in the contact modes and the behavior of the object during the interaction with the robot.

In this work we consider the task of planar pushing, widely studied in the nonprehensile literature [1], [4], [8]–[10]. The task, as seen in Figure 1.1, consists of using a robotic pusher to control the motion of an object sliding on a flat surface. In order to address the aforementioned challenges in nonprehensile manipulation, previous works developed robot controllers for planar pushing generally following one of two approaches: model-based via Model Predictive Control (MPC) [4], [10], or model-free via Reinforcement Learning (RL) [11]–[14]. The notion of prediction and feedback is a key characteristic

of both MPC and RL methods, which addresses the challenges of the planar pushing task by anticipating the outcomes of the actions and making adjustments in real-time.

These approaches typically face different open problems. MPC lacks scalability to more complex scenarios, such as multiple contacts and switching contact faces [4], [6], while RL methods tend to produce non-smooth robot motions and, in the case of planar pushing, show limited range of sub-optimal, idiosyncratic motions. Specifically, the current literature on the application of RL methods to planar pushing achieve low accuracy and only simple motions, i.e. without orientation of the manipulated object [11]–[14], which we aim to consider.

1.2 Goals and Contributions

The primary goal of this project was to investigate the application of RL methods to the task of planar pushing. In particular, we were interested in using RL to develop a robot controller that can accurately push an object to an arbitrary target pose (position and orientation). Lastly, we aimed to validate the developed controller on a physical planar pushing robotic hardware.

As discussed in the previous section, the current RL literature addressing the planar pushing task achieves simple motions that are only capable of pushing an object to a target position, disregarding the orientation of the object [11]–[14]. These RL methods share a common trait: they perform exploration using a multivariate Gaussian distribution, with diagonal covariance, defined in the action space (Section 3.2.1). This limits the exploration to unimodal policies across each action space dimension. However, the model-based literature identifies the planar pushing problem as a hybrid-dynamic system due to the different possible contact modes (sticking, sliding left, sliding right, and separation). This provides us with the insight that perhaps planar pushing is fundamentally a multimodal control problem. Therefore, we ask the question:

Can multimodal exploration enable us to learn robust, scalable, and accurate planar pushing RL policies that incorporate object orientation?

In this work, we address this question and make the following contributions:

- We propose a multimodal exploration approach, with categorical distributions on a discrete action space, which enables us to learn planar pushing RL policies for arbitrary initial and target object poses, i.e. different positions and orientations.
- We demonstrate that the proposed framework is robust to disturbances and observation noise, scalable to two pushers, and exhibits smooth pushing motions.
- We validate the policies, trained only in simulation, on a physical hardware set-up using the KUKA iiwa robot.

We submitted these contributions for publication in the *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, one of the major conferences for robotics research. The following is our submitted paper, which is currently under review:

[15] J. Del Aguila Ferrandis, J. Moura, and S. Vijayakumar, "Nonprehensile planar manipulation through reinforcement learning with multimodal categorical exploration," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, [Under review], 2023

1.3 Report Structure

Chapter 2 presents the previous literature on dynamics modelling, MPC, and RL for planar pushing, and discusses their limitations and open problems. Furthermore, it defines the planar pushing task for arbitrary initial and target object poses, and provides the background on the RL algorithm used within the proposed multimodal categorical exploration framework. Chapter 3 describes how the RL task is constructed in order to learn planar pushing policies, as well as the techniques used to achieve accurate policies with good transferability to the physical hardware. It also presents the Gaussian exploration approach used in previous literature, and our proposed categorical exploration approach. Chapter 4 details the procedure to train the policies in simulation and deploy them on the physical robot. Chapter 5 describes the design of the experiments conducted, presents the results, and discusses the findings. Finally, Chapter 6 summarizes the contributions, highlights the key insights from this work, discusses the limitations, and outlines possible avenues for future work.

Chapter 2

Background

2.1 Literature Review

Since the work of Mason [1], planar pushing has generally been considered one of the fundamental problems to study nonprehensile manipulation. We conduct a literature review of the previous works in dynamics modelling and robot control for planar pushing. In general, robot control systems can be model-based or model-free. Model-based controllers rely on a mathematical model of the system in which they act, which may include models of the environment, the robot, contact interactions, and more. On the other hand, model-free controllers do not require such models of the system and instead generally make use of data-driven methods to learn the desired behaviors. Previous works have explored both model-based and model-free controllers for planar pushing. The state-of-the-art model-based controllers formulate the task using MPC, while the state-of-the-art model-free controllers utilize RL. We now discuss the current literature on dynamics modelling, MPC and RL for planar pushing.

2.1.1 Dynamics Modelling

Various works have developed analytical models of the dynamics of planar pushing. To begin with, Mason [1] proposed the voting theorem, which models the direction of rotation of an object experiencing a pushing force at a particular contact point, without requiring information about pressure distribution between the object and the support surface. Later on, Goyal *et al.* [9] introduced the concept of the limit surface, which provides a geometrically interpretable model of the interaction between the frictional forces and the motion of a flat object sliding on a planar surface.

Data-driven models of the dynamics of planar pushing have also been developed. Zhou *et al.* [16] proposed a physics-informed data-driven model of the surface frictional forces as well as the object velocities in planar pushing scenarios by optimising a convex polynomial representation. Additionally, Bauza and Rodriguez [8] used Gaussian processes to model the expected position and orientation, and their corresponding variance, of an object after applying a particular pushing motion. These data-driven models are limited to the specific objects and surface material that were used for model

learning. Additionally, they assume continuous contact between the pusher and object.

The previously discussed works on dynamics modelling, with the exception of the work of Bauza and Rodriguez [8], have a key limitation since they rely on the quasi-static assumption. The quasi-static assumption involves a regime where, at low pushing velocities, frictional forces are balanced and inertial effects are negligible [4], [17]. Therefore, these dynamics models are unsuitable for pushing tasks requiring nimble interactions between the robot end-effector and the object due to the increased sliding velocity. The quasi-static assumption is also unsuitable in scenarios where the robot exerts a throwing motion on the object since inertial effects become non-negligible. Bauza and Rodriguez [8] managed to side step the quasi-static assumption by providing the pusher velocity as a model input. A key conclusion of their work is that the quasi-static assumption does not hold with pushing velocities greater than $0.08\,\mathrm{m\,s^{-1}}$.

2.1.2 Model Predictive Control

Model Predictive Control (MPC) is a control technique that uses a dynamics model to predict the future behavior of the system, over a finite time horizon, in order to generate a sequence of actions that optimizes this forecast [18]. The first action in the sequence is applied and then the process is repeated for the next control step using the updated system state information [18]. This receding horizon strategy enables MPC to account for the longer-term effect of the actions throughout the prediction horizon, as well as react to changes in the environment and inaccuracies in the dynamics model. Overall, MPC is a widely adopted technique in robotics since it leverages knowledge about the dynamics and constraints of the system to produce sophisticated control behaviors [19].

Hogan and Rodriguez [4] proposed a mixed integer MPC formulation for offline tracking of nominal trajectories in planar pushing. They used a mixed integer quadratic programming approach within their MPC to formulate the decision between sticking and sliding contact modes. However, this MPC formulation is too computationally expensive to be used for online tracking of nominal trajectories. In the same work, Hogan and Rodriguez [4] used a deep neural network to learn an approximation of the contact mode transitions within the MPC prediction horizon. The deep neural network was trained using optimal solutions computed offline with their mixed integer formulation. This learned approximation of the contact mode selection enabled an online deployment of the MPC. Nevertheless, only the sticking and sliding contact modes are considered, which is appropriate when the pusher remains in contact with the object throughout the interaction; however, scaling this approach to more complex scenarios, for example involving switching contact faces or controlling multiple pushers, could be problematic.

Recently, Moura *et al.* [10] proposed a complementarity constraint MPC formulation for offline planning and online tracking of planar pushing nominal trajectories. They used a Mathematical Program with Complementarity Constraints (MPCC) formulation within a trajectory optimisation framework to plan trajectories with sticking and sliding contact modes. This framework was used within a closed-loop MPC to achieve online tracking of the planned nominal trajectories, i.e. without pre-learning the contact mode selection as in [4]. A limitation of this approach is that the MPCC assumes that it is feasible to transition between contact modes instantaneously. However, this is not

always the case, for example, when there is a large separation between the pusher and the object and we want the pusher to make contact.

The MPC formulations proposed by Hogan and Rodriguez [4] and Moura *et al.* [10] are capable of tracking nominal trajectories and achieving target object poses with significant accuracy, while recovering from moderate disturbances. Nevertheless, they only incorporate the sticking and sliding contact modes within a fixed object face, and therefore are unable to switch contact faces during the interaction, which limits the range of feasible object motions. Additionally, both methods rely on offline planning and online tracking of nominal trajectories. This involves computing a valid trajectory, and then using an MPC with a short time horizon to follow this trajectory. The trajectory planning stage cannot be solved online due to its computational complexity. When a disturbance is applied to the pushed object, the short time horizon MPC is used to regain control of the object and return to the nominal trajectory. However, with large disturbances, the MPC might fail to converge due to its short time horizon, leading to failure to regain control of the object. In such cases, it would be necessary to re-plan the trajectory offline. Finally, both MPC formulations rely on dynamics models that make the quasi-static assumption. This suggests that their performance would significantly degrade with higher pushing velocities. In fact, as previously discussed, the quasi-static assumption does not hold with pushing velocities greater than $0.08\,\mathrm{m\,s^{-1}}$ [8].

2.1.3 Reinforcement Learning

Reinforcement Learning (RL) is a branch of machine learning that focuses on enabling agents to learn through interactions with the environment in order to take actions that maximize a cumulative reward signal [20]. The agent discovers the actions that lead to greater rewards through trial and error, which requires a careful balance between exploitation of previously effective strategies and exploration of new strategies, aiming to improve the current behavior. This is known as the exploration-exploitation trade-off. RL systems generally rely on the notion of value functions, which attempt to capture how good is a particular environment state, or state-action pair, in the long term [20]. Actions can have long-term effects on future rewards, so one of the key challenges in RL is determining which actions were responsible for the eventual success or failure. Finally, the aim is to learn a policy function that maps the current state of the agent and the environment to an optimal action.

Throughout recent years, RL has achieved impressive results in various robot control tasks such as manipulation [21] and locomotion [22], [23]. In particular, RL has proven to be a powerful framework to solve complex tasks in simulation due to the availability of large sample sizes of diverse scenarios. However, transferring the learned policies to a physical robot can be challenging. There have been significant efforts to reduce the sim-to-real gap and one of the most successful techniques, which we use in this work, is called dynamics randomization [11]. It consists on randomizing the physical properties that might differ between the simulation and real world environments. This leads to learning more robust RL policies that can cope with the dynamics of unfamiliar environments [11].

Peng *et al.* [11] used dynamics randomization to transfer planar pushing RL policies

learned in simulation to a physical robot. They implemented a Recurrent Deep Deterministic Policy Gradient (RDDPG) [24] algorithm, and randomized various properties of the simulation, such as friction coefficients and the mass of different components. The learned policy demonstrates complex behaviors such as making and breaking contact with the manipulated object to make adjustments and reach a target position. Nevertheless, they define episode success as pushing the object within 7 cm of a target position. Hence, the policy achieves significantly low accuracy and disregards the orientation of the manipulated object. Finally, the policy exhibits non-smooth motions, presumably due to the learning taking place in the configuration space, which means that the policy actions correspond to target joint angles of the robot. This could lead to non-smooth and unstable behavior due to the high dimensionality of the output, as well as the complexity involved in coordinating multiple joints to achieve a particular object motion. In our work, we avoid this issue by learning the RL policies in the task space, meaning that the policy actions correspond to target motions of the robot end-effector. We then use an Inverse Kinematics (IK) solver to map policy actions to robot joint configurations.

Later on, Jeong *et al.* [12] learned a state dependent Generalized Force Model (GFM) to capture the difference between the real-world and simulation environments, and used it to transfer planar pushing RL policies learned in simulation to a physical robot. They trained the RL policies in the configuration space using Maximum a Posteriori Policy Optimisation (MPO) [25]. Furthermore, the policies are trained first in simulation, then deployed on the physical robot to collect data, next the GFM is learned with this data, and finally the policies are re-trained using the GFM. However, the policies exhibit low accuracy and only simple motions. In particular, an episode is defined as successful when the object is within 2.5 cm of the target position, again disregarding object orientation. Additionally, the policies are only trained for a fixed starting position, and ten fixed possible target positions. Finally, Jeong *et al.* [12] attempted to extend their approach to incorporate object orientation. They restricted the task to a fixed starting pose and ten fixed possible target poses, and considered an episode successful when the object is within 5 cm and 20 deg of the target pose. Even in this simplified scenario, the best policy only completes the task on 44% of its attempts.

Recently, Cong *et al.* [13] used RL with a vision-proprioception model to learn planar pushing policies for objects with different shapes. They extract information about the object's shape, pose, and target position from raw images by segmenting an object mask and extracting a latent representation through a Variational Autoencoder (VAE). Furthermore, they combine this latent representation with the robot's proprioception and use Soft Actor Critic (SAC) [26] to learn control policies in the task space. However, this approach disregards the orientation of the object and exhibits poor accuracy, with an average distance between the target position and the final object position of around 4 cm. Finally, when completing the same tasks in simulation and in the real-world, the policy takes on average four times longer in the real-world, indicating that the policy generalizes poorly to unfamiliar dynamics.

These model-free RL methods manage to overcome some of the online scalability limitations of model-based MPC through extensive offline exploration during training. In particular, they can switch contact faces and generally recover from significant

external disturbances. There are challenges in scaling model-based MPC to handle these more complex scenarios due to the increased computational complexity, as well as the difficulty in modelling them. Furthermore, the model-free RL methods do not rely on the quasi-static assumption so they can use higher pushing velocities.

Nevertheless, the RL methods exhibit low accuracy. They reach the target position with errors around 2.5 cm [12] or greater, while MPC achieves errors around 0.5 cm [10]. Additionally, MPC methods perform longer pushing motions, demonstrating skillful control of the object, and resulting in more efficient trajectories. On the other hand, RL methods rely on short pushes to approximate the object to the target position over multiple attempts. Most importantly, previous RL methods addressing the planar pushing task are unable to handle arbitrary initial and target object poses, i.e. different positions and orientations [11]–[14]. These methods perform exploration using a multivariate Gaussian with diagonal covariance, which is unimodal across each action space dimension (Section 3.2.1). In this work, we propose a multimodal exploration approach through categorical distributions on a discrete action space. This approach enables us to train planar pushing RL policies for arbitrary initial and target object poses across the entire workspace, with significantly improved accuracy.

2.2 The Planar Pushing Task

We consider the task of pushing a box to a specified target pose, composed of the box position and orientation, from a random initial system configuration, composed of the initial box pose and robot pusher position, all within a bounded planar workspace. Figure 2.1 illustrates the planar pushing system, where $(v_{x,p}, v_{y,p})$ is the current velocity of the robot pusher, located at (x_p, y_p). Additionally, (x_b, y_b, θ_b) is the current pose of the box, and $(x_{\text{targ}}, y_{\text{targ}}, \theta_{\text{targ}})$ is the target box pose.

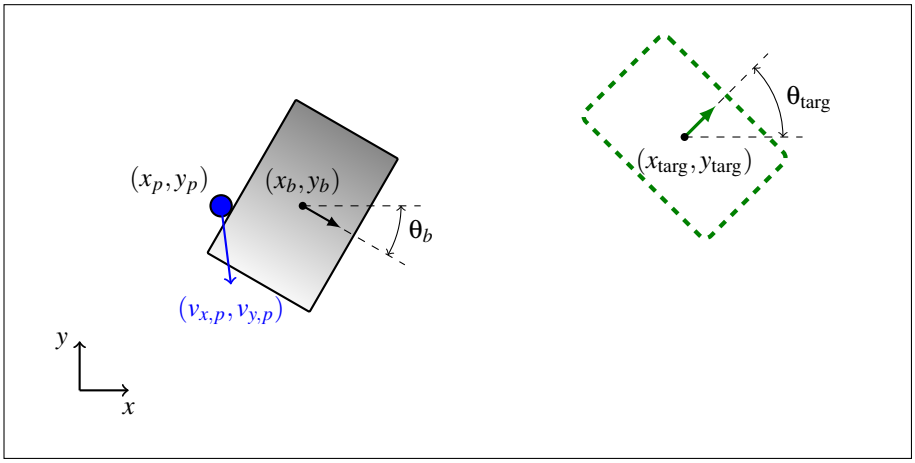


Figure 2.1: Illustration of the planar pushing system.

2.3 Formulation of the Reinforcement Learning Problem

We consider a finite horizon goal-conditioned Partially Observable Markov Decision Process (POMDP) defined by the tuple $(\mathcal{S}, \Omega, \mathcal{G}, \mathcal{A}, O, \mathcal{P}, \mathcal{R}, H, \rho_0, \rho_g)$ [13], [23]. At each time step t, the environment has state $\boldsymbol{s}_t \in \mathcal{S}$, we receive an observation $\boldsymbol{o}_t \in \Omega$, our goal is $\boldsymbol{g}_t \in \mathcal{G}$, and we take an action $\boldsymbol{a}_t \in \mathcal{A}$. Note that the goal remains fixed during an episode. Additionally, $O : \mathcal{S} \times \mathcal{A} \to \Pr(\Omega)$ is the observation model, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \Pr(\mathcal{S})$ is the transition dynamics, and $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{G} \to \mathbb{R}$ is the reward function. We limit episodes to have a maximum horizon H. Finally, the initial state and goal of an episode are distributed according to ρ_0 and ρ_g respectively.

We consider a POMDP because the observation $\boldsymbol{o}_t$ of the policy does not capture all the information in the environment state $\boldsymbol{s}_t$. For example, external forces as well as frictional forces between the robot end-effector, the box, and the planar surface are not observable.

We wish to learn a stochastic policy $\pi_{\boldsymbol{\phi}} : \Omega \times \mathcal{G} \to \Pr(\mathcal{A})$, parametrized by $\boldsymbol{\phi}$, that maximizes the expected cumulative reward, given by

$$\mathbb{E}_{\pi_{\boldsymbol{\phi}}} \left[\sum_{t=0}^{H-1} \gamma^t r_t \right], \tag{2.1}$$

where r_t is the reward at time step t, and $\gamma \in [0, 1]$ is a discount factor.

2.4 Proximal Policy Optimisation

In order to learn the policy $\pi_{\boldsymbol{\phi}}$, we use Proximal Policy Optimisation (PPO), a popular model-free on-policy RL algorithm proposed by Schulman *et al.* [27]. We use PPO since it was designed to be used both for continuous action spaces and discrete action spaces [27]. This is important for our investigation since we compare: (a) PPO with multimodal exploration through categorical distributions on a discrete action space, our proposed approach, against (b) PPO with unimodal exploration through a multivariate Gaussian with diagonal covariance on a continuous action space, the previously used exploration approach. Additionally, there are various reliable implementations of PPO available [28]–[30], and PPO has already been successfully applied to various control tasks, including in-hand manipulation [21] and locomotion [23], [31].

PPO is an on-policy algorithm because it updates the policy during training using data collected from the current policy [20]. It belongs to a class of RL methods called Policy Gradient Methods. Let $f(\boldsymbol{\phi})$ be an objective function that measures the performance of the policy. Policy Gradient Methods attempt to maximize $f(\boldsymbol{\phi})$ through gradient ascent, making policy updates of the form

$$\boldsymbol{\phi}_{i+1} = \boldsymbol{\phi}_i + \eta \widehat{\nabla(f(\boldsymbol{\phi}_i))}, \tag{2.2}$$

where i denotes the gradient ascent iteration, η is the learning rate, and $\widehat{\nabla(f(\boldsymbol{\phi}))}$ is an estimate of the gradient of $f(\boldsymbol{\phi})$ [20]. In practice, a more complex gradient ascent formulation, such as Adam [32], is usually used.

PPO uses an actor-critic architecture. The actor corresponds to the policy function $\pi_{\boldsymbol{\phi}}$, while the critic corresponds to an estimate of the state value function, which measures how good is a particular environment state [20], [27]. Since we consider a goal-conditioned POMDP, we define the estimated state value function as $V_{\boldsymbol{\psi}} : \Omega \times \mathcal{G} \to \mathbb{R}$, where $\boldsymbol{\psi}$ are the parameters of the function. PPO learns $\boldsymbol{\phi}$ and $\boldsymbol{\psi}$ through gradient ascent, so $\pi_{\boldsymbol{\phi}}$ and $V_{\boldsymbol{\psi}}$ must be differentiable. As a result, $\pi_{\boldsymbol{\phi}}$ and $V_{\boldsymbol{\psi}}$ are usually implemented using deep neural networks due to their differentiability and large learning capacity.

Additionally, PPO relies on the notion of an advantage function, which measures the additional performance with respect to the current policy that is obtained by taking a particular action [27], [33]. PPO uses a truncated form of Generalized Advantage Estimation (GAE) [33] to estimate the advantage function for time step $t \in [0, T]$, which is given by

$$\hat{A}_t = \sum_{i=0}^{T-t} (\gamma\lambda)^i \delta_{t+i}, \tag{2.3}$$

with

$$\delta_t = r_t + \gamma V_{\boldsymbol{\psi}}(\boldsymbol{o}_{t+1}, \boldsymbol{g}_{t+1}) - V_{\boldsymbol{\psi}}(\boldsymbol{o}_t, \boldsymbol{g}_t), \tag{2.4}$$

where λ is the GAE parameter [27], [33]. The γ and λ parameters control the bias-variance tradeoff in the advantage estimation [33].

We now turn to the objective function. PPO seeks to maximize the following objective:

$$L_t(\boldsymbol{\phi}, \boldsymbol{\psi}) = \hat{\mathbb{E}}_t \left[L_t^{\pi_{\boldsymbol{\phi}}} + c_1 L_t^{V_{\boldsymbol{\psi}}} + c_2 S_t^{\pi_{\boldsymbol{\phi}}} \right], \tag{2.5}$$

where the expectation is computed as the empirical average over a mini-batch of time steps t, $L_t^{\pi_{\boldsymbol{\phi}}}$ is the objective of the policy function, $L_t^{V_{\boldsymbol{\psi}}}$ is the objective of the state value function, $S_t^{\pi_{\boldsymbol{\phi}}}$ is an entropy bonus, and c_1, c_2 are weights [27].

The objective of the policy function is given by

$$L_t^{\pi_{\boldsymbol{\phi}}} = \min \left(\frac{\pi_{\boldsymbol{\phi}}(\boldsymbol{a}_t \mid \boldsymbol{o}_t, \boldsymbol{g}_t)}{\pi_{\boldsymbol{\phi}_{old}}(\boldsymbol{a}_t \mid \boldsymbol{o}_t, \boldsymbol{g}_t)} \cdot \hat{A}_t,\ \text{clip} \left(\frac{\pi_{\boldsymbol{\phi}}(\boldsymbol{a}_t \mid \boldsymbol{o}_t, \boldsymbol{g}_t)}{\pi_{\boldsymbol{\phi}_{old}}(\boldsymbol{a}_t \mid \boldsymbol{o}_t, \boldsymbol{g}_t)},\ 1-\varepsilon,\ 1+\varepsilon \right) \hat{A}_t \right), \tag{2.6}$$

where ε controls the clip range, and $\boldsymbol{\phi}_{old}$ are the policy parameters before the policy update [27]. The intuition is that, for a given observation $\boldsymbol{o}_t$ and goal $\boldsymbol{g}_t$, if the action $\boldsymbol{a}_t$ leads to a positive advantage, then we want the probability of the policy taking this action, given by $\pi_{\boldsymbol{\phi}}(\boldsymbol{a}_t \mid \boldsymbol{o}_t, \boldsymbol{g}_t)$, to increase. Indeed, if $\hat{A}_t > 0$, then the objective $L_t^{\pi_{\boldsymbol{\phi}}}$ increases if $\pi_{\boldsymbol{\phi}}(\boldsymbol{a}_t \mid \boldsymbol{o}_t, \boldsymbol{g}_t)$ increases. The purpose of clipping the probability ratio to the interval $[1-\varepsilon, 1+\varepsilon]$, and then taking the minimum, is to restrict how much the objective can increase, thereby preventing excessively large updates to the policy that could be detrimental. In the case that the action leads to a negative advantage, then we want the probability of the policy taking this action to decrease. Accordingly, if $\hat{A}_t < 0$, then the objective $L_t^{\pi_{\boldsymbol{\phi}}}$ increases if $\pi_{\boldsymbol{\phi}}(\boldsymbol{a}_t \mid \boldsymbol{o}_t, \boldsymbol{g}_t)$ decreases.

The objective of the state value function is given by

$$L_t^{V_{\boldsymbol{\psi}}} = -(V_{\boldsymbol{\psi}}(\boldsymbol{o}_t, \boldsymbol{g}_t) - \hat{V}_{\text{targ}}(\boldsymbol{o}_t, \boldsymbol{g}_t))^2, \tag{2.7}$$

where $\hat{V}_{\text{targ}}$ is an estimate of the true state value function [27]. Note that we aim to minimize the squared difference between the current state value function and the estimated true state value function, but we perform gradient ascent on the overall objective $L_t(\boldsymbol{\phi}, \boldsymbol{\psi})$, which is why $L_t^{V_{\boldsymbol{\psi}}}$ is explicitly formulated as a negative objective. There are various techniques to compute the estimate of the true state value function, but a common choice [28], [34] is

$$\hat{V}_{\text{targ}}(\boldsymbol{o}_t, \boldsymbol{g}_t) = \hat{A}_t + V_{\boldsymbol{\psi}}(\boldsymbol{o}_t, \boldsymbol{g}_t), \tag{2.8}$$

such that

$$L_t^{V_{\boldsymbol{\psi}}} = -\hat{A}_t^2. \tag{2.9}$$

The definition of the true advantage function is $A(\boldsymbol{s}, \boldsymbol{a}) = Q(\boldsymbol{s}, \boldsymbol{a}) - V(\boldsymbol{s})$, where $Q(\boldsymbol{s}, \boldsymbol{a})$ is the true state-action value function, and $V(\boldsymbol{s})$ is the true state value function [33]. Note that PPO does not explicitly use a state-action value function. Therefore, the intuition behind Equation (2.9) is that, if the expected value of the squared advantage increases, this is an indicator that the accuracy of our estimated state value function $V_{\boldsymbol{\psi}}$ has decreased.

Lastly, the entropy bonus $S_t^{\pi_{\boldsymbol{\phi}}}$ can be used to encourage a stochastic policy function with a higher entropy [27], which leads to a greater degree of exploration of new behaviors, and a lower degree of exploitation of known information. The calculation of the entropy depends on the formulation of the stochastic policy. In practice, it is common for PPO implementations to set $c_2 = 0$, such that the entropy bonus is disregarded, since sufficient exploration is provided by the stochastic actions of the policy [27]–[29]. We also disregard the entropy bonus in our experiments.

Chapter 3

Method

3.1 Definition of the Learning Task

Beyond the algorithm used to train the policy, RL tasks have four primary components: the observation of the environment received by the policy, the model architectures of the policy and value functions (or any functions required by the learning algorithm), the set of possible policy actions, and finally the reward function. In this section, we define each of these components for the planar pushing task studied.

In our case, since we consider a goal-conditioned setting, at each time step t the policy receives an observation of the environment $\boldsymbol{o}_t$, as well as the goal of the episode $\boldsymbol{g}_t$. The observation of the environment consists of the current box pose (x_b, y_b, θ_b) and the current pusher position (x_p, y_p). Additionally, the goal of the episode consists of the target box pose $(x_{\text{targ}}, y_{\text{targ}}, \theta_{\text{targ}})$.

As discussed in Section 2.3, we formulate the problem as a POMDP because there is important information from the environment state $\boldsymbol{s}_t$ that the policy observation $\boldsymbol{o}_t$ fails to capture, such as the frictional contact forces and the velocity of the box. We consider two model architectures that attempt to enable the policy and state value functions in PPO to capture this hidden information. Both architectures have been widely used in the RL literature with multiple tasks and learning algorithms. The first architecture consists of a Multilayer Perceptron (MLP) [35] which receives as an input a stack of previous observations $\{\boldsymbol{o}_t, \boldsymbol{o}_{t-1}, \dots, \boldsymbol{o}_{t-l}\}$ in addition to the goal $\boldsymbol{g}_t$ [11], [36]. Since MLPs have no memory, providing a stack of previous observations allows the MLP to observe the evolution of the pusher position and box pose over a fixed time period, and hence learn to extract feature representations that are predictive of the hidden dynamics of the environment. The second architecture consists of a Long Short-Term Memory (LSTM) model [37], which receives as an input the current observation $\boldsymbol{o}_t$ and goal $\boldsymbol{g}_t$, as well as the hidden state vector of the model [11], [21]. The LSTM model can update its hidden state during policy interaction with the environment such that it is predictive of the dynamics of the environment.

In general, the MLP architecture is simpler and less computationally expensive at training time and inference time than the LSTM architecture. However, the MLP

architecture is limited since it can only access a fixed amount of previous information at any given time step. On the other hand, the LSTM architecture can capture long-term dependencies, learning what information to store and what information to forget during policy interaction with the environment. Therefore, it may be able to capture more useful information about the hidden dynamics of the environment. Previous work [11] has found the LSTM architecture to provide faster convergence and better sim-to-real transfer of the learned policies.

Given an observation $\boldsymbol{o}_t$ (or stack of observations) and a goal $\boldsymbol{g}_t$, the policy function outputs the parameters of a probability distribution over the set of possible actions. During training, the action $\boldsymbol{a}_t$ is sampled from this probability distribution, which enables the policy to explore different behaviors. The formulation of the probability distribution shapes the exploration strategy of the policy, which is central to our work and will be discussed in Section 3.2. We learn the RL policies in the task space and therefore the policy action is $\boldsymbol{a}_t = (v_{x,p}, v_{y,p})$, which consists of the x and y velocity of the pusher. We limit the velocity on each axis to the range $[-0.1, 0.1]\ \mathrm{m\,s^{-1}}$.

The reward at time step t is calculated as follows:

$$r_t = \begin{cases} \alpha & \text{successful episode,} \\ -\beta & \text{unsuccessful episode,} \\ k_1(1-d_{x,y}) + k_2(1-d_\theta) + k_3(1-v_p) & \text{otherwise.} \end{cases} \tag{3.1}$$

In particular, if the box reaches the target pose, then the episode terminates successfully with a large positive reward $r_t = \alpha$. Alternatively, if the object fails to reach the target within the maximum horizon, or the workspace boundaries are violated by the pusher or box, then the episode terminates unsuccessfully with a large negative reward $r_t = -\beta$. Otherwise, while the episode has not finished and the policy is working on the task, the reward is given by $r_t = k_1(1-d_{x,y}) + k_2(1-d_\theta) + k_3(1-v_p)$, where $d_{x,y}$ is the normalized distance to the target position, d_θ is the normalized angular distance to the target orientation, v_p is the normalized magnitude of the pusher velocity, and k_1, k_2, k_3 control the weights of the three terms. The normalization calculations for $d_{x,y}$, d_θ, and v_p are performed using min-max scaling to map the original values to the range $[0, 1]$. The first two terms, corresponding to k_1 and k_2, provide signals reflecting the desirability of the current box pose relative to the target pose. The last term, corresponding to k_3, acts as a regularizer designed to encourage efficient motions of the pusher.

We use a combination of sparse and dense signals in the reward function. A sparse signal is given when the episode terminates, while a dense signal is given throughout the episode before it terminates. A purely sparse reward function can provide signals that accurately indicate policy success or failure; however, the sparsity of the rewards can create a challenging learning environment for the policy [11]. On the other hand, a purely dense reward function can help guide the policy towards the goal; however, designing the function can be challenging and it can lead the policy to learn suboptimal behaviors [11]. Therefore, we use a combination of both, with significantly larger sparse reward signals at episode termination to provide the ground truth of episode success and failure, and a smaller dense reward signal throughout the episode to improve learning efficiency by guiding the policy towards the episode goal.

3.2 Exploration Strategies

3.2.1 Gaussian Exploration

As discussed in the Literature Review, previous works on RL for planar pushing perform exploration using a multivariate Gaussian with diagonal covariance. The formulation of this exploration strategy varies across different RL algorithms. Peng *et al.* [11] use RDDPG to learn deterministic planar pushing policies and, during training, add Gaussian exploration noise with mean zero and fixed standard deviation to each dimension of the policy actions. Jeong *et al.* [12] use MPO to learn stochastic policies where the policy function outputs the values of the mean vector $\boldsymbol{\mu}$ and positive diagonal elements of a Cholesky factor $\boldsymbol{F}$, which together define a multivariate Gaussian $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ with diagonal and positive definite covariance $\boldsymbol{\Sigma} = \boldsymbol{F}\boldsymbol{F}^T$. Exploration is therefore performed by sampling actions from the multivariate Gaussian. Lastly, Cong *et al.* [13] use SAC to learn stochastic planar pushing policies which output the values of the mean $\boldsymbol{\mu}$ and diagonal covariance $\boldsymbol{\Sigma}$ that define the multivariate Gaussian $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ from which the actions are sampled.

In this work, we use PPO, which can also be formulated to perform exploration through a multivariate Gaussian with diagonal covariance. In this case, PPO learns a stochastic policy that outputs the mean vector $\boldsymbol{\mu}$. The diagonal entries of the covariance matrix $\boldsymbol{\Sigma}$ are learnable parameters, but they are not determined by the policy function, which means that they are state-independent [27]. Since we learn the policy in the task space, we have

$$\boldsymbol{\mu} = (\mu_x, \mu_y), \qquad \boldsymbol{\Sigma} = \begin{pmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{pmatrix}, \tag{3.2}$$

where μ_x, μ_y are the mean velocities in x and y, and σ_x^2, σ_y^2 are their corresponding variances. Hence, with this formulation, PPO performs exploration during training by sampling $\boldsymbol{a}_t \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, where $\boldsymbol{\mu}$ is determined by the policy function based on the observation $\boldsymbol{o}_t$ and the goal $\boldsymbol{g}_t$, and $\boldsymbol{\Sigma}$ contains the current learned variances. Figure 3.1 illustrates this procedure.

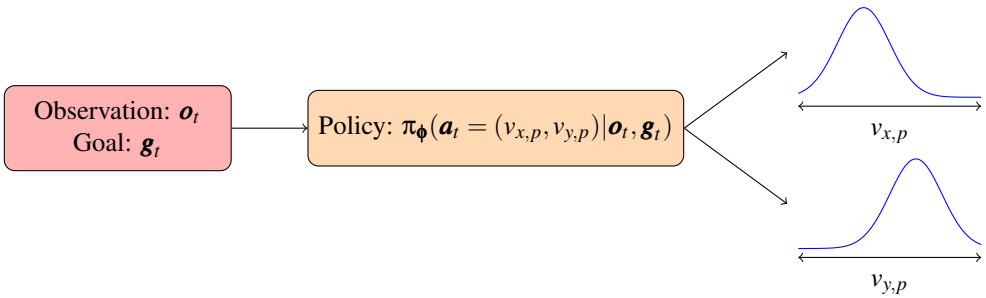


Figure 3.1: Exploration through a multivariate Gaussian with diagonal covariance.

We also conduct experiments with Gaussian exploration using SAC as this provides a valuable baseline for comparison with previous works applying RL to the planar pushing task. SAC is an off-policy algorithm with a maximum entropy objective, and therefore attempts to maximize the cumulative reward and the entropy of the policy

simultaneously [26]. Note that SAC was designed for continuous control tasks and hence uses Gaussian exploration as this is the standard exploration approach in the RL literature for continuous action spaces. As previously discussed, Cong *et al.* [13] recently used SAC for planar pushing. Additionally, the other aforementioned works on RL for planar pushing utilize off-policy algorithms with Gaussian exploration [11], [12], which are key characteristics of SAC. Therefore, SAC serves as an informative baseline for comparison with these works. Finally, in task that we consider, SAC learns a stochastic policy that outputs μ_x, μ_y, σ_x^2, and σ_y^2, which define the multivariate Gaussian used for exploration. Note that, in contrast to PPO, SAC uses a diagonal covariance matrix that is state-dependent [26].

The multivariate Gaussian distribution has domain $(-\infty, \infty)$ in each dimension; however, as previously discussed, we restrict the policy action such that $v_{x,p}, v_{y,p} \in [-0.1, 0.1]$ $\mathrm{m\,s^{-1}}$. PPO implementations generally constrain the action space by clipping the sampled action to the desired interval [28]. On the other hand, SAC implementations generally constrain the action space by applying a squashing function such as `tanh` to the sampled action, which maps the action to the interval $(-1, 1)$, and then re-scaling to the desired interval [13], [28]. We use the same approaches in our experiments with PPO and SAC.

3.2.2 Categorical Exploration

We now discuss our proposed approach for multimodal exploration. A straightforward way to formulate a stochastic policy capable of multimodal exploration is to discretize the action space and construct the policy function such that it outputs the parameters of a categorical distribution for each action space dimension. In our case, we discretize $v_{x,p}$ and $v_{y,p}$ using 11 equally sized bins for each velocity. During training, given an observation $\boldsymbol{o}_t$ and the goal $\boldsymbol{g}_t$, the policy function outputs 11 logits that define a categorical distribution over $v_{x,p}$, and 11 logits that define a categorical distribution over $v_{y,p}$. We then sample the action $\boldsymbol{a}_t = (v_{x,p}, v_{y,p})$ from these distributions. Figure 3.2 illustrates this procedure.

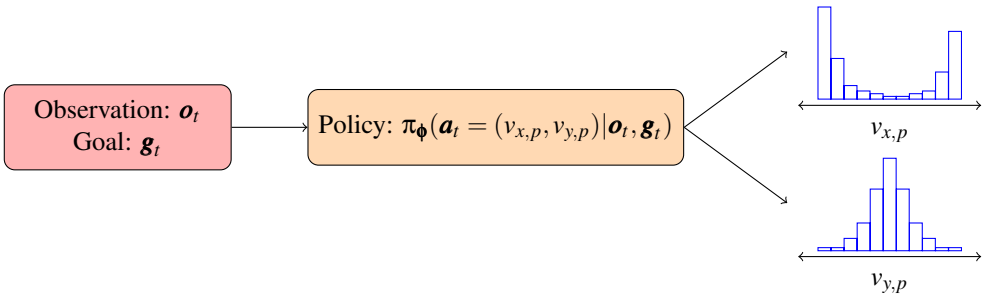


Figure 3.2: Multimodal exploration through categorical distributions.

Increasing the number of bins allows the policy to learn more precise behaviors; however, the learning task becomes more complex since the number of learnable parameters increases. In our case, it is important to use an odd number of equally sized bins in order to allow the possibility of zero velocity. Additionally, we use 11 bins for each action

space dimension since this has been found to provide a good compromise between precision and learning complexity in previous works that discretize continuous action spaces for RL [21], [38].

Finally, when training finishes and the policies are deployed, there is no need for stochastic exploration. As a result, we do not sample the actions from the policy-defined distributions. Instead, if the policy function outputs the parameters of a multivariate Gaussian, we use the mean vector as the action. On the other hand, if the policy function outputs the logits of categorical distributions, we use the mode of the categorical distribution for each action space dimension to define the action. This is a common practice in RL implementations [28], [29].

3.3 Sim-to-Real Transfer

We train the policies entirely in simulation and use dynamics randomization, observation noise, and synthetic disturbances to bridge the sim-to-real gap. At the start of every episode during policy training, we sample random values for:

- The coefficients of friction and restitution of the floor, box, and pusher.
- The dimensions of the box and the pusher.
- The mass of the box.
- The time duration of the action.

Note that the coefficient of restitution describes the behavior of two bodies that collide. In particular, it is calculated as the magnitude of the relative velocity between the two bodies before the collision, divided by the magnitude of the relative velocity after the collision. Additionally, the time duration of the action refers to the amount of time during which a particular action selected by the policy is executed in the environment before the policy can select a different action. Dynamics randomization enables us to learn more robust policies that can handle environments with a wide range of dynamics, therefore improving the sim-to-real transfer of the policies [11], [13], [21]–[23]. Lastly, when the policy function has memory, either through an LSTM architecture or a stack of previous observations, dynamics randomization enables the policy function to learn how to infer the dynamics of the current environment using its memory, and therefore adapt the policy behavior based on the inferred dynamics [11].

We also add independent Gaussian noise to each element of the policy observations during training in order to improve the robustness of the policy to sensor uncertainty in the real world [11], [21]–[23]. In particular, we add correlated noise, sampled at the beginning of each episode, and uncorrelated noise, sampled at every time step, to the policy observations of the box pose and pusher position. Correlated noise simulates sensor bias and incorrect calibration, while uncorrelated noise simulates sensor noise. For example, let $O(x_b)$ be the policy observation of x_b, the current x position of the box. Then, during training, $O(x_b) = x_b + \delta_{x_b}^{\text{episode}} + \delta_{x_b}^{\text{step}}$, where $\delta_{x_b}^{\text{episode}}$ is sampled at the beginning of each episode and then kept fixed, while $\delta_{x_b}^{\text{step}}$ is sampled at every time step.

Finally, we apply random synthetic disturbances to the manipulated box throughout each training episode. Random disturbances have been shown to improve the robustness of the learned policies to external forces and unmodeled dynamics in the real world, and also enables the policy to explore more diverse scenarios [21]–[23]. We generate the disturbances by applying random forces on random locations in the box, and at random time steps throughout an episode.

3.4 Curriculum Learning

The goal of the policy is to move the box to the target pose. If the box reaches the target pose within the maximum horizon, and the box has velocity $0\,\mathrm{m\,s^{-1}}$, then the episode terminates successfully. Therefore, we define success thresholds $T_{x,y}$ and T_θ, corresponding to the position and the orientation respectively, such that, if $\|(x_b, y_b) - (x_{\text{targ}}, y_{\text{targ}})\| \leq T_{x,y}$ and $|\theta_b - \theta_{\text{targ}}| \leq T_\theta$, then the episode terminates successfully. Overall, the policy achieves the goal of the episode if the distance between the box position and the target position is at most $T_{x,y}$, the angular distance between the box orientation and the target orientation is at most T_θ, and the box has velocity $0\,\mathrm{m\,s^{-1}}$.

Note that the magnitude of the success thresholds $T_{x,y}$ and T_θ determines the accuracy of the learned policy. This is because there is no incentive for the policy to reduce the position or orientation error of the box pose, relative to the target pose, once the success thresholds are satisfied. Smaller $T_{x,y}$ and T_θ lead to more accurate learned policies; however, this is at the expense of increased task complexity and a sparser reward signal, which could lead to significantly slower training or lack of convergence entirely. To mitigate this issue, we use curriculum learning, which involves using a simpler task at the beginning stages of training, and then increasing the difficulty [39]. In particular, we define a curriculum such that the policy learning starts with larger thresholds $T_{x,y}$ and T_θ, which are reduced to $T_{x,y}/2$ and $T_\theta/2$ if the policy reaches a 90% average success rate. We refer to this reduction in the success thresholds as a *curriculum step*. Finally, we use a maximum of one curriculum step when training each policy, allowing the policy to acquire a success rate beyond 90% with the reduced success thresholds, in order to avoid an excessive increase in training complexity.

Chapter 4

Implementation

4.1 Policy Training

4.1.1 Training Set-up

We develop a custom simulation environment in which to train the RL policies for planar pushing. This enables us to define a workspace with similar characteristic to the real robot and implement key functionalities in order to utilize dynamics randomization, correlated and uncorrelated observation noise, as well as synthetic external disturbances during training. Additionally, our training environment allows us to enforce the defined curriculum and supports observation stacking, which is necessary when using an MLP architecture for the policy and value functions. We use PyBullet [40] as the physics engine for the environment since it is open source and it has been widely used to train RL policies for robot control [41], [42]. PyBullet also has rendering capabilities which allow us to visualize the behavior of the policies in simulation. Lastly, we use the PyBullet physics engine in the training environment in order to maintain a consistent implementation of the robot workspace since, as will be discussed in Section 4.2.3, we test the controller for policy deployment on the robot hardware using the ROS-PyBullet interface [43].

Our custom training environment follows the Gym [44] interface so as to be compatible with Stable Baselines3 [28], a collection of open source implementations of RL algorithms based on PyTorch [45]. We use the PPO and SAC implementations from Stable Baselines3 since they are reliable, well tested, and have been widely used in previous works [13], [46], [47]. Stable Baselines3 also provides functionalities to develop custom policy and value functions, which we leverage for our work.

The default configuration of the robot workspace in the training environment consists of a bounded planar surface of size 60×35 cm, a $10 \times 12 \times 7$ cm box, and a spherical pusher of radius 1.25 cm. This is designed to be relatively similar to the robot workspace in the real world. Furthermore, the planar pushing policies are executed in the environment at a frequency of 30 Hz. A higher frequency enables more precise control; however, it requires a greater number of policy executions in order to complete the same number of training episodes, which slows down training. We selected a frequency

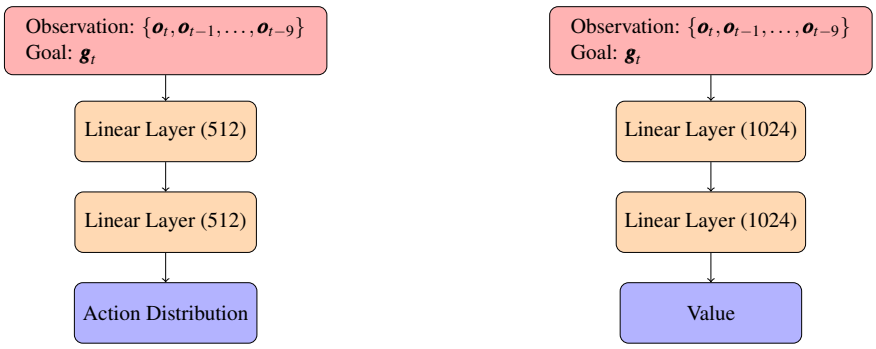


Figure 4.1: MLP architecture for the policy function (left) and value function (right). Observation stacking is used and the size of each hidden layer is shown in parenthesis.

of 30 Hz after performing experiments with different frequencies and concluding that it provides a good compromise between precision and computational complexity at training time. Note that the first two dimensions of the box, corresponding to the x and y axis, the radius of the pusher, and the policy execution frequency, are randomized during training, which will be further discussed in Section 4.1.4. Additionally, we abstract away the robot control during training and apply the policy actions to the spherical pusher directly. In particular, the pusher remains at a constant height of 3.5 cm in the z axis and, at each time step, its planar velocity $(v_{x,p}, v_{y,p})$ is updated according to the selected policy action. This significantly improves the computational complexity of the training environment, which enables us to train the RL policies with a larger amount of environment interactions.

We train the policies with data collected from 128 actors. This involves deploying the current policy in 128 parallel instantiations of our custom planar pushing environment to collect data, using the data to update the policy, and then repeating this process. Furthermore, we define a maximum episode length of $H = 300$ time steps, which corresponds to 10 seconds in real-time since the policy execution frequency is 30 Hz. Therefore, if the policy does not accomplish the episode goal within 300 time steps, the episode terminates unsuccessfully. We determined that $H = 300$ is suitable for our task after experimentally validating that the vast majority of the possible pushing tasks in our environment can be completed within this time limit given the velocity constraints of the pusher. Our reward function is as defined in Equation (3.1) and we use parameter values $\alpha = 50$, $\beta = 20$, $k_1 = 0.1$, $k_2 = 0.02$, and $k_3 = 0.004$. We evaluated various reward function formulations, including purely dense and purely sparse functions, as well as various parameter values within these formulations, and concluded that the aforementioned reward function leads to the best performance and training efficiency. Lastly, we train all policies using a workstation with an Intel Core i9 (3.60GHz) CPU, an Nvidia GeForce RTX 2080 GPU, and 64 GiB of RAM.

4.1.2 Network Architectures

As discussed in Section 3.1, we experiment with two different neural network architectures of the policy and value functions in PPO: an MLP architecture that receives a stack of previous observations, and an LSTM architecture. Figure 4.1 illustrates the

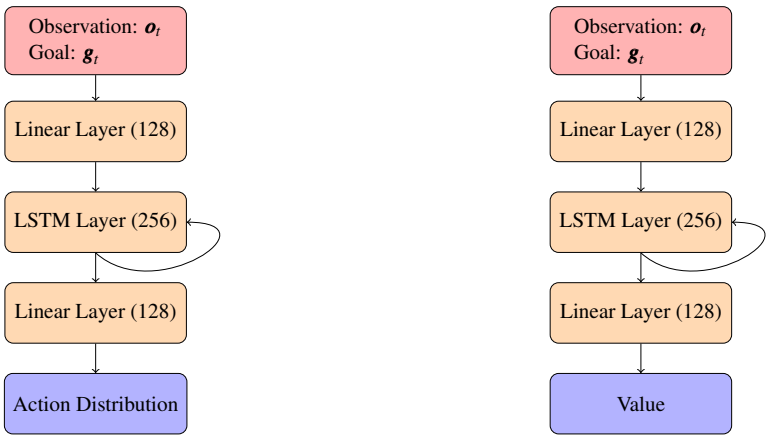


Figure 4.2: LSTM architecture for the policy function (left) and value function (right). The size of each hidden layer is shown in parenthesis.

MLP architecture. We use a stack of 10 observations in order to provide substantial information to infer the hidden dynamics, without excessively increasing the dimensionality of the input. This is a similar configuration to the observation stacking previously used by Peng *et al.* [11] for RL planar pushing policies. In the policy function, we use 2 hidden linear layers, each of size 512, while in the value function we use 2 hidden linear layers, each of size 1024. Furthermore, Figure 4.2 illustrates the LSTM architecture. The policy and value functions have the same structure of hidden layers, specifically, a linear layer of size 128, an LSTM layer of size 256, and a linear layer of size 128.

We empirically evaluated different depths and widths of the policy and value functions in both architectures, and concluded that the illustrated architectures work well for our task. In general, we found that smaller networks led to slower convergence and a lower asymptotic performance, while larger networks created an unstable training regime. It is worth mentioning that we normalize the observation and goal before feeding them through the policy and value functions. Additionally, the output action velocities are in the range $[-1,1]$, which we then scale to the range $[-0.1,0.1]$. Finally, we use `tanh` nonlinearities in both architectures as this is common practice in PPO [27]–[30], [48].

For our experiments with SAC, we only use the MLP architecture due to the unavailability of an open source implementation with LSTM models compatible with our training environment. Cong *et al.* [13] also used SAC with an MLP architecture to train planar pushing policies. Additionally, when using the same RL algorithm with MLP (stacking observations) and LSTM architectures, if the algorithm converges with one of the architectures, it generally converges with the other as well, the main difference being in convergence speed and final performance [11], [21]. We use the same architecture for SAC as in Figure 4.1 with minor adjustments since we did not find alternative architectures to yield any significant advantages. In particular, we use `ReLu` nonlinearities since this is more common across SAC implementations [26], [28]. Additionally, SAC learns two separate state-action value functions (also referred to as Q-functions), rather than a single state value function as is the case with PPO [28]. The architecture for the Q-functions in SAC is the same as the value function in Figure 4.1, except that the input includes the action $\boldsymbol{a}_t$ in addition to the observation stack and the goal.

Hyperparameter	Value
Clip range (ε)	0.2
GAE parameter (λ)	0.95
Discount factor (γ)	0.99
Value function coefficient (c_1)	0.5
Entropy bonus coefficient (c_2)	0
Epochs	10
Optimizer	Adam [32]
Learning rate	$3 \cdot 10^{-4}$
Batch size	7680
KL divergence threshold	0.01

Table 4.1: PPO hyperparameters.

4.1.3 Policy Hyperparameters

We select the hyperparameter values of PPO and SAC based on previous works as well as our own hyperparameter exploration to ensure that they are well suited for the planar pushing task. Table 4.1 summarizes the hyperparameters used in PPO. We select the clip range, GAE parameter, discount factor, optimizer, and learning rate based standard values used in previous works [21], [27], [48]. The coefficients of the value function and entropy bonus are also common across PPO implementations [27]–[29], [49]. Given the importance of policy exploration for our work, we evaluated alternative values of the entropy bonus coefficient, namely $c_2 \in \{0.005, 0.01, 0.02\}$; however, we did not find the entropy bonus to have a positive effect on policy training.

During training, we deploy the policy for 300 time steps on each parallel environment, then run the optimizer on the objective function to update the policy and value functions using using the data collected from these interactions, and repeat this process. The number of epochs and batch size are hyperparameters of the optimizer and they are highly dependent on the characteristics of the task being solved. We conducted hyperparameter exploration for the number of epochs using values $\{5, 10, 15\}$, as well as the batch size using values $\{1920, 3840, 7680, 12800\}$, and concluded that 10 epochs and batch size 7680 are well suited for the planar pushing task.

PPO uses a clipped policy objective in order to prevent excessively large updates of the policy function. Nevertheless, during training we observed abrupt increases in the Kullback-Leibler (KL) divergence [50] between the updated policy and the previous policy, which were closely followed by a collapse in the performance of the policy. Such behavior is an indicator of excessively large updates of the policy function causing instability and failure to converge. Here, the KL divergence is a measurement of the mean difference between the action distributions generated by the updated policy and the previous policy in the environment states encountered during policy deployment. Note that this unstable behavior persisted even when decreasing the clip range. A common technique to mitigate excessively large policy updates in PPO is early stopping of the optimizer when the KL divergence between the updated policy and the previous

policy exceeds a certain threshold, usually around 0.01 [28], [30]. We also use this technique since we found it to be highly effective when training our policies.

For our experiments with SAC we use the same optimizer, batch size, and learning rate as in PPO. We evaluated alternative options but did not find them to yield any significant advantages. Furthermore, SAC is an off-policy algorithm and hence stores data from previous environment interactions in a replay buffer, which is then used to update the policy and Q-functions. We use a buffer size of size 10^6 for our experiments as in previous implementations [26], [28], [30]. The remaining hyperparameters are standard as originally proposed for SAC [26].

4.1.4 Learning Task Implementation

We now discuss the characteristics of the task that we aim to solve during policy training. In particular, we aim to solve planar pushing tasks for arbitrary initial and target box poses. Furthermore, we aim to reach the target box pose with significant accuracy, for which we leverage curriculum learning. Finally, we aim to learn policies in simulation that can adapt to the dynamics of different environments, including our physical robot hardware. To this end, we use dynamics randomization, observation noise, and external disturbances during training.

At the beginning of every episode, we generate a random initial configuration of the box and the pusher, as well as a random target box pose. The initial and target box positions are independently and uniformly sampled from the available planar workspace. Additionally, the initial and target box orientations are independently sampled from $\mathcal{U}([-\pi,\pi])$ rad. Lastly, the initial pusher position is sampled uniformly from a perimeter around the initial box pose. This enables us to facilitate exploration in the beginning stages of training and hence reduce training time. Specifically, since the pusher is close to the box, the policy actions are more likely to lead the pusher to interact with the box and receive feedback from the reward function. Nevertheless, as can be seen in the supplemental video, this does not prevent the policy from learning how to act when there is a large separation between the pusher and the box, presumably due to the synthetic disturbances encountered during training.

Recall from Section 3.4 that we consider an episode successful if the box reaches the target pose within certain position and orientation thresholds $T_{x,y}$ and T_θ, respectively, and the box has velocity $0\,\mathrm{m\,s^{-1}}$. We define a curriculum such that if the policy exceeds a 90% success rate, averaged over the 128 parallel environments and the previous 100 episodes, the success thresholds are reduced to $T_{x,y}/2$ and $T_\theta/2$. In particular, during the first stage of the training process we use thresholds

$$T_{x,y}^{(1)} = 1.5\,\mathrm{cm}, \quad T_\theta^{(1)} = 0.34\,\mathrm{rad} \approx 19.5\,\mathrm{deg}. \tag{4.1}$$

Then, if the curriculum step is taken, the success thresholds become

$$T_{x,y}^{(2)} = 0.75\,\mathrm{cm}, \quad T_\theta^{(2)} = 0.17\,\mathrm{rad} \approx 9.7\,\mathrm{deg}. \tag{4.2}$$

Finally, we consider an episode unsuccessful if the maximum episode length is reached without successfully completing the task as previously described, as well as if at any point the pusher or the box leave the bounded planar workspace.

Parameter	Sampling Distribution
Friction	$\mathcal{U}([0.5, 0.7])$
Restitution	$\mathcal{U}([0.4, 0.6])$
Box Length	$\mathcal{U}([0.115, 0.125])$ m
Box Width	$\mathcal{U}([0.095, 0.105])$ m
Box Mass	$\mathcal{U}([0.4, 0.6])$ kg
Pusher Radius	$\mathcal{U}([0.012, 0.013])$ m
Action Duration	$\mathcal{N}(1/30, (1/320)^2)$ s
Position Noise	$\mathcal{N}(0, 0.001^2)$ m
Orientation Noise	$\mathcal{N}(0, 0.02^2)$ rad

Table 4.2: Dynamics randomization and observation noise parameters.

Table 4.2 shows the parameters and corresponding sampling distributions for the dynamics randomization and observation noise. We sample the dynamics parameters from their respective distributions at the beginning of each episode. Additionally, we independently sample correlated (per episode) and uncorrelated (per time step) noise for the observation of the position of the box and the position of the pusher from the Position Noise distribution in Table 4.2. Similarly, we add correlated and uncorrelated noise to the observation of the box orientation by sampling from the Orientation Noise distribution in Table 4.2. Finally, at every time step we apply a disturbance to the box with probability 1%, in a uniformly random location throughout the box, and with force in x and y independently sampled from $\mathcal{U}([-25, 25])$ N.

4.1.5 Planar Pushing Environment with Two Pushers

In order to evaluate the scalability of our framework with respect to the number of pushers, we develop a custom planar pushing training environment which enables the RL policy to control two pushers. This environment has similar characteristics to the main training environment with one pusher as discussed in Section 4.1.1. Nevertheless, several important modifications are required.

We augment the action space such that $\boldsymbol{a}_t = \left(v_{x,p}^{(1)}, v_{y,p}^{(1)}, v_{x,p}^{(2)}, v_{y,p}^{(2)}\right)$ in order to include the x and y velocity of the two pushers. Additionally, to guarantee that the policy only explores motions that are feasible for a bi-manual manipulation platform, we add two constraints:

(a) Each pusher can exert pushing forces with a maximum magnitude of 75 N.

(b) The distance between pushers in the x coordinate must be at least 5 cm.

If the policy violates any of these constrains, the episode terminates unsuccessfully. With a single pusher, the magnitude of the pushing force is naturally limited by the dynamics of the environment and the velocity of the pusher; however, with two pushers the policy can exert forces of arbitrary magnitudes, such as by moving the pushers toward each other when they are in opposite sides of the box. Hence, the purpose of constraint (a) is to prevent excessive forces. On the other hand, constraint (b) prevents

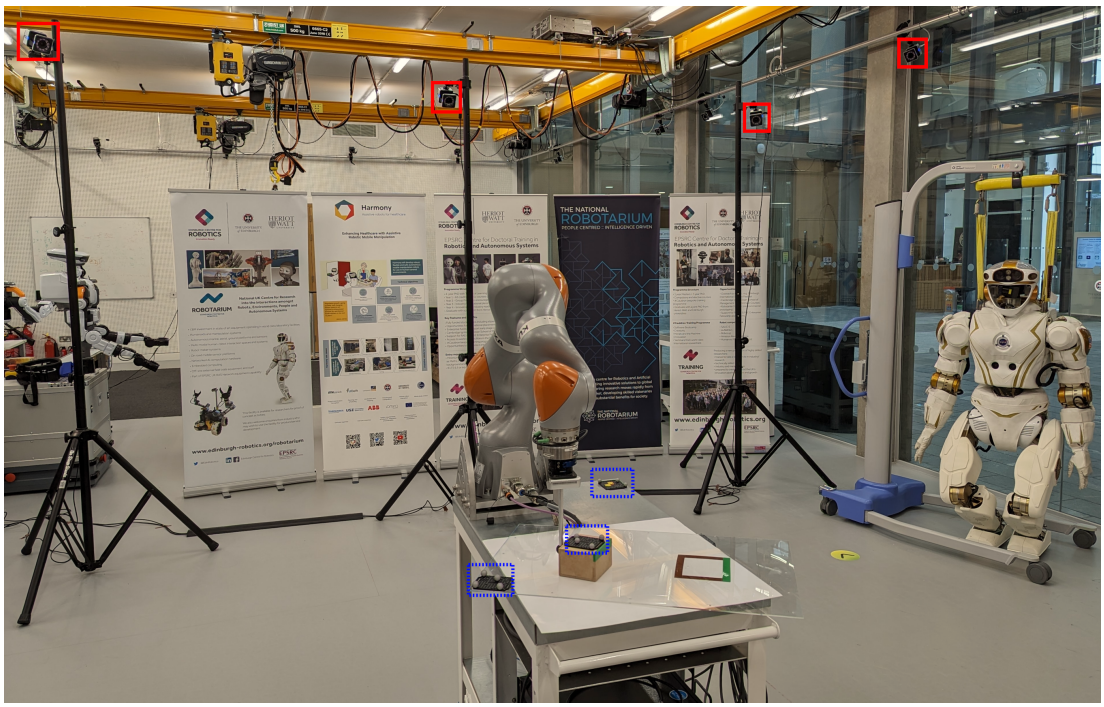


Figure 4.3: Robot hardware set-up for planar pushing. Motion capture cameras are labeled with solid red squares. Reflective markers are labeled with dotted blue squares.

pusher trajectories that would be unrealizable when using two robotic arms to control the pushers. An example of an unrealizable trajectory is one pusher rotating around the other. We require one pusher to be always to the right of the other, which should be sufficient to guarantee the realizability of the trajectories. Finally, we sample the starting position as well as the dynamics randomization and observation noise parameters corresponding to each pusher independently.

4.2 Policy Deployment on the Robot

4.2.1 Hardware Set-up

We assemble a planar pushing robot set-up, depicted in Figure 4.3, in order to evaluate the transferability of the trained policies to the real-world. The robot we use is a KUKA LBR iiwa robot arm, which has 7 degrees of freedom and is mounted on a planar workspace. The end-effector of the robot is a cylinder with a spherical pusher at the end. Note that we only use one pusher for our real-world experiments as this is the main task explored in this work. Additionally, we use a wooden box as the manipulated object. The workspace, pusher, and box have similar dimensions to those in our simulated environment, as described in Section 4.1.1. Furthermore, in order to maneuver and visualize the target pose, we use an acrylic plastic sheet with a marked rectangle that represents the target pose. One of the sides of the box is marked with green tape, and the rectangle corresponding to the target pose also has a green side. This is done to illustrate the orientation of the box pose and the target pose. The functionality of the robot, the manipulated box, and the target pose is best seen in the supplemental video.

We use a Vicon motion capture system to track the state of the environment. Some of the Vicon cameras used are shown in Figure 4.3, and there are several more at the sides and behind in order to obtain a complete view of the environment from multiple angles. We have a total of 22 active cameras around the robot platform, which enables us to obtain significantly accurate readings and avoid occlusions. Some cameras were already mounted to the ceiling; however, these did not provide sufficient coverage of the environment, so we added new cameras using temporary stands. Since this was a new camera set-up, we calibrated the Vicon system to ensure that the position and orientation of each camera is known with significant accuracy, which is crucial to obtain accurate readings of the environment state.

The Vicon cameras track the position of spherical reflective markers relative to a world coordinate frame, which is located around the center of the floor of the environment. These reflective markers can be seen in Figure 4.3. We are interested in tracking the current box pose, the target box pose, and the robot pose. Therefore, we place clusters of reflective markers on the base of the robot, on the top of the manipulated box, and on the acrylic sheet with the target pose. We use three markers for the robot and four markers for the box and the acrylic sheet since the latter experience more movement. Most importantly, we use these clusters of markers to define the coordinate frames F_r, F_b, and F_{targ} relative to the world frame. F_r lies at the origin of the robot, where the base is anchored to the workspace, F_b lies at the center of the box, and F_{targ} lies at the center of the target pose. Finally, we align the orientations of F_r, F_b, and F_{targ} with those of the robot, the box, and the target, respectively. Overall, these coordinate frames enable us to obtain the robot pose, box pose, and target pose in the environment.

4.2.2 Robot Controller

In order to deploy the trained RL policies on the physical robot set-up, we develop a controller using the Robot Operating System (ROS) [51], the most widely used software framework for robot applications. We had access to an existing code base to control the robot, which we modify so as to track the state of the planar pushing environment and generate the control commands using the RL policy. Figure 4.4 illustrates the high-level functionality of the robot controller. Our primary addition to the code base is the implementation of the controller interactions with the RL policy and the Vicon motion capture system. We now discuss the functionality of the controller.

We begin with the task of tracking the state of the environment. The controller receives the current box pose, target box pose, and robot pose from the Vicon system. As discussed in the previous section, these are in the form of coordinate frames F_b, F_{targ}, and F_r, respectively, relative to the world frame of the Vicon system. Furthermore, we define a workspace coordinate frame F_w, relative to F_r, which lies at the origin of the planar pushing workspace of the robot. This enables us to determine the box pose (x_b, y_b, θ_b) and the target pose $(x_{\text{targ}}, y_{\text{targ}}, \theta_{\text{targ}})$ relative to the workspace frame. This is necessary since the RL policy was trained using observations relative to F_w.

The robot pose F_r received from the Vicon system only tells us the location of the robot in the external environment; however, the state of the robot arm remains unknown. The controller uses forward kinematics to compute the position of the pusher relative to F_r

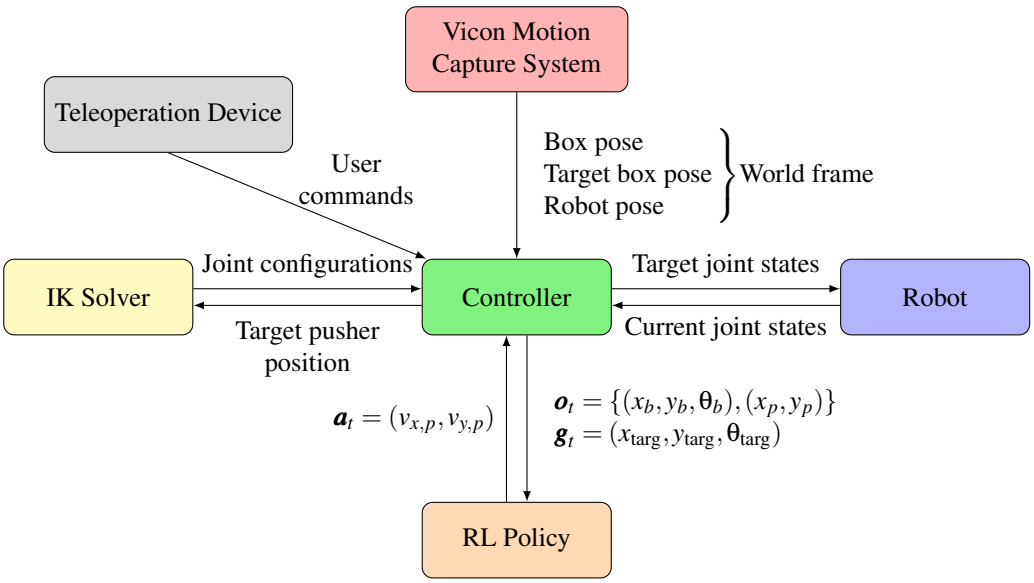


Figure 4.4: Functionality of the robot controller for planar pushing.

based on the joint states $\boldsymbol{q} \in Q$ received from the robot. Note that Q is the configuration space, the set of possible joint states of the robot. Then, we can calculate the pusher position (x_p, y_p) relative to the workspace frame F_w. Overall, we use the Vicon system and robot joint states to determine the box pose, target pose, and pusher position relative to F_w, and hence construct the observation and goal of the RL policy.

Once we have the observation and goal, we normalize them and feed them to the RL policy to obtain the predicted action $(v_{x,p}, v_{y,p})$. The robot is position controlled in the configuration space at a frequency of 100 Hz. Hence, we send target joint states to the robot at 100 Hz. However, the RL policy outputs 2D velocities of the pusher and, as discussed in Section 4.1.1, it runs at 30 Hz. In order to determine the target joint states required to realize the policy actions on the robot, we integrate the pusher velocity from the policy action to obtain target pusher positions, and then use an IK solver to map the target pusher positions to joint configurations. In particular, the RL policy updates the predicted pusher velocity $(v_{x,p}, v_{y,p})$ at 30 Hz. On the other hand, the change in the pusher position corresponding to every target joint state sent to the robot is given by $(\Delta x_p, \Delta y_p) = (v_{x,p} \cdot \Delta t, v_{y,p} \cdot \Delta t)$, where $\Delta t = 1/100$ in our case since we send target joint states at 100 Hz. Note that we enforce a fixed vertical position of the pusher $z_p = 5\,\text{cm}$ such that $\Delta z_p = 0$. Using the current joint states $\boldsymbol{q}$ and the desired change in pusher position $(\Delta x_p, \Delta y_p, \Delta z_p)$, the IK solver calculates the change in joint states $\Delta\boldsymbol{q}$ required to achieve the desired change in pusher position. The IK solver is implemented using OpTaS [52]. Finally, we send to the robot the target joint states $\boldsymbol{q} + \Delta\boldsymbol{q}$.

We use a 3Dconnexion SpaceMouse Compact device to teleoperate the robot. It has two buttons and a joystick sensor with 6 degrees of freedom. This enables us to start and stop the execution of the RL policy, as well as to move the robot to different desired locations in the workspace for our experiments. When using the SpaceMouse to move the robot, the user inputs specify the motion of the robot pusher, which is mapped to target joint states by the controller using an IK solver as previously discussed.

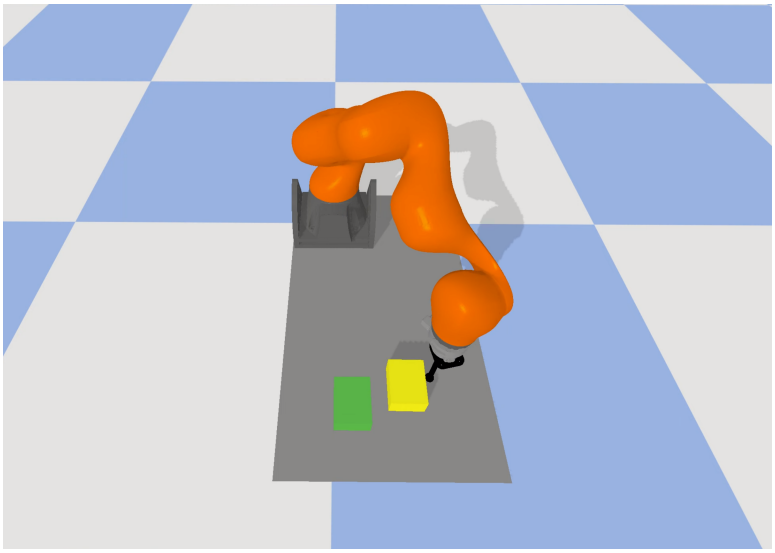


Figure 4.5: Simulation environment to test the robot controller. The yellow box is the manipulated object and the green box is a visualization of the target pose.

4.2.3 Testing Environment

Finally, we develop a testing environment using the ROS-PyBullet interface [43] in order to validate the implementation of the robot controller in simulation before deploying it on the real robot. Figure 4.5 shows the testing environment, which we design such that it closely matches the real-world planar pushing set-up. The environment is simulated using PyBullet and we represent the KUKA robot arm using a Unified Robot Description Format (URDF) model. The ROS-PyBullet interface enables the communication between the robot controller, implemented on ROS, and the PyBullet simulation environment. To test the developed controller, we keep its entire functionality the same, as shown in Figure 4.4, except for two components:

- We re-map the communication between the Vicon motion capture system and the controller such that the box pose, target box pose, and robot pose received by the controller are from the corresponding elements in the simulation environment.
- We re-map the communication between the robot and the controller such that the current joint states received by the controller are from the simulated robot, and the target joint states sent by the controller are executed on the simulated robot.

Overall, this testing environment enables us to run our robot controller, with minimal adaptations, in a simulated representation of our real-world robot set-up for planar pushing.

Chapter 5

Experiments and Results

5.1 Simulation Experiments

5.1.1 Learning the Planar Pushing Task

We begin with the process of training the RL policies in our simulated planar pushing environment, using the standard set-up with one pusher. We train PPO policies with the MLP and LSTM architectures and, for each architecture, we compare our proposed categorical exploration approach against the previously used Gaussian exploration approach. We also train a SAC policy with the MLP architecture and Gaussian exploration. Figure 5.1 illustrates the resulting learning curves.

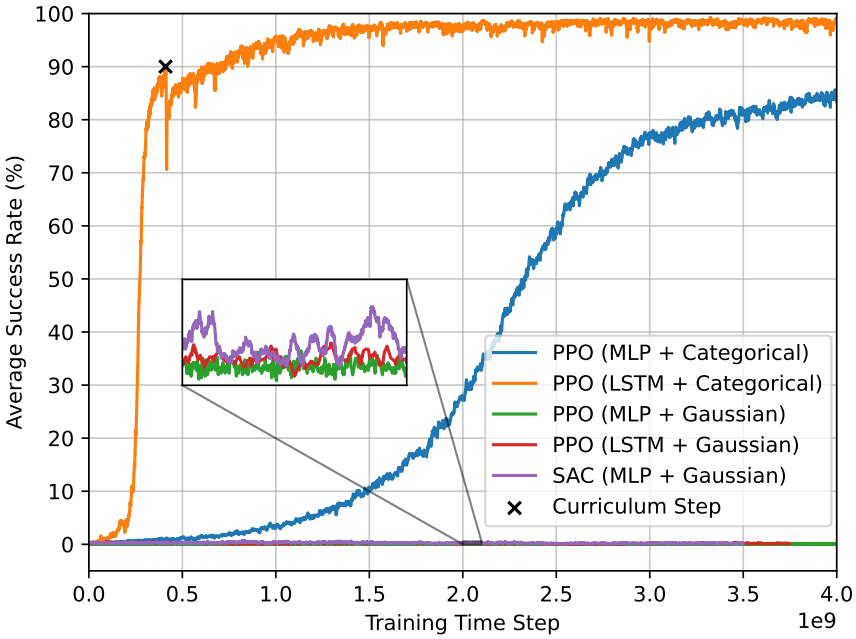


Figure 5.1: Training performance of the RL frameworks evaluated with our proposed categorical exploration and the previously used Gaussian exploration. Success rate is averaged over 128 parallel environments and the previous 100 episodes.

We find that only the policies using our proposed categorical exploration approach manage to learn the planar pushing task for arbitrary initial and target object poses. The policies using Gaussian exploration are unable to make any meaningful progress in the learning task. In particular, the PPO policies enable us to compare the categorical and Gaussian exploration approaches while maintaining the underlying learning algorithm, function approximation architectures, and hyperparameters constant. Therefore, our findings suggest that the failure of the Gaussian policies to make progress in the learning task may be attributed to poor exploration of the state-action space. We believe that this is due to the inherently unimodal Gaussian preventing effective exploration of the different possible contact modes required to control the orientation of the pushed object.

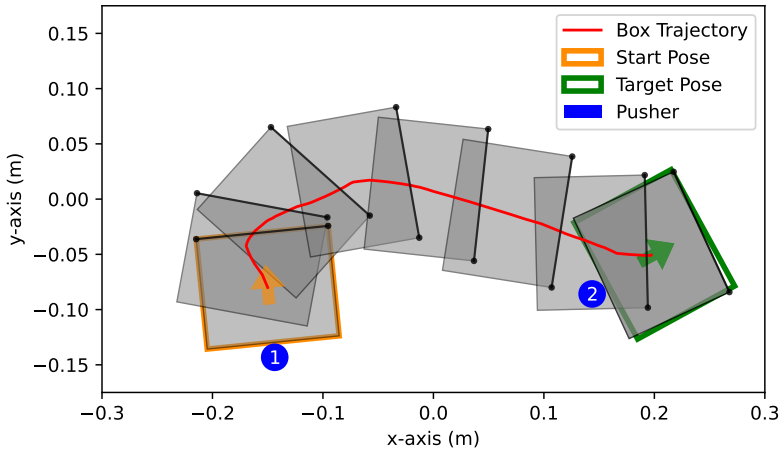
The SAC policy serves as a baseline for comparison with previous works applying RL to the planar pushing task. Cong *et al.* [13] recently used SAC with an MLP architecture and Gaussian exploration for planar pushing. Furthermore, the other previously discussed works on RL for planar pushing employ off-policy algorithms with Gaussian exploration [11], [12], which are essential attributes of SAC. We use the same SAC implementation as Cong *et al.* [13]; however, we were unable to evaluate additional approaches from previous works since their code implementation is not publicly available. Figure 5.1 shows that similar to the other policies with Gaussian exploration, the SAC policy fails to make any meaningful progress in learning the planar pushing task for arbitrary initial and target object poses. This is expected since the previous works applying RL to the planar pushing task disregarded the orientation of the pushed object [11]–[13]. As previously discussed, we believe the Gaussian exploration used in these works prevents effective exploration of the different contact modes.

Focusing on the policies with categorical exploration, the learning curves in Figure 5.1 show that both policies converge with significant stability. The LSTM architecture provides substantially faster convergence and hence reaches the curriculum step, which is expected since, as discussed in Section 3.1, it can capture long-term dependencies as well as learn to store and forget certain information during the interactions. Despite the slower convergence of the MLP architecture with frame stacking, which prevented it from reaching the curriculum step, it was still capable of obtaining considerable performance during training. Overall, our best policy is PPO (LSTM + Categorical). It converges rapidly and reaches a 90% average success rate early on. This triggers the curriculum step, which reduces the position and orientation success thresholds to half of their initial values. The curriculum is effective at providing rapid initial convergence in the first stage, and then guiding the policy to learn significantly accurate planar pushing motions in the second stage. In particular, the PPO (LSTM + Categorical) policy consistently achieves over 98% average success rate with a position threshold of $T_{x,y}^{(2)} = 0.75\,\text{cm}$ and an orientation threshold of $T_{\theta}^{(2)} = 0.17\,\text{rad} \approx 9.7\,\text{deg}$.

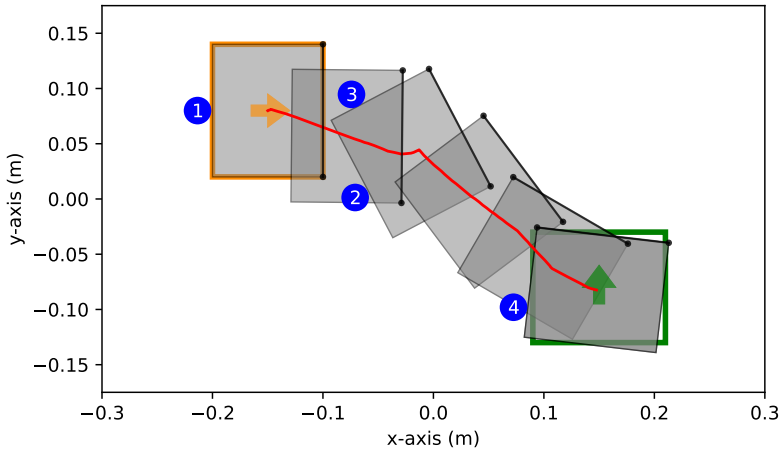
We trained all policies for $4 \cdot 10^9$ time steps in our simulated planar pushing environment. Figure 5.1 omits the last few time steps of two Gaussian policies to improve the visualization. On average, each policy required around seven days to train in our RL workstation. As a result, we were unable to repeat the policy training experiments with multiple random seeds for more robust results due to computational constraints. This also applies to the policy training experiments in Section 5.1.4 and Section 5.1.6.

5.1.2 Pushing Trajectories

In this experiment, we analyse the pushing trajectories generated by the PPO (LSTM + Categorical) policy in simulation. We use the reduced position and orientation success thresholds $T_{x,y}^{(2)}$ and $T_{\theta}^{(2)}$. Figure 5.2a shows the resulting trajectory for a relatively simple pushing task that does not require sharp rotations of the box. The policy manages to maneuver the box such that it follows a smooth and efficient trajectory reaching the target pose with significant accuracy, thereby successfully completing the task. The pusher remains in contact with the box throughout the entire trajectory. Figure 5.2b shows the resulting trajectory for a more difficult pushing task where the policy performs a face switch. The policy first re-orients the box, breaks contact at point 2, makes contact again at point 3, and finally pushes the box to the target and completes the task. Note that the boundary in the plot corresponds to the workspace boundary, so here a face switch is necessary in order to solve the task. The policy successfully executes the required phase switch in an efficient manner and exhibits smooth pushing trajectories.



(a) Simple pushing task



(b) Pushing task with a phase switch

Figure 5.2: Trajectories generated by the PPO (LSTM + Categorical) policy in simulation. The pusher numbers indicate chronological order and the plot boundaries correspond to the workspace boundaries.

Based on our simulation experiments, we observe that the PPO (LSTM + Categorical) policy not only solves the planar pushing task for arbitrary initial and target object poses, unaddressed by previous RL works, but also significantly improves upon the sub-optimal idiosyncratic pushing trajectories exhibited by the previous methods. As discussed in Section 2.1.3, previous RL methods for planar pushing achieve low accuracy, deeming episodes with a position error of 2.5 cm and greater as successful, and they rely on repeated sub-optimal short pushing motions to approximate the object to the target position over multiple attempts [11]–[13]. The PPO (LSTM + Categorical) policy consistently solves the task under position and orientation success thresholds of 0.75 cm and 0.17 rad, and exhibits significantly smooth and efficient pushing trajectories. In particular, it achieves improved control over the manipulated object, purposefully maintaining contact throughout longer pushing motions, and thereby avoiding the use of sub-optimal short pushes to approximate the object to the target over multiple attempts.

5.1.3 Observation Noise

We are interested in evaluating the robustness of the PPO (LSTM + Categorical) policy to observation noise. To this end, we evaluate the performance of the policy under various combinations of correlated and uncorrelated observation noise levels. The noise levels are measured by the standard deviations of the Position Noise and Orientation Noise sampling distributions, which we introduced in Section 4.1.4. For each combination of correlated and uncorrelated observation noise, we set-up our simulated planar pushing environment with the corresponding noise levels and calculate the success rate and time to target of the policy, averaged over 1000 episodes with random starting configurations and target poses. Note that the dynamics are randomized and external disturbances are generated as during policy training. We use the reduced position and orientation success thresholds $T_{x,y}^{(2)}$ and $T_{\theta}^{(2)}$. Additionally, we increase the time limit from 10 to 30 seconds to allow the policy to make adjustments if necessary, which can be particularly relevant in settings with significant observation noise. The results are recorded in Table 5.1.

		Uncorrelated Noise (SD)			
		0 cm *0 rad*	*0.1 cm* *0.02 rad*	*0.3 cm* *0.06 rad*	*0.45 cm* *0.09 rad*
Correlated Noise (SD)	*0 cm* *0 rad*	98.5 ± 0.4 % 4.9 ± 0.1 sec	99.4 ± 0.3 % 4.8 ± 0.1 sec	98.6 ± 0.4 % 5.0 ± 0.1 sec	98.7 ± 0.4% 5.4 ± 0.1 sec
	0.1 cm *0.02 rad*	99.1 ± 0.3 % 4.8 ± 0.1 sec	98.6 ± 0.4% 4.7 ± 0.1 sec	98.5 ± 0.4 % 5.1 ± 0.1 sec	98.6 ± 0.4 % 5.4 ± 0.1 sec
	0.3 cm *0.06 rad*	94.4 ± 0.7 % 5.6 ± 0.1 sec	96.9 ± 0.5% 5.8 ± 0.1 sec	98.0 ± 0.4 % 6.0 ± 0.1 sec	96.0 ± 0.6 % 6.5 ± 0.1 sec
	0.45 cm *0.09 rad*	84.2 ± 1.2 % 6.5 ± 0.1 sec	89.2 ± 1.0 % 6.8 ± 0.1 sec	90.2 ± 0.9 % 7.5 ± 0.1 sec	88.4 ± 1.0 % 8.2 ± 0.2 sec

Table 5.1: Average success rate and time to target, and corresponding estimated standard error, with correlated (per episode) and uncorrelated (per step) observation noise. Results for the training set-up are underlined.

We find that the policy is significantly robust to both correlated and uncorrelated observation noise. Increasing the uncorrelated noise generally leads to a slight increase in the time to target. Additionally, the success rate seems to be slightly higher with some uncorrelated noise rather than none, especially with larger correlated noise levels. This could be because the uncorrelated noise results in the policy receiving more diverse observations of the environment, some of which could trigger certain policy behaviors that are beneficial for solving the task. Moderate levels of uncorrelated noise should not pose major challenges since the noise is centered around zero and it is generated at every time step.

On the other hand, we expect correlated noise to significantly increase the difficulty of the task since it remains fixed throughout the episode and hence causes the policy to receive an observation of the environment that is consistently shifted relative to the true state. For example, the policy may need to make contact with a particular corner of the box in order to execute a sharp rotation; however, the correlated noise might cause the policy to perceive the corner always shifted by a significant amount, which could potentially result in the policy failing to make contact with the box and therefore executing the required rotation of the box. We observe in Table 5.1 that increasing the correlated observation noise indeed generally leads to a decrease in the success rate and an increase in the time to target, which becomes more noticeable at larger noise levels.

Overall, the policy exhibits significant robustness, consistently achieving over 95% success rate and a time to target below 6 seconds, even in settings with considerable correlated and uncorrelated observation noise. The worst success rate was 84.2% and the worst time to target was 8.2 seconds, obtained under different settings. This also demonstrates significant robustness considering the large amount of observation noise, with standard deviations for the position and orientation sampling distributions of 0.45 cm and 0.09 rad respectively, which often lead to noise levels of comparable magnitudes to the success thresholds $T_{x,y}^{(2)} = 0.75\,\text{cm}$ and $T_{\theta}^{(2)} = 0.17\,\text{rad}$.

5.1.4 Limits of Gaussian Exploration

In order to better understand why the policies with Gaussian exploration did not manage to learn the planar pushing task for arbitrary initial and target object poses, we examine their training performance in a simplified setting. To this end, we modify our planar pushing simulation environment as follows. The initial position of the box is uniformly sampled from the left side of the workspace ($x < 0$), while the target position of the box is uniformly sampled from the right side of the workspace ($x > 0$). Additionally, we restrict the sampling distribution for the initial and target orientations of the box to $\mathcal{U}([-\pi/4, \pi/4])$ rad. This results in a significantly simpler pushing task since the initial box pose is always approximately oriented towards the target pose, thereby eliminating the need for complex maneuvers of the box in order to complete the task, such as sharp rotations or phase switching. In other to preserve this simplified setting throughout the task we also remove the synthetic external disturbances. Furthermore, the initial position of the pusher is uniformly sampled such that it is in contact with the backside of the box. This enables the policy to naturally push the box towards the target in the right side of the workspace without needing to break contact with the box. We use

constant success thresholds $T_{x,y}^{(1)} = 1.5\,\text{cm}$ and $T_{\theta}^{(1)} = 0.34\,\text{rad} \approx 19.5°$ throughout.

For this experiment, we define a different curriculum than previously used. In particular, if the policy reaches a 90% average success rate, we increase the width of the uniform sampling distribution for the initial and target box orientations by $\pi/2$, still remaining centered around zero. For example, in the first curriculum step, the sampling distribution transitions from $\mathcal{U}([-\pi/4, \pi/4])$ rad to $\mathcal{U}([-\pi/2, \pi/2])$ rad. The maximum number of curriculum steps is three as this would result in the sampling distribution $\mathcal{U}([-\pi, \pi])$ rad, which encompasses the entire range of orientations. This curriculum enables us to gradually increase the complexity of the pushing task once the policies develop sufficient skills in the simpler settings. Specifically, in the first setting the policies can solve the task without requiring any complex maneuvers of the box. As the width of the sampling distribution increases, the policies must learn to perform sharper rotations of the box as well as phase switching in order to master the task and proceed to the next curriculum step. We can examine the progression of the policies through the curriculum to determine at which point the task complexity hinders further improvement, and hence analyze the planar pushing features that are problematic for these policies. The learning curves obtained by the policies with Gaussian exploration are shown in Figure 5.3.

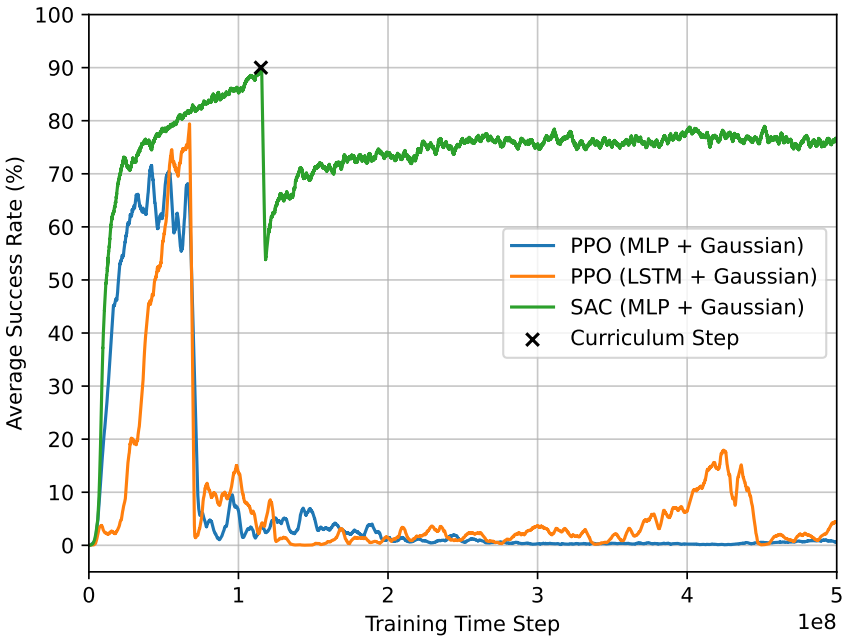


Figure 5.3: Training performance of the policies with Gaussian exploration on a simplified planar pushing task. A different curriculum is used which increases the range of starting and target orientations upon reaching 90% average success rate.

We can see that the PPO policies do not manage to improve beyond 80% average success rate and, in fact, experience a collapse in performance. The SAC policy manages to reach 90% average success rate once, which triggers the first curriculum step, thereby increasing the range of starting and target orientations to $[-\pi/2, \pi/2]$ rad. However, the SAC policy is not able to reach the next curriculum step. Most pushing tasks in this simplified setting, especially when the starting and target orientations remain within $[-\pi/2, \pi/2]$ rad, do not require significant object rotations and hence phase-switching,

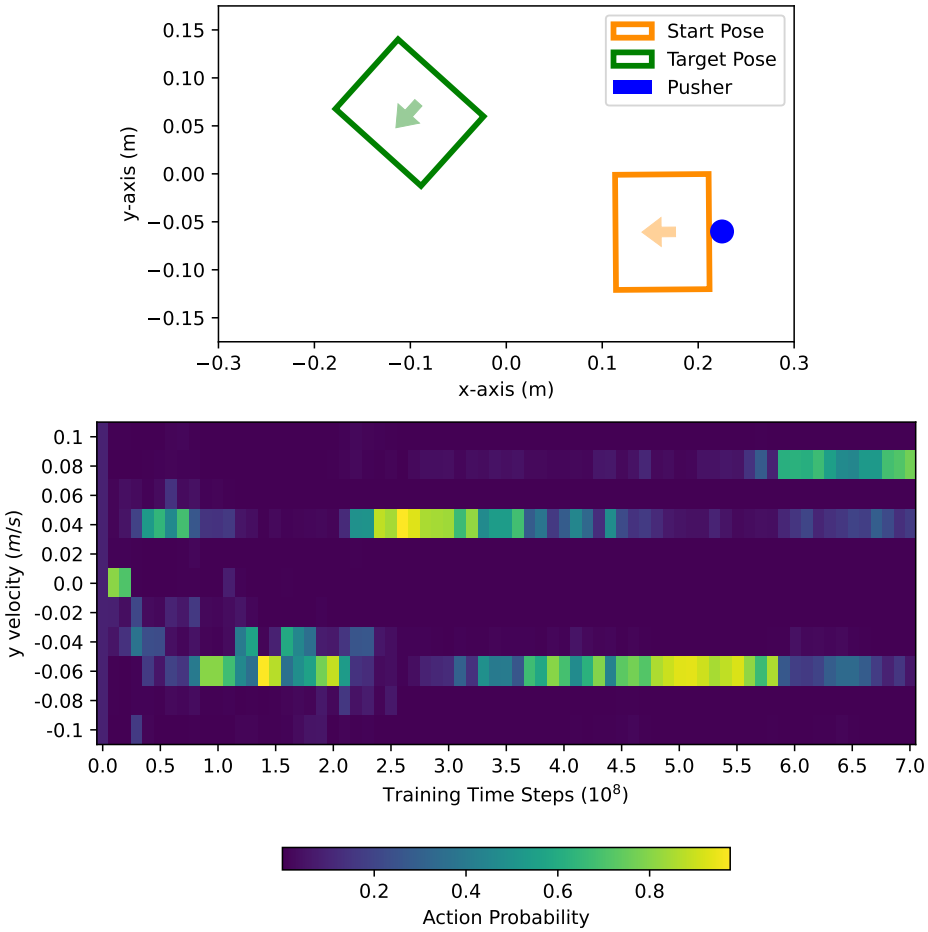


Figure 5.4: Evolution during training of the categorical exploration distribution for the pusher velocity in the y axis $(v_{y,p})$ when the policy receives the observation corresponding to the environment state above.

so they can be performed in a smooth trajectory without the pusher breaking contact with the box. Therefore, the reason why the policies with Gaussian exploration do not manage to progress further in the curriculum could be that they struggle with the tasks that require phase switching since their exploration strategy is not able to effectively capture this additional modality.

5.1.5 Multimodal Exploration

In this experiment, we investigate whether our proposed categorical exploration approach indeed leads to multimodal strategies. We examine the evolution during training, under various environment states, of the categorical exploration distribution for the action $v_{y,p}$ generated by the PPO (LSTM + Categorical) policy. We find that the exploration distribution is multimodal in many environment states. Figure 5.4 shows the results for one of these environment states.

The exploration distribution broadly contains two modes which correspond to upward $(v_{y,p} > 0)$ and downward $(v_{y,p} < 0)$ motions of the pusher. Considering the environment

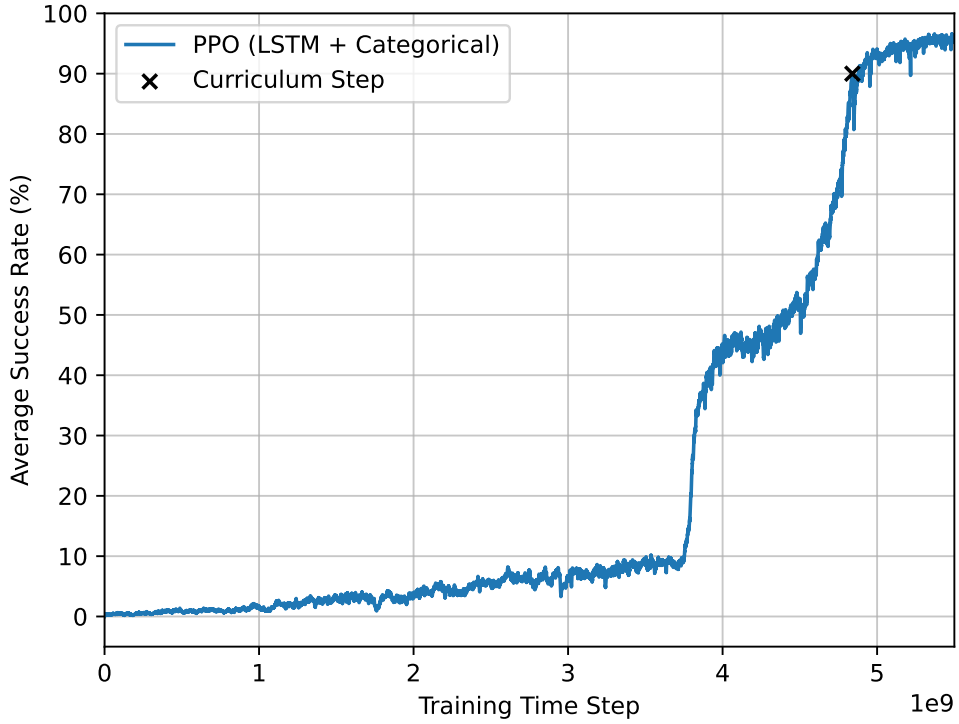


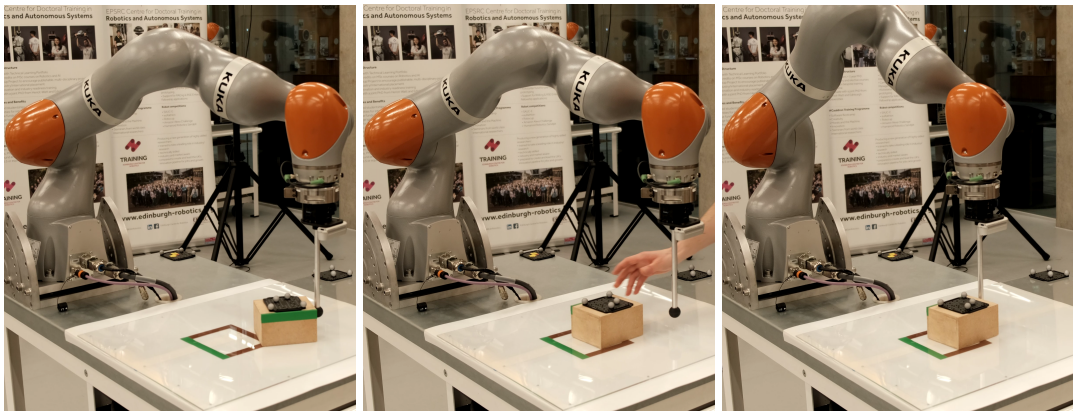
Figure 5.5: Training performance of the PPO (LSTM + Categorical) policy on a planar pushing environment with two pushers.

state in which these distributions where generated, this is reasonable since there are two primary ways of solving the task, each requiring different contact modes. The first option consists of a single continuous trajectory, without the pusher breaking contact with the box. This option first requires the pusher to slide left $(v_{y,p} < 0)$, relative to the box, in order to rotate the box towards the target. The second option consists of a phase switch. This option first requires the pusher to slide right $(v_{y,p} > 0)$, relative to the box, in order to align the orientation of the box with the target orientation, and then switch to the left phase of the box in order to push it towards the target.

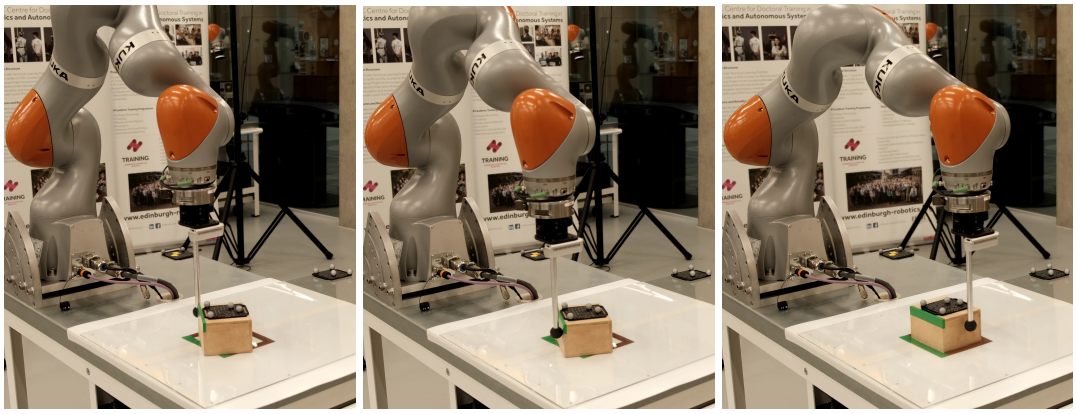
Overall, as hypothesized, the policy leverages our proposed categorical exploration approach to establish multimodal exploration strategies. We believe that this enables the policy to explore different possible contact modes concurrently during training.

5.1.6 Scalability with Two Pushers

To evaluate the scalability of our framework, we train the PPO (LSTM + Categorical) policy on a planar pushing task with two pushers. The results are shown in Figure 5.5. We use the same curriculum as described in Section 4.1.4 such that the success thresholds are $T_{x,y}^{(1)}, T_{\theta}^{(1)}$ in the beginning and, if the policy reaches 90% success rate, these are reduced to $T_{x,y}^{(2)}, T_{\theta}^{(2)}$. We also use dynamics randomization, observation noise, and synthetic disturbances during training as in the planar pushing task with one pusher. Since this task is significantly more complex, we increase the size of the LSTM architecture for the policy and value functions. The shape remains the same as in



(a) (b) (c)



(d) (e) (f)

Figure 5.6: Key frames of the KUKA iiwa robot pushing the box to a target pose. (a) shows the starting configuration, a large disturbance is applied in (b), and (c)-(f) exhibit the RL policy recovering from the disturbance and reaching the goal.

Figure 4.2; however, we use linear layers of size 256 and LSTM layers of size 512.

We find that our framework scales well to the planar pushing task for arbitrary initial and target object poses when using two pushers. The policy reaches the curriculum step and achieves over 96% average success rate under the reduced success thresholds. Learning this task is significantly more complex than the task with one pusher due to the increased dimensionality of the problem. In particular, the state-action space is significantly larger, so the policy must explore a wider range of possibilities in order to identify effective pushing behaviors. Indeed, we observe that the convergence of the policy is slower than in the task with one pusher.

5.2 Real Robot Experiments

We investigate the performance of the PPO (LSTM + Categorical) policy, after the training process in Section 5.1.1, on our physical planar pushing hardware set-up. Figure 5.6 shows a sequence of key frames of the robot pushing the box to a target

pose and recovering from an external disturbance. Furthermore, the supplemental video clearly demonstrates the resulting behavior of the policy. We find that the policy translates well to the real world and is able to effectively cope with the dynamics of the new environment. Additionally, the policy is robust to large external disturbances and changes in the target pose. Figure 5.7 shows a sample trajectory generated by the robot. The policy manages to remain within the workspace boundaries and generates a significantly smooth and efficient trajectory towards the target box pose.

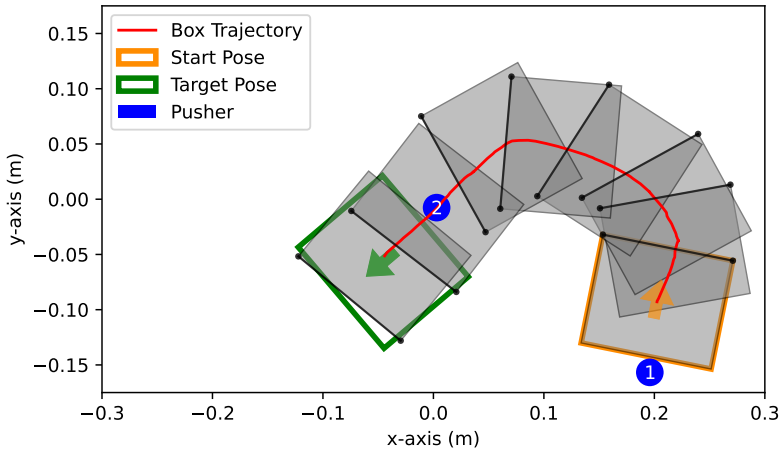


Figure 5.7: Trajectory generated by the PPO (LSTM + Categorical) policy in the real robot.

We also obtain statistics for the average success rate and time to target of the policy on the real robot. We use the reduced success thresholds $T_{x,y}^{(2)}, T_{\theta}^{(2)}$ and enforce a time limit of 30 seconds to complete the task. To collect the data, we generate 5 random target box poses and, for each target pose, we run the policy starting from 15 random initial configurations of the box and pusher. The target box poses and initial configurations were generated using the same criteria as in our simulated planar pushing environment. Overall, this results in 75 planar pushing tasks with arbitrary initial and target object poses. The policy achieves an average success rate of 97.3 ± 1.9 % (SE) and an average time to target of 6.5 ± 0.3 sec (SE).

The policy manages to consistently solve the planar pushing task for arbitrary initial and target object poses in the physical hardware. It achieves a significantly high success rate while preserving a low time to target, which indicates that the policy does not require multiple corrections to complete the task. Additionally, recall that the position and orientation success thresholds are 0.75 cm and 0.17 rad, so the policy demonstrates highly accurate pushing motions. Overall, we find that our framework leads to learned policies with a good transferability to the physical hardware.

While previous RL methods for planar pushing disregard the orientation of the manipulated object [11]–[14], we manage to learn policies that incorporate object orientation, and therefore realize more sophisticated pushing motions. Our policies adapt well to the unfamiliar dynamics of the physical robot set-up, successfully recover from external disturbances, and exhibit significantly accurate and smooth planar pushing motions.

Chapter 6

Conclusions

6.1 Summary and Contributions

The primary goal of this project was to investigate the application of RL methods to the planar pushing task. More specifically, we were interested in learning RL policies that could control the position and orientation of the manipulated object, previously unaddressed by the RL literature [11]–[14]. This led us to propose a multimodal exploration approach, through categorical distributions on a discrete action space, which enables us to learn planar pushing RL policies for arbitrary initial and target object poses, i.e. different positions and orientations. Our experiments show that even while maintaining the underlying algorithm, function approximation architectures, and hyperparameters constant, the previously used exploration approach leads to failure in the learning task, while our proposed exploration approach enables the policy to learn the task with significant accuracy and success rate.

In addition, we analysed the pushing trajectories as well as the behavior under different levels of correlated and uncorrelated observation noise of a planar pushing RL policy learned through the proposed framework. Our experiments demonstrate that the learned policy is robust to observation noise and produces smooth and efficient trajectories. Furthermore, we showed that our framework scales well when increasing the number of pushers.

Finally, we assembled a planar pushing physical robot set-up and developed a robot controller to deploy the RL policies. This enabled us to validate that our proposed framework leads to learned policies with good transferability to the physical hardware, achieving high success rate, smooth pushing trajectories, and small target error.

6.2 Discussion and Future Work

One of the key realizations in this work was that, when attempting to learn planar pushing RL policies for the case of arbitrary initial and target object poses, the use of a multivariate Gaussian with diagonal covariance for exploration, as per previous literature [11]–[14], would lead to the RL policies failing to converge. Borrowing

the insight from the model-based literature [4], [10], that planar pushing has hybrid-dynamics reflected in a set of different contact modes that constraint the control actions, we hypothesised that we can reason about planar pushing as a multimodal control problem. Therefore, we proposed describing the action space through categorical distributions to capture the multimodal nature of the problem, potentially leading to more effective exploration of different contact modes during training. We have shown that indeed, during training, the categorical action distributions exhibit multimodal exploration strategies.

Although the proposed framework enabled us to learn RL policies that incorporate object orientation, it is important to address several notable limitations. To begin with, our policies require significant computational resources to learn the task. In particular, the best policy required around 1 day of training to reach 90% success rate in the task with one pusher, and around 8 days to reach 90% success rate in the task with two pushers. Furthermore, if we wished to change the set-up of the planar pushing environment, for instance in order to use a different object, we would need to develop a simulation environment with the new set-up and re-train the policies. Future work could leverage recent advances in distributed training of RL algorithms in order to improve the training time of the planar pushing policies [21], [31].

In terms of scalability, even though we showed that our framework manages to learn the planar pushing task with two pushers in simulation, we were unable to validate its transferability to the physical hardware. Additionally, we constrained the control output of the RL policies to the 2D plane, which limits the range of behaviors that can be realized by the policy. In future work, we intend to evaluate the scalability of our framework when moving beyond planar pushing to more complex nonprehensile manipulation tasks requiring policy actions in the 3D world. In this work, we used a box as the manipulated object, which has a relatively simple geometry. Furthermore, as previously discussed, changing the manipulated object necessitates re-training the policies. Future work could explore whether our framework can be scaled to manipulate objects requiring more complex contact surface selection as well as previously unseen object geometries. The latter could potentially be achieved by training the policies with a diverse range of objects.

Our proposed approach for multimodal exploration requires the action space to be discretized. However, this restricts the motion of the robot and thereby hinders the precision of the pushing trajectories. Therefore, we believe that investigating multimodal exploration strategies capable of preserving the continuity of the action space is an interesting avenue for future work. In particular, we believe that Gaussian mixture models could be leveraged to achieve this.

Finally, we used a motion capture system for our experiments in the physical hardware, which provides significantly accurate readings of the environment state. In future work, we intend to use onboard vision-based perception, eliminating the need for a motion capture system. However, this would significantly exacerbate the noise and uncertainty in the policy observations, thereby hindering the dexterity of the pushing motions realized by the RL policies. We believe this could be largely mitigated by integrating force feedback, measured at the robot end-effector, into the RL policies.

Bibliography

[1] M. T. Mason, "Mechanics and planning of manipulator pushing operations," *The International Journal of Robotics Research*, vol. 5, no. 3, pp. 53–71, 1986. DOI: 10.1177/027836498600500303.

[2] M. T. Mason, "Progress in nonprehensile manipulation," *The International Journal of Robotics Research*, vol. 18, no. 11, pp. 1129–1141, 1999. DOI: 10.1177/02783649922067762.

[3] K. M. Lynch and M. T. Mason, "Dynamic nonprehensile manipulation: Controllability, planning, and experiments," *The International Journal of Robotics Research*, vol. 18, no. 1, pp. 64–92, 1999. DOI: 10.1177/027836499901800105.

[4] F. R. Hogan and A. Rodriguez, "Reactive planar non-prehensile manipulation with hybrid model predictive control," *The International Journal of Robotics Research*, vol. 39, no. 7, pp. 755–773, 2020. DOI: 10.1177/0278364920913938.

[5] M. S. Branicky, "Introduction to hybrid systems," in *Handbook of Networked and Embedded Control Systems*, D. Hristu-Varsakelis and W. S. Levine, Eds. Boston, MA: Birkhäuser Boston, 2005, pp. 91–116, ISBN: 978-0-8176-4404-8. DOI: 10.1007/0-8176-4404-0_5.

[6] T. Xue, H. Girgin, T. S. Lembono, and S. Calinon, "Demonstration-guided optimal control for long-term non-prehensile planar manipulation," *arXiv preprint arXiv:2212.12814*, 2022.

[7] J. Zhou, R. Paolini, A. M. Johnson, J. A. Bagnell, and M. T. Mason, "A probabilistic planning framework for planar grasping under uncertainty," *IEEE Robotics and Automation Letters*, vol. 2, no. 4, pp. 2111–2118, 2017. DOI: 10.1109/LRA.2017.2720845.

[8] M. Bauza and A. Rodriguez, "A probabilistic data-driven model for planar pushing," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 3008–3015. DOI: 10.1109/ICRA.2017.7989345.

[9] S. Goyal, A. Ruina, and J. Papadopoulos, "Limit surface and moment function descriptions of planar sliding," in *International Conference on Robotics and Automation*, vol. 2, 1989, pp. 794–799. DOI: 10.1109/ROBOT.1989.100081.

[10] J. Moura, T. Stouraitis, and S. Vijayakumar, "Non-prehensile planar manipulation via trajectory optimization with complementarity constraints," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2022, pp. 970–976. DOI: 10.1109/ICRA46639.2022.9811942.

[11] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in *IEEE International*

Conference on Robotics and Automation (ICRA), 2018, pp. 3803–3810. DOI: 10.1109/ICRA.2018.8460528.

[12] R. Jeong, J. Kay, F. Romano, T. Lampe, T. Rothorl, A. Abdolmaleki, T. Erez, Y. Tassa, and F. Nori, "Modelling generalized forces with reinforcement learning for sim-to-real transfer," *arXiv preprint arXiv:1910.09471*, 2019.

[13] L. Cong, H. Liang, P. Ruppel, Y. Shi, M. Görner, N. Hendrich, and J. Zhang, "Reinforcement learning with vision-proprioception model for robot planar pushing," *Frontiers in Neurorobotics*, vol. 16, 2022, ISSN: 1662-5218. DOI: 10.3389/fnbot.2022.829437.

[14] K. Lowrey, S. Kolev, J. Dao, A. Rajeswaran, and E. Todorov, "Reinforcement learning for non-prehensile manipulation: Transfer from simulation to physical system," in *IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR)*, 2018, pp. 35–42. DOI: 10.1109/SIMPAR.2018.8376268.

[15] J. Del Aguila Ferrandis, J. Moura, and S. Vijayakumar, "Nonprehensile planar manipulation through reinforcement learning with multimodal categorical exploration," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, [Under review], 2023.

[16] J. Zhou, R. Paolini, J. A. Bagnell, and M. T. Mason, "A convex polynomial force-motion model for planar sliding: Identification and application," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 372–377. DOI: 10.1109/ICRA.2016.7487155.

[17] M. T. Mason, *Mechanics of Robotic Manipulation*. The MIT Press, Jun. 2001, ISBN: 9780262256629. DOI: 10.7551/mitpress/4527.001.0001. [Online]. Available: https://doi.org/10.7551/mitpress/4527.001.0001.

[18] J. B. Rawlings, D. Q. Mayne, and M. Diehl, *Model predictive control: theory, computation, and design*. Nob Hill Publishing Madison, WI, 2017, vol. 2.

[19] E. Dantec, R. Budhiraja, A. Roig, T. Lembono, G. Saurel, O. Stasse, P. Fernbach, S. Tonneau, S. Vijayakumar, S. Calinon, *et al.*, "Whole body model predictive control with a memory of motion: Experiments on a torque-controlled talos," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2021, pp. 8202–8208.

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

[21] OpenAI, M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, *et al.*, "Learning dexterous in-hand manipulation," *The International Journal of Robotics Research*, vol. 39, no. 1, pp. 3–20, 2020.

[22] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning quadrupedal locomotion over challenging terrain," *Science Robotics*, vol. 5, no. 47, eabc5986, 2020. DOI: 10.1126/scirobotics.abc5986.

[23] 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, eabk2822, 2022.

[24] N. Heess, J. J. Hunt, T. P. Lillicrap, and D. Silver, "Memory-based control with recurrent neural networks," *arXiv preprint arXiv:1512.04455*, 2015.

[25] A. Abdolmaleki, J. T. Springenberg, Y. Tassa, R. Munos, N. Heess, and M. Riedmiller, "Maximum a posteriori policy optimisation," *arXiv preprint arXiv: 1806.06920*, 2018.

[26] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *Proceedings of the 35th International Conference on Machine Learning*, vol. 80, PMLR, 2018, pp. 1861–1870.

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

[28] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021.

[29] P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu, and P. Zhokhov, *Openai baselines*, https://github.com/openai/baselines, 2017.

[30] J. Achiam, "Spinning Up in Deep Reinforcement Learning," 2018.

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

[32] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[33] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

[34] N. Thuerey, P. Holl, M. Mueller, P. Schnell, F. Trost, and K. Um, *Physics-based Deep Learning*. 2021.

[35] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[36] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[37] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[38] Y. Tang and S. Agrawal, "Discretizing continuous action space for on-policy optimization," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 4, pp. 5981–5988, 2020. DOI: 10.1609/aaai.v34i04.6059.

[39] P. Soviany, R. T. Ionescu, P. Rota, and N. Sebe, "Curriculum learning: A survey," *International Journal of Computer Vision*, vol. 130, no. 6, pp. 1526–1565, 2022.

[40] E. Coumans and Y. Bai, *Pybullet, a python module for physics simulation for games, robotics and machine learning*, http://pybullet.org, 2021.

[41] 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," *arXiv preprint arXiv:1804.10332*, 2018.

[42] X. B. Peng, E. Coumans, T. Zhang, T.-W. Lee, J. Tan, and S. Levine, "Learning agile robotic locomotion skills by imitating animals," *arXiv preprint arXiv: 2004.00784*, 2020.

[43] C. E. Mower, T. Stouraitis, J. Moura, C. Rauch, L. Yan, N. Z. Behabadi, M. Gienger, T. Vercauteren, C. Bergeles, and S. Vijayakumar, "Ros-pybullet interface: A framework for reliable contact simulation and human-robot interaction," in *Conference on Robot Learning (CoRL)*, 2022.

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

[45] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., Curran Associates, Inc., 2019, pp. 8024–8035.

[46] S. Rammohan, S. Yu, B. He, E. Hsiung, E. Rosen, S. Tellex, and G. Konidaris, "Value-based reinforcement learning for continuous control robotic manipulation in multi-task sparse reward settings," *arXiv preprint arXiv:2107.13356*, 2021.

[47] Z. Mandi, P. Abbeel, and S. James, "On the effectiveness of fine-tuning versus meta-reinforcement learning," *arXiv preprint arXiv:2206.03271*, 2022.

[48] M. Andrychowicz, A. Raichuk, P. Stańczyk, M. Orsini, S. Girgin, R. Marinier, L. Hussenot, M. Geist, O. Pietquin, M. Michalski, *et al.*, "What matters in on-policy reinforcement learning? a large-scale empirical study," *arXiv preprint arXiv:2006.05990*, 2020.

[49] J. Hilton, K. Cobbe, and J. Schulman, "Batch size-invariance for policy optimization," *Advances in Neural Information Processing Systems*, vol. 35, pp. 17 086–17 098, 2022.

[50] S. Kullback and R. A. Leibler, "On information and sufficiency," *The annals of mathematical statistics*, vol. 22, no. 1, pp. 79–86, 1951.

[51] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng, *et al.*, "Ros: An open-source robot operating system," in *ICRA workshop on open source software*, Kobe, Japan, vol. 3, 2009, p. 5.

[52] C. E. Mower, J. Moura, N. Z. Behabadi, S. Vijayakumar, T. Vercauteren, and C. Bergeles, "Optas: An optimization-based task specification library for trajectory optimization and model predictive control," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2023.