Deep Reinforcement Learning with Dense Reward Shaping and Domain Randomization for Sim-to-Real Transfer in Robotic Push Manipulation

by

Dalel Abekenov

Submitted to the Department of Robotics
in partial fulfillment of the requirements for the degree of

Master of Science in Robotics

at the

NAZARBAYEV UNIVERSITY

Apr 2026

© Nazarbayev University 2026. All rights reserved.

Author .....
Department of Robotics
Apr 17, 2026

Certified by .....
Almas Shintemirov
Associate Professor
Thesis Supervisor

Accepted by .....
Elizabeth Arkhangelsky
Dean, School of Engineering and Digital Sciences

Deep Reinforcement Learning with Dense Reward Shaping and Domain Randomization for Sim-to-Real Transfer in Robotic Push Manipulation

by

Dalel Abekenov

Submitted to the Department of Robotics
on Apr 17, 2026, in partial fulfillment of the
requirements for the degree of
Master of Science in Robotics

Abstract

Deep reinforcement learning represents a promising approach for teaching robot manipulation skills through simulation training, however the transfer of learned policies to the real robot remains a major challenge. This thesis is focused on investigation of sim-to-real transfer for planar push manipulation using the Soft Actor-Critic policy trained in MuJoCo simulation for a Franka Emika Panda robot. The work is organized as a three-condition ablation study, examining individual effects of reward function design and domain randomization, followed by zero-shot deployment of the candidate policy on a physical hardware.

The ablation provides a comparison between sparse reward, dense progress-based reward, and dense reward with domain randomization. Each of these conditions was trained over 2,000,000 timesteps and evaluated over 100 episodes. This result is consistent with prior findings that sparse rewards can support policy learning in combination with efficient off-policy methods and large replay buffers for constrained manipulation tasks [16]: sparse and dense reward functions achieved the same success rate in simulation (79%), with differences only in training convergence speed. Adding domain randomization strategy leads to a drop of 12 percentage points to 67%, however produces the only policy suitable for real robot deployment.

Real robot evaluation over 20 episodes demonstrates that policy with domain randomization achieved 80% success rate in straight-line configurations, which exceeds its own simulation result. It confirms that the sim-to-real gap is effectively bridged for aligned push configurations. Overall success rate over all types of configurations is 50%, in which the main limitation is restricted policy's ability to generalize laterally for diagonal push outside of training distribution.

The key finding in this thesis is that success rate in simulation is an unreliable indicator for performance in the real-world experiment. The policy with the lowest success rate in simulation was the only candidate for real robot deployment, and it even showed better performance compared to simulation. This result shows that real

robot evaluation is important and should be used along with simulation benchmarks in robotic research. Also, it provides empirical evidence of the trade-off between specialization and generalization inherent in domain randomization for contact-rich manipulation tasks.

Thesis Supervisor: Almas Shintemirov
Title: Associate Professor

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation and Problem Statement

One of the most fundamental and long-standing challenges in robotics is the ability to manipulate objects in unstructured environments. Although industrial robots succeeded at completing repetitive and accurately programmed tasks, the real world deployment, where robots are expected to perform tasks in environments with changing or uncertain object positions, surface properties and contact dynamics - remains a relevant research gap [21]. Connecting the gap between controlled laboratory conditions and the challenge of real robot deployment demands not only capable hardware, but also learning algorithms that are able to generalize across variability and adapt to new conditions without manual programming.

Over the past few years, deep reinforcement learning (RL) has appeared as a promising approach for training robot manipulation policies from direct interaction with the environment, without deliberate programming of each movement and contact event [21], [12]. In this framework, manipulation is represented as a sequential decision-making problem. The agent receives a current observation of the state of the world, completes an action, and receives the reward signals that correspond with progress to the goal. Deep RL allows robots to learn complex behaviors by this process of trial and error in simulation. A crucial benefit of this method is the opportunity to perform training in simulation instead of real robot setup. It prevents high cost,

time requirements, and possible risks related to the direct training on real hardware. In real-world settings there are factors such as mechanical wear, safety limitations, and irreversible physical contacts which significantly reduces the number of training attempts making large-scale learning even more difficult.

However, a remarkable challenge appears when models trained in simulation are being applied on physical robotic setup: the sim-to-real gap. Despite simulation environments' accuracy they make simplified assumptions about factors like contact dynamics, friction, object inertia, sensor noise, and actuator response. A policy that performs successfully in simulation along with the artifacts and quirks, can experience total failure in real robot experiment, where those idealized conditions do not exist [2], [22]. Therefore, solving the sim-to-real gap and keeping the efficiency and scalability advantages of simulation-based training remains one of the important and relevant problems in robot learning.

This thesis explores these difficulties in the framework of planar push manipulation, where a controlling robotic arm is expected to push the rigid object into a specific goal position. Push manipulation, also known as non-prehensile manipulation is a fundamental robotic skill which can be usefully applied in various areas including warehouse automation, assembly line operations, and assistive robotics [5], [7]. In comparison with grasping, pushing does not need precise and stable grip, which makes it suitable for large or flat objects that cannot be grasped. Despite the fact that pushing may seem a simple task, it involves contact-rich, underactuated dynamics that are difficult to model analytically. Also, the task is highly sensitive to external factors like surface friction, object mass, and other parameters that are different between simulation and real world.

This work is specifically focused on training of Soft Actor-Critic (SAC) [13] policy in MuJoCo simulation for 7-DOF Franka Emika Panda robotic arm and following deployment of the trained policy to the physical robot. The research is represented as a three-condition ablation study that explores how the design of reward function and use of domain randomization influence the performance in both simulation and physical experiment. The obtained results are empirical evidence to several open

questions, considering the comparison of sparse and dense reward functions, and domain randomization effect on behaviour of trained policy.

1.2 Robotic Push Manipulation: Scope and Relevance

Push manipulation or planar non-prehensile manipulation refers to the task where a robot moves an object through applying controlled contact force without setting stable grasp [10]. Because of mechanical simplicity, which demands only a rigid end-effector instead of stable gripper, it is effective for situations where speed, robustness, or object geometry preclude reliable grasping. At the same time, it is difficult to define accurately the dynamics of sliding objects that are controlled by friction models. Furthermore, correlation between end-effector's movement and final motion of the object is nonlinear, it depends on history and is sensitive to initial conditions [5].

Traditional methods of performing push manipulation mostly relied on analytical models of contact mechanics and friction for planning pushing trajectories [10]. This type of model-based approaches can be successful and precise with awareness of system parameters, however they can become fragile when facing the modelling errors such as a layer of dust on the table, slight changing in object's weight or robot end-effector's misalignment, which can cause failure of planned trajectory. In contrast, learning-based methods obtain policies that can adapt to natural variability present in training data. It happens due to relying on learning from data and experience instead of precise physical models. This makes them a decent alternative for situations where physics models are difficult to obtain or unreliable [3], [7].

Among all learning-based methods, deep reinforcement learning has demonstrated strong potential for pushing manipulation. RL agents can be fully trained in simulation with physics engines that approximate contact dynamics, and studies reported that trained models can be successfully transferred to the real robots for similar pushing manipulation tasks [1], [15]. The key idea is that by training policy in a

wide range of conditions, which can be achieved by using domain randomization, the learned behaviour can remain effective and functional across different environments in the real world, even if no single simulated condition matches the physical system exactly [22].

The push tasks considered in this work represents a wide range of goal-directed manipulation challenges. A Franka Panda robot, working at the fixed table height must push cylindrical puck from random initial position to the random goal position within workspace 40 x 60cm. Task success is defined by puck staging in the goal position for 10 consecutive timesteps within 5 cm, meaning that the robot should not only make contact with the puck, but also push it accurately and maintain precise positioning. This task setup embraces key aspects of push manipulation, while remaining manageable for systematic experimental study.

1.3 Sim-to-Real Transfer Challenge

The main problem of sim-to-real transfer comes from the difference between the environment the model was trained in, and the physical environment where it is deployed [2]. Even highly detailed and accurate simulations cannot reproduce the continuous contact dynamics of a real robot interacting with a real surface. In simulation, friction coefficients are usually considered as constant fixed values, while in reality friction is affected by many factors like surface microstructure, contact area, velocity, and temperature. Also, object inertia is exactly known in simulation, however actual mass of an object can be different from its nominal value. Sensor noise can be enabled in simulation with Gaussian perturbations, but real noise may have complex, structured patterns that do not correspond with this simplified model.

Particularly for push manipulation, the sim-to-real gap is remarkably important because of task dynamics being determined by contact interactions [5]. The trajectory that the object follows is severely influenced by friction between the object and table surface, the impulsive forces created by the end-effector touching the object, and object's and surface's compliance. Slight changings in those factors can significantly

affect movement of an object, which means that a policy that succeeded in simulation can fail or show worse performance in the real experiment.

In order to address the sim-to-real gap, two strategies have been proposed: domain randomization and system identification. The goal of system identification is to measure real-world parameters as accurately as possible, and make simulation match them closely, and as a result reduce the gap through increasing accuracy [2]. While domain randomization is focused on the opposite aspect. It intentionally brings variety in simulation parameters. For instance, it randomizes properties as mass, friction, sensor noise, and other factors across training episodes. Therefore, policy must learn the behaviours that are robust to these changes, rather than behaviours that are effective for a precisely determined environment [15], [22]. The key insight is that if policy achieves success across a broad range of conditions, then the real parameters will likely belong to this distribution, so the policy will be able to adapt and generalize.

This thesis implements structured domain randomization, which is focused on key sources of uncertainty for push manipulation; it randomizes object mass ($\pm 10\%$), surface friction ($\pm 15\%$), and observation noise (2mm standard Gaussian deviation). These ranges were considered because they do not make training unstable, at the same time they reflect real-world variation. The influence of this domain randomization on simulation and physical experiment is a key empirical question in this thesis.

The next important aspect of the sim-to-real gap is the perception pipeline. In simulation, positions of goal and object are received directly from the physics engine with ideal precision, while in real experiments this data comes from sensors. In this work it is done by Intel RealSense RGB-D camera that is mounted over the table surface, and a system of visual markers - AprilTag. The localization errors that are presented by the perception pipeline, and the way they interact with observation noise model used in training with domain randomization, is a serious engineering problem, which is regarded further in methodology.

1.4 Research Questions and Objectives

The work presented in this thesis revolves around three central research questions, each one is focused to address different aspects of the sim-to-real pipeline for push manipulation.

RQ1: Can a SAC policy trained in MuJoCo simulation successfully transfer to a physical Franka Panda robot for planar push manipulation?

This question considers general effectiveness and pragmatism of sim-to-real pipeline. Specifically it investigates if the combination of simulation-based training, domain randomization, and the perception and control interface regarded in this work can transfer policy to a real robot. The answer helps assess practical feasibility of this method for push manipulation and creates a baseline for further research.

RQ2: How do reward function design choices — specifically the use of sparse versus dense reward — affect learning efficiency and final policy performance?

The question discovers the effect of reward shaping on the policy that is learned. Sparse rewards produce clean and easily interpretable objective, however it can slow down and worsen learning through giving no gradient signal until the task is completed. Dense, progress-based rewards produce more continuous feedback during an episode, but it involves extra design choices, and creates potential for reward hacking, in which agents find a way to increase reward avoiding completing the task. This thesis presents ablation studies that demonstrated empirical evidence about this trade off for the specific case of push manipulation.

RQ3: Does domain randomization improve or degrade policy performance, and does this effect differ between simulation and real-robot evaluation?

This question considers correlation between simulation performance and real-world robustness. A spread assumption in domain randomization studies is that domain randomization enhances transfer to physical robot, while decreasing the simulation performance. This thesis provides empirical evidence to this trade off. The simulation results demonstrated Domain Randomization (DR) dropped success rate by 12% in simulation in comparison with baseline without domain randomization.

The ongoing real-robot experiment is aimed to find if this drop in simulation comes with improved transfer.

To address these questions, the following objectives were specified:

- Develop and apply custom MuJoCo environment for push manipulation with adjustable reward type and domain randomization
- Soft Actor-Critic policies trained in three conditions: sparse reward without DR, dense reward without DR, and dense reward with DR.
- Conduct evaluation of these three conditions with the following standard metrics: success rate, mean and median final object-to-goal distance, and episode efficiency.
- Create a sim-to-real pipeline with camera calibration, perception based on AprilTag system, and a real-robot control interface matching observation and action space.
- Transfer model with the best results and potential to the physical Franka Panda robot and define the results of performance and failures.

1.5 Thesis Contributions

The following are the specific contributions of this thesis:

Contribution 1: Empirical ablation study of reward design and domain randomization for push manipulation. This study conducted a three condition ablation study, where reward type and domain randomization were the only modified properties, while other hyperparameters, network architectures, and training budgets remained identical. The study faced non-obvious finding: sparse and dense rewards achieved the same success rate, whereas domain randomization led to drop of performance in simulation, although it is still expected to positively influence robustness in real experiments. These results assume that successful performance in simulation does not guarantee decent performance in deployment.

Contribution 2: Dense progress-based reward function for push manipulation. A developed reward function includes a combination of distance to the goal penalty, a per-step progress reward (scaled by a factor of 10), penalty for end-effector for being too far from the goal, and a terminal bonus for task completion. A key component is the progress term, it gives a reward for any decreasing the distance between object and goal each timestep. It provides a useful learning signal even when the object is far from the goal.

Contribution 3: Relative observation design for spatial generalization. The 14-dimensional observation is designed to include absolute position information and relative vectors like end-effector to object, object to goal. It enables policy to learn geometric relationships between different elements of the task, instead of simply memorizing absolute position in the workspace. This design is anticipated to positively influence generalization to new initial configurations.

Contribution 4: Conservative domain randomization targeting push-specific uncertainty. A proposed domain randomization is aimed on three parameters that are likely to be different in simulation and real environment. These are: object mass, surface friction, and observation noise, the randomization range is set to $\pm 10\%$ for mass, $\pm 15\%$ for friction, and 2 mm for observation noise, they are chosen to apply realistic variability, instead of arbitrarily large values.

Contribution 5: End-to-end sim-to-real pipeline with zero-shot policy transfer. The complete pipeline is used for a deployment to a real Franka Panda robot. This involves design of MuJoCo environment, policy training, sensor perception based on the AprilTag visual system, calibration of the camera, and a real robot interface (`real_robot_env.py`) that provides the same observation and action space as the simulation environment. It allows transfer policy as a zero-shot, without any additional tunings.

1.6 Thesis Organization

The remainder of this thesis is organized the following way, Chapter 2 reviews relevant literature on deep reinforcement learning for robotic manipulation, reward shaping, domain randomization, the SAC algorithm, and push manipulation. It demonstrates the gaps that motivate this work. Chapter 3 gives a description of methodology, including task definition, environment design, reward formulation, SAC training procedure, DR strategy, and the full sim-to-real pipeline. Chapter 4 provides results of experiments of the ablation study, including training curves, quantitative evaluation metrics, failure mode analysis, and the results of real robot deployment. Chapter 5 considers interpretations of obtained results and findings, discussing their potential roles for the development of sim-to-real pipeline, and appropriate current limitations of this approach. Chapter 6 concludes the contribution of this thesis and highlights further directions.

Chapter 2

Related Work

This chapter examines existing literature related to the appropriate key research areas relevant to this thesis. Section 2.1 provides an overview of deep reinforcement learning applied to robotic manipulation. Section 2.2 is focused particularly on non-prehensile and push manipulation. Section 2.3 reviews the development of reward function, including debate between sparse and dense reward function. Section 2.4 considers Hindsight Experience Replay as a promising alternative approach for addressing challenges of sparse reward. Section 2.5 embraces the Soft Actor-Critic (SAC) algorithm and its abilities for continuous control in robotics. Section 2.6 covers the question of sim-to-real deployment and role of domain randomization. Section 2.7 regards task-adaptive reinforcement learning, which served as initial motivation for this study. Section 2.8 is focused on gaps in reviewed literature that aimed to be addressed in this work.

2.1 Deep Reinforcement Learning for Robotic Manipulation

Reinforcement Learning provides a general framework for learning control policies by interacting with the environment. In this framework, an agent observes the current state of the environment, then it chooses an action based on a policy, and obtains

scalar reward signals, which presents how desirable the resulting state transitions are. The goal is to find a policy that maximizes the expected cumulative discounted reward over an episode. When this framework is combined with deep neural network function approximators, which is known as deep reinforcement learning, it has demonstrated ability to learn complex, high dimensional control behaviours that would be impossible to specify analytically [21].

The implementation of deep reinforcement learning in robotics has a large and growing body of research. Early works demonstrated that model free RL algorithms could learn manipulation skills directly from raw sensory observation, like learning to grasp objects from image input with policy gradient methods [21]. Further studies extended these results to more complex tasks such pushing, pick-and-place, sliding and in-hand reorientation. It established deep RL as a practically working approach for obtaining manipulation skills by interacting with an environment [15], [16].

A key finding was that policies trained solely in simulation, could be transferred to real robot without fine-tuning of physical setup. This zero-shot sim-to-real transfer, although not guaranteed for all tasks, has been successfully demonstrated for a variety of tasks when combined with certain techniques that help connect simulation and real experiment dynamics [15], [22]. The opportunity to perform large scale trainings with generation of millions episodes without causing robot wear or safety risks has made sim-to-real transfer a main focus of research among the robotics community.

Recent literature describes numerous examples of deep RL implemented for robotic manipulation that are relevant for this work. Shahid et al. [3] demonstrated use of deep RL for continuous control over robotic grasping. It showed that off-policy can learn stable grasping behaviour in simulation with sensible sample efficiency. Li and Wang [8] studied model free RL for robot arm control based in simulation, which showed successful results in learning of reaching and positioning tasks using standard policy gradient methods. Liu et al. [5] investigated vision-based RL for dynamic object manipulation; they integrated camera input directly into policy observation space in order to track the objects during manipulation. Avhad et al. [7] presented an adaptive deep RL method for manipulation in dynamic environments, showing that

policies can be trained for handling variability in placement of objects and workspace condition without additional retraining.

More recent studies expanded deep RL to the progressively complex manipulation tasks. Bai et al. [6] applies deep RL to the demanding problem of robotic whip targeting, demonstrating that policy can be learned for highly-dynamic and contact rich tasks. Elsaman et al. [10] studied deep RL in continuum robot manipulation, where a robot's structure, which is deformable and continuous, adds complexity not present in rigid manipulators. Vadlamudi and Lakshmi [9] investigated deep RL in a task, where the robotic arm is balancing a ball, demonstrating stable policy learning, which requires maintaining sustained contact instead of single push. A common observation among these studies is that off-policies, which can reuse previous experience from replay buffer, are more sample-efficient compared to on-policies. It makes them a more attractive choice for tasks where real-world data collection is expensive or time demanding [12].

The wider examination of reinforcement learning in robotic manipulation and automation is brought by Farooq and Iqbal [12]. They review reinforcement learning approaches for optimization in different fields such as manufacturing, logistics and process control. The study which is focused specifically on the design of RL for mobile manipulators is provided by Kim et al. [11]. Their work uses the Robo-gym framework and shows the importance of well-developed simulation infrastructures in reproducible learning research. The work in this thesis builds based on these ideas through creating a custom MuJoCo-based environment, which is created particularly for planar push manipulation. The environment includes adjustable reward function and domain randomization settings, which enables controlled experimental comparison.

2.2 Non-prehensile and Push Manipulation

Non-prehensile manipulation is a group of tasks in robotic manipulation where objects are moved without involving grasping. In contrast with prehensile manipulation, where establishing stable grip is needed, in non-prehensile manipulation the robot

achieves the object's motion through pushing, sliding, rolling, or any other physical contact without enclosure of gripper [10]. This thesis considers a sub-class of non-prehensile manipulation - push manipulation. It includes direct application of contact forces to displace the object across the surface to the goal position. Although task definition may seem simple, it requires contact-rich and underactuated dynamics, which are challenging to control accurately. Moreover, object behaviour is highly sensitive to the surface conditions and object's properties, which can be different between environments or deployments [5].

Traditional approaches for push manipulation were mostly model-based. In order to describe quasi-static mechanics, pushing these methods used analytical frameworks, mostly relying on limit surface models of friction to predict the object's movement when specific contact force and direction were applied. Then planning algorithms searched for sequences of pushes that could move object from initial state to the target position. Despite the fact that these methods achieved certain success in laboratories with controlled environments, their dependence on exact knowledge of friction coefficients and object geometry caused issues in practical experiments. Slight differences between assumed model and actual conditions could lead to large prediction errors and constant task failures.

The move toward learning-based approaches happened because of realization that real-world push dynamics are too complex and strongly depend on environment conditions to be modeled solely by first-principle physics. Early works of learning-based methods used supervised learning for creating predictive models of object movement from collected data. Then these models were used in planning frameworks to verify how a robot has to push the object. In the more recent studies, deep RL is applied directly, which allows robots to learn pushing policies end-to-end. In this method, policy maps observation directly to motor commands and does not require explicit model of object's dynamics [3], [7].

Among the most relevant studies in this field, Wang et al. [19] presented a multi-stage reinforcement learning approach for non-prehensile manipulation. This approach divides complex manipulation tasks into a sequence of stages based on the

object's pose and changes in contact conditions. Their research showed that decomposition of non-prehensile into stages, where each one has its own RL policy, can increase learning efficiency and improve performance in the task, where flexible skill combination is required. In contrast, the approach from this work trains a single unified policy that controls the whole push episode. It simplifies learning problem, but it may reduce flexibility in more complex manipulation tasks.

The task used in this thesis, where the robot should push the puck to the goal position and keep it there in a certain radius for a particular number of timesteps, represents a common benchmark for manipulation learning literature. Similar task formulation can be seen in the OpenAI robotics environment [16], where Fetch robot completes tasks such as pushing, sliding, and pick-and-place. Related formulations also appear in many custom simulation environments created to study specific aspects of push manipulation. In the recent studies use of the Franka Panda robot is consistent [1], [3], [5]. It made the Panda widely used platform for tabletop manipulation research due to its 7-DOF reach, force-torque sensing, and compatibility with the FrankaPy and libfranka control interfaces.

A key design decision that makes this thesis different from other comparable studies is the use of 2D end-effector velocity action space instead of direct joint torque and control of joint angles. This method is applied via an inverse kinematics subloop running for ten sub-steps in every RL timestep. It reduces dimensionality of the learning problem from seven joint degrees of freedom to two planar translation dimensions. The obtained action space is easier to interpret and decreases policy need to learn low-level arm kinematics additionally to the push task. This design decision shows a large trend in manipulation RL, in which task-space control abstractions are implemented for accelerating the learning process through dividing high-level task reasoning from low-level motion control [7], [11].

2.3 Reward Function Design: Sparse versus Dense Rewards

One of the most important decisions in every reinforcement learning system is the design of the reward function. The reward function contains what behaviour the agent will learn, its structure is highly important for learning dynamics, the speed of convergence, and the characteristics of final policy [20]. Reward design is specifically complex for robotic manipulation, due to the fact that expected behavior, which is completing a task is mostly clearly observable at the end of the episode, the intermediate actions that lead to success are much more difficult to evaluate and reward step by step.

The difference between sparse and dense rewards is foundational for this challenge. Sparse reward provides binary or almost binary signal, usually fixed positive reward for task completion, and zero for opposite. Even though this type of reward is clear and easy to define, it does not provide a learning signal for exploratory experience, when agents fail to complete the task [16]. In comparison, denser reward provides a signal that varies continuously, it is mostly based on distance to the goal, progress toward the goal, or shaped combinations of task-relevant quantities. It gives informative feedback to the agent at each step, but it requires thorough design to escape unintended behaviours.

The theoretical bridge between reward shaping and optimal policy preservation was established by Ng et al. [20], who proved that potential-based reward shaping with adding a function of the form $F(s, s') = \gamma\Phi(s') - \Phi(s)$ to the initial reward, where Φ is potential function over states, changes the reward landscape without altering optimal policy. Their result gives the foundation of dense reward design. If the shaping term is derived from a valid potential function, like negative distance to the target, the shaped reward keeps the optimal policy of the original sparse reward problem and provides a denser learning signal. Practically, numerous dense reward functions in robotic manipulation approximate this potential-based form, although they do not strictly satisfy the theoretical conditions.

Advantages and disadvantages of sparse and dense rewards have been broadly discussed in manipulation learning literature. Supporters of sparse reward claim that they are easier to specify, have less chance of reward hacking, which means agents find ways to increase reward signals without actual task completion, and are more aligned with the task objective [16]. On the other hand, supporters of dense rewards claim that sparse rewards cause significant exploration difficulties in complex manipulation tasks. In this type of task agents have extremely small chances to find a valid solution from exploration. They argue that carefully designed shaping functions can remarkably increase the speed of learning, while allowing agents to develop high-quality task behaviour [3], [7].

Empirical results in literature are mixed and vary on the particular task. Sparse rewards have been shown to maintain successful policy learning in combination with efficient off-policy and large replay buffers in simple manipulation tasks with small state space, for example planar push on a clean surface [16]. While for relatively complex tasks with multi-step contact sequence or high-dimensional configuration spaces, sparse rewards typically make learning difficult. Dense rewards or curriculum-based methods are usually required in order to achieve reliable convergence [1], [15].

The dense reward function used in this thesis consists of 4 key components. First, there is a distance penalty based on the current distance between the goal and the object. Second, there is a per-step progress reward, which rewards any decrease in distance between the goal and the object at each timestep. It is scaled by a factor of ten, so it becomes a dominant part of the signal. Third, an end-effector proximity penalty that discourages the robot arm from moving far away from the object. Finally, a terminal success bonus that is given after task completion. The progress term is a key innovation in this reward design. Through rewarding reduction of object-to-goal distance at every timestep, it gives informative feedback even at the start of a task, when the object is far from the goal position, and distance penalty is large and almost constant. This formulation is related to potential-based shaping framework from Ng et al. [20], with the progress term corresponding to a difference in a goal-distance potential between consecutive timesteps.

2.4 Hindsight Experience Replay and the Sparse Reward Problem

Andrychowicz et al. [16] introduced a separate and highly influential approach to sparse reward problem in manipulation - Hindsight Experience Replay (HER). This approach proposes a conceptually simple solution: instead of making reward function denser, it changes the experience of replay, so that sparse rewards become more informative. The key point of this method is that even a failed episode with the agent failing to reach desired goal, can be reinterpreted after a while as a successful episode for a different goal, particularly the state that agent reached at the end of episode. By replaying transitions with this alternative goal assignment, HER generates a dense supply of positive reward signals from otherwise unrewarded experience, effectively circumventing the exploration problem without requiring any reward engineering.

HER was demonstrated on several robotic manipulation tasks that were implemented in OpenAI robotics environments such as object pushing, sliding and pick-and-place with a Fetch robot arm [16]. The results demonstrated that HER made to successfully train policy with sparse binary rewards in the task where policy trained without HER failed to converge. This established HER as a reliable and widely used method for goal-conditioned manipulation learning using sparse rewards. This method is especially effective when goal space closely matches with achievable state space. It means that the states that agent faces in exploration should be valid goals. This condition is satisfied in the planar push task, where the goal is 2D position in a continuous workspace.

Despite its effectiveness, HER introduces several extra design decisions. For instance, a method used to choose hindsight goals, like using the final state of an episode, randomly chosen future state or a different selection strategy can affect efficiency and stability of learning. Moreover, HER inherently frames the problem as a multi-goal learning setup. It means, policy must learn to reach any goal in its workspace, adjusting its behaviour on the current goal as an additional input. Although this flexibility is advantageous for tasks where the goal may change or need

to be specified dynamically, it increases complexity of the learning problem compared to formulation with fixed goal.

HER was deliberately excluded because one of the main research questions (RQ2) raises the issue of comparison between sparse and dense reward functions under the same conditions, which would be violated by adding HER to the sparse reward function. The key finding that sparse reward achieves final success comparable to dense reward without HER is the direct empirical contribution which shows that for the specific task considered in this thesis, the exploration problem related to sparse reward can still be addressed by SAC with use of sufficiently large replay buffer and parallel environment samplings.

2.5 Soft Actor-Critic and Off-Policy Reinforcement Learning

The Soft Actor-Critic algorithm proposed by Haarnoja et al. [13] and then extended with automatic temperature tuning [14] has been established as one of the most widely used model-free deep RL algorithms for continuous control tasks. SAC is an off-policy actor-critic algorithm that is based on the maximum entropy reinforcement learning framework. In this framework, the objective of policy optimization not only maximizes expected return it augments the standard expected return with an entropy bonus:

$$J(\pi) = \sum_{t=0}^{T} \mathbb{E}_{(s_t,a_t)\sim\rho_\pi}[r(s_t, a_t) + \alpha\mathcal{H}(\pi(\cdot|s_t))], \tag{2.1}$$

where $\mathcal{H}(\pi(\cdot|s_t)) = -\log\pi(a_t|s_t)$ represents an entropy of the policy state s_t, and $\alpha > 0$ is an entropy property that defines the strength of entropy term influence on the objective. This formulation keeps policy as random as it is possible for maximization entropy, and still achieving high reward, which means the agent is encouraged to explore different actions, and it prevents policy from premature convergence to deterministic suboptimal behaviors.

The actor-critic architecture used in SAC includes three key neural networks. It has a stochastic policy network – an actor that outputs distribution over actions in the current space. Additionally, there are two Q-functions networks – critics that evaluate expected return in the current policy. Using two critics along with taking the minimum of their estimates in target computation reduces the overestimation bias that affects single-actor algorithms, for example Deep Deterministic Policy Gradient (DDPG). Due to SAC being off-policy, it has the ability to reuse the experience from previous versions of policy from the replay buffer. This ability to reuse earlier data strongly improves sample efficiency compared to on-policy algorithms such as Proximal Policy Optimization (PPO) or Trust Region Policy Optimization (TRPO), which collect new data from every policy update.

SAC has several properties that make it appropriate for robotic manipulation. Firstly, entropy regularization provides automatic exploration; it does not need manual tuning of the exploration noise schedule. This is specifically effective in contact-rich tasks because of difficulty to predict the most effective type of exploration in advance. Secondly, because of being an off-policy, SAC can reuse past data from the replay buffer. In this work, it has one million transitions, it improves diversity and stability of gradient updates. Finally, the stochastic policy in SAC can naturally represent multimodal action distribution that is remarkably important when many different action strategies can achieve similar outcomes. These benefits have been demonstrated in multiple continuous control benchmarks, where SAC succeeded to achieve strong sample efficiency and high final performance compared to other approaches [13] [14].

Recent investigations in robotic manipulation have widely approved SAC as an effective training algorithm. Hlavaty and Kozakova [4] proposed a SAC-based control strategy for robotic system, it showed stable convergence and robust performance in conditions of real deployment. Cui et al. [1] used SAC as a key learning algorithm in their task-adaptive policy for dual-arm robot. They took advantage of its sample efficiency and stability in challenging multitask learning problem. Among the wider range of literature, SAC has been used for different tasks, such as grasp-

ing [3], pushing [7], dynamic object manipulation [5], and assembly tasks. In all cases SAC demonstrated competitive performance compared to other off-policy learning algorithms.

In this thesis SAC is implemented with the Stable-Baseline3 library [18], because it gives reliable and reproducible implementation with a consistent API. According to established practices for continuous robot manipulation, the key hyperparameters were chosen: learning rate of 3×10^{-4}, replay buffer with one million transitions, a batch size of 512, and network architecture with three layers, 256 units for each layer, which is deeper than default configuration of SB3 for better handling of contact-rich dynamics of push manipulation task. Eight parallel simulation environments through SubprocVecEnv were used for collecting the data. The number of gradient updates per environment step is aligned with the number of parallel environments for keeping a balanced ratio between training updates and collected experience. The target entropy is deliberately established as -2.0, because based on recommendation of Haarnoja et al. [14] it corresponds to the negative of the action space dimension.

2.6 Sim-to-Real Transfer and Domain Randomization

Deployment of policies that were trained in simulation to real robots is one the most central challenges in robot learning. The sim-to-real gap is the difference between simulation and real environment, it comes from several sources. It includes: inaccurate friction models, simplified contact mechanics, unmodeled actuator dynamics, sensor noise that does not match real hardware characteristics, and systematic biases in the physics engine's numerical integration [2]. This problem can be especially severe for policies that were trained with deep reinforcement learning. These policies can exploit the advantage of subtle patterns and regularities in simulations in order to achieve high results in performance. Even small differences between simulation and reality can lead to failure at transfer.

Zhao et al. [2] proposed a comprehensive survey of sim-to-real transfer methods in robotics; they organized existing methods in two categories. The first group includes methods that are aimed to reduce the sim-to-real gap by increasing accuracy of simulations (system identification and adaptive simulation). The second group has methods that produce policies which are robust for the gap (domain randomization and domain adaptation). System identification methods accurately estimate physical parameters and according to them update the simulator, which reduces the gap by increasing accuracy of the simulated environment. Although this method is effective when physical parameters are accessible and stable, it requires manual effort and repetition after changing setup or environment.

Domain randomization takes a completely different approach. Instead of matching simulation to real conditions, it randomly varies simulation's parameters across a range of plausible physical conditions during training. This approach makes policy learn behaviors that are robust across this distribution [15], [22]. The key insight, explained by Peng et al. [22], is that if the range of randomized conditions in simulation is large enough to embrace real-world conditions, then policy, which was trained in this distribution, will be able to adapt to the real system. Peng et al. [22] applied this approach in locomotion and robotic arm control task. They randomized dynamics parameters like ink masses, joint damping, and surface friction during training, so they could deploy policy to a real robot without real-world training data.

OpenAI et al. [15] provided a vivid demonstration of domain randomization effectiveness in dexterous manipulation. They trained policy to perform in-hand cube reorientation using Shadow Dexterous Hand in simulation. Multiple physical parameters were strongly randomized in training, for example friction coefficients, joint damping, tendon stiffness, and visual appearance. Despite the fact that policy was trained solely in simulation, final model transferred to real Shadow hand succeeded to achieve stable multi-step reorientation, which proved that domain randomization can solve the sim-to-real gap even for complex tasks with multiple contact interactions. The important fact that success was not caused by randomizing any single parameter, but by the combined effect of training across broad distribution conditions, which

prevented it from overfitting to a specific set of simulated dynamics.

A consistent finding related to domain randomization literature, and one of the central ideas in this thesis, is that domain randomization reduces the best performance in simulation, while improves robustness in real-world deployment. The reason for this is a basic trade off between specialization and generalization. Specialized policy can ideally learn the behaviour for one specific environment, while generalized policy must work well in multiple environments. A policy trained with domain randomization cannot use the same dynamics for any single simulated conditions, due to changing of dynamics in each episode. It must learn behaviors that are effective across the whole randomization range. In simulation evaluation with a fixed and predictable environment, specialized policy trained for this exact environment will likely show better performance. While during real robot deployment with physical parameters, which are different from any simulated setup, the more general policy trained with domain randomization is likely to perform better.

Domain randomization implemented in this thesis is intentionally conservative and aimed on three parameters that are directly related to push manipulation task: object mass ($\pm 10\%$), surface friction ($\pm 15\%$), and observation noise (2 mm Gaussian standard deviation). The range in object mass reflects real uncertainty in puck weight and normal manufacturing differences. The friction range reflects changes in the table surface caused by dust, wear or contact area between puck and table. The observation noise reflects AprilTag localization uncertainty under real camera conditions. These variations are significantly smaller than the ones that were used in more complex manipulation tasks such as dexterous in-hand manipulation [15]. This conservative approach was chosen deliberately because bigger randomization ranges can lead to inconsistency of simulated dynamics, which can make training unstable and reduce learning ability of policy [2].

2.7 Task-Adaptive Reinforcement Learning: Original Motivation and Limitations

Development of task-adaptive reinforcement learning framework for single arm robot manipulation was the original motivation of this thesis. The idea was to create a policy that could generalize across different object sizes, shapes and goal configurations in zero-shot manner, without retraining policy each time when the task is changed. This motivation was inspired by Cui et al. [1], who designed a task-adaptive reinforcement policy for dual-arm robot performing assembly task. In their approach, the policy receives additional input describing task parameters like object dimensions or the target pose. Through conditioning the policy on these parameters, the neural network could regulate its behavior for new task setups without requiring further training. This method expands the idea of generalization further than standard goal-conditioned reinforcement learning. In classical goal-conditioned RL policy generalizes only across goal-position. While task-parameter conditioning makes policy adapt to changes in the task itself with object differences and task configurations.

Applying this task-adaptive approach for the Franka Panda robot brought certain practical obstacles that were faced in earlier implementation. To implement task-parameter conditioning for push task, the task must be represented with a continuous set of parameters. It means that different objects with different sizes and forms must be defined as vectors that policy would use as additional input. In the MuJoCo simulation environment changing the geometry of pushed objects demands modification of the XML model, which defines the physical scene, due to the fact that object geometry defines collision geometry, mass distribution and visual appearance. Compared to varying goal position, which requires only updating of goal site coordinates in the environment, it is more complicated. Randomizing object geometry would require either maintaining a library or another XML file for each object type or creating a system that programmatically edits XML file during the training. Both of these options significantly increase complexity of implementation.

The next difficulty is a learning problem itself. Training one policy for working on

multiple object geometries means that the policy must learn several qualitatively different pushing strategies simultaneously. For instance, a small light puck is pushed differently from a large, heavy box. Also, the inertia and friction of these objects change, which changes the amount of force that the robot needs to apply and the trajectory that the end-effector should follow. Early training runs – task_adaptive/SAC_1, discrete_two_sizes/SAC_1, and stage1_fixed/SAC_1 recorded in TensorBoard logs - constantly showed low convergence. The policy failed to improve beyond near-chance performance in most cases. It is assumed that combined variability of various task parameters made the learning problem too complex for the policy to generalize within available computational resources and training budget.

This result is consistent in observation of multi-task reinforcement learning literature. One of the most commonly highlighted problems is conflicting gradients. In training policy on multiple various tasks simultaneously, gradient update that improves performance on one task, may show worse performance on another. Therefore, learning becomes slower and potentially prevents convergence to any effective policy [1]. In order to address this problem, significantly larger training budget and more complex neural network architectures are required. For example, modular or hierarchical designs, or careful task curriculum scheduling — all of which were beyond the scope of the available timeline.

The shift from task-adaptive reinforcement learning to focused ablation study on a single push task was a scientifically justified decision. Instead of creating partially working task-adaptive policy, research changed the focus towards producing rigorous, reproducible results on a clearly defined problem. The ablation study provides directly actionable findings - empirical evidence on the influence of reward design and domain randomization on sim-to-real transfer. Also, the resulting pipeline is a fully deployable system. The lack of task-adaptive manipulation systems for single-arm robots compared to dual-arm work of Cui et al [1], which was the original gap, remains open and highlighted as further research direction in Chapter 6.

2.8 Gaps in Existing Literature

The discussion of related works in this chapter indicates particular research gaps that this thesis is aimed to address.

Controlled ablation of reward design for push manipulation. Although the focus of many studies relies on reward design in robotic manipulation, works on careful comparison between sparse and dense reward functions, where all experimental variables remain constant, are relatively rare. The majority of existing investigations implement dense reward functions without comparison with sparse rewards, or they present HER as a solution for sparse-based problems without comparison of two methods in the same conditions. This thesis addresses this gap by providing a three-condition ablation study with keeping all experiment factors constant. This decision isolates the effect of dense reward structure and provides empirical evidence on the relative benefits of sparse and dense rewards for push manipulation tasks considered in this work.

Domain randomization effects on push manipulation. Most domain randomization demonstrations in existing literature are focused on completing highly complex tasks like dexterous in-hand manipulation [15] or robotic locomotion [22]. While simpler tasks such as planar push, where dynamics is less complex, receives less attention from the robotics community, even though the sim-to-real gap can be significant. This thesis provides an empirical definition of domain randomization effect on both simulation and real robot deployment experiment.

Simulation success rate as an evaluation metric. Many studies on reinforcement learning based on simulation unintentionally suggest that policy with high success rate in simulation will show the identical performance in physical setup. This thesis contradicts this assumption and provides empirical evidence that this assumption is not reliable when domain randomization is used. In the experiments presented in this thesis, policy trained with domain randomization showed lower success rate in simulation compared to dense reward policy without domain randomization – 67% versus 79%, respectively. However, it is expected that policy trained with domain

randomization will demonstrate more robust performance in real-world experiments. This result motivates conducting evaluation with a physical robot instead of using simulation metrics alone, which makes real physical evaluation essential for real-world deployment.

End-to-end pipeline transparency for Franka Panda push manipulation. Despite the demonstration of robotic manipulation via Franka Panda robot from many studies, they rarely describe the full sim-to-real transfer with details to reproduce results. Essential stages like environment design, perception setup, calibration procedures, and deployment on the real robot are only briefly mentioned and not fully documented. This thesis is aimed to improve reproducibility of the sim-to-real research by providing a clear, end-to-end account of each pipeline component, including the observation and action interface design, camera calibration procedure, and real robot safety measures.

Chapter 3

Methodology

This chapter provides full technical methodology for experiments conducted in this thesis. Section 3.1 defines planar push task and explains its formal specification. Section 3.2 describes the implementation of the simulation environment. Section 3.3 details the design of observation space design and reasons behind it. Section 3.4 covers the action space and inverse kinematics control abstraction. Section 3.5 introduces both reward function formulations from the ablation study. Section 3.6 describes the SAC algorithm configuration and hyperparameter choice. Section 3.7 describes in detail the domain randomization strategy. Section 3.8 demonstrates the full sim-to-real pipeline with perception, camera calibration, and the real robot interface.

3.1 Task Definition: Planar Push-to-Goal

The manipulation task considered in this thesis is a planar push-to-goal task. In this task, the robot arm must push a cylindrical object from a randomly chosen starting position across the table surface to the randomly chosen target position. For pushing, a robot can contact the object only with its end-effector and must not grasp it. The task is represented as a series of episodes. In the beginning of the episode, a new initial configuration is created by random sampling of object and goal positions. The problem is formally defined as a Markov Decision Process (MDP), it is characterized by a state space, action space, transition dynamics, reward function, and success

criterion.

The physical setup used for this task includes 7-DOF Franka Emika Panda robotic arm and work space $40\,\text{cm} \times 60\,\text{cm}$ on a flat table surface. The object moved by a robot is a cylindrical puck with radius of 4cm, height of 2cm, and approximate mass of 100g. It should be noted that the simulation environment models a cylindrical puck with 4 cm radius, while the physical deployment uses a cubic object with approximately 5 cm sides. This geometry difference was accepted as the task-relevant dynamics — planar pushing contact — are preserved in both cases. The success criterion is distance-based and unaffected by object shape. The practical implication is discussed in Section 4.6. At the start of each step puck and goal positions are uniformly randomly sampled within workspace boundaries, and a minimum separation distance is enabled to avoid puck starting right on the goal position or too close to it.

The success criteria for this push-to-goal task is defined by keeping the center of puck within 5cm radius from goal position for 10 consecutive timesteps. This requirement shows sustained accuracy by distinguishing actual task completion from momentary contact with the goal area. In other words, robot must push the object to the goals and keep it there instead of passing through the goal area. Episodes run for a maximum of 200 timesteps, corresponding to 10 seconds at the control frequency of 20 Hz. Episodes finish earlier in case of success, but do not terminate early in cases of failure other than out-of-bounds detection.

The simulation setup used throughout this work is illustrated in Figure 1. This task formulation touches on the central challenges of push manipulation such as contact initiation, sustained directed pushing, and precise final positioning, while remaining sufficiently constrained to enable systematic experimental study. The learning problem is simplified by the use of fixed object geometry and single object workspace, in contrast with more complex manipulation scenarios with multiple objects, variable object shapes, which is appropriate given the ablation focus of this work.

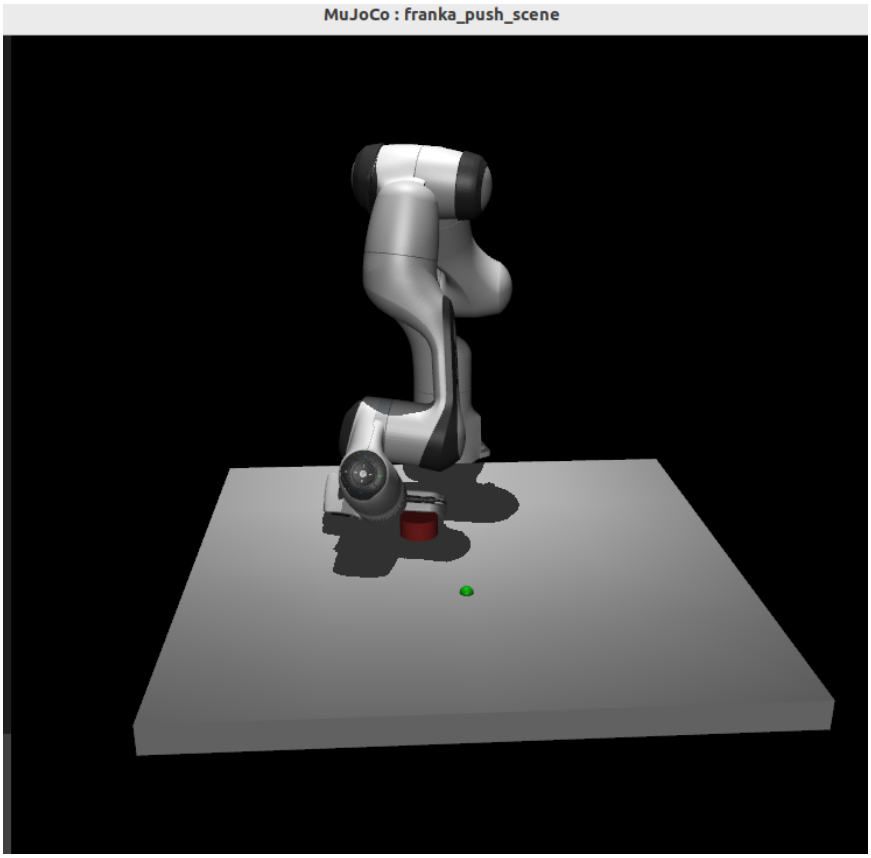

Figure 1: MuJoCo simulation screenshot showing the Franka Panda robot, cylindrical puck, and goal site marker on the table surface.

3.2 Simulation Environment

Platform. The training in this work was performed in MuJoCo physics engine (version 3.1.0) [17], and was interfaced via the Gymnasium environment API (version 0.29.0). The MuJoCo choice was motivated by its accurate and well-documented dynamics solver, its compatibility with the Franka Menagerie model library, and its common adoption in robotic manipulation research, which enables comparison with earlier studies [17]. Gymnasium interface supplies a standardized step-reset-observe API, ensuring compatibility with the Stable-Baselines3 training framework [18].

Environment implementation. The simulation environment is implemented as a custom Python class – Franka Push Env, its path – `src/franka_push_env.py`.

This class wraps the MuJoCo Franka Menagerie XML model, which is high-fidelity kinematic and dynamic description of the Franka Panda arm, with the push task logic, including episode initialization, observation construction, reward computation, domain randomization, success detection, and boundary enforcement. This environment was created from scratch, instead of adapting from existing benchmarks in order to provide total control over all task parameters and reward components.

Episode initialization. At the start of the episode, the state of MuJoCo simulation is reset to canonical configuration and the end-effector is positioned above the workspace. The object position is sampled within workspace boundaries, and the goal position is sampled independently subject to a minimum separation constraint to avoid degenerate episodes. When the domain randomization parameters are enabled, they are resampled in episode initialization, to make each episode present a different physical configuration to the policy.

Physical configuration. The simulation runs at an internal physics timestep of 1 ms (1000 Hz), with the RL control loop executing at 20 Hz. Then each RL timestep corresponds to 50 physics simulation steps, with the inverse kinematics sub-loop executing at 10 sub-steps per RL step (200 Hz effective control rate for the arm joints). This multi-rate structure guarantees the physics engine to resolve contact dynamics accurately at high frequency while the RL policy operates at a lower frequency appropriate for the task timescale.

3.3 Observation Space Design

At each step policy obtains a 14-dimensional observation vector, it is constructed for providing full information about the task-relevant configuration of the robot and the workspace. The observation components are represented in Table I.

The policy deliberately observes two-dimensional planar quantities. The Z-axis was excluded because the end-effector works at fixed height over the table and puck

Table I: Observation Space Definition (14 Dimensions)

Index	Variable	Type	Description
[0:2]	`ee_pos`	Absolute	End-effector XY position in world frame
[2:4]	`obj_pos`	Absolute	Object (puck) XY position in world frame
[4:6]	`obj_vel`	Absolute	Object XY velocity
[6:8]	`ee_to_obj`	Relative	Vector from end-effector to object center
[8:10]	`obj_to_goal`	Relative	Vector from object center to goal
[10:12]	`goal_xy`	Absolute	Goal position in world frame
[12:14]	`ee_vel`	Absolute	End-effector XY velocity

is limited to a planar movement, so the Z-axis coordinates have no relevant information for the task. This reduces input dimensionality without losing task-relevant information, it simplifies policy representation and increases learning efficiency.

Combined inclusion of absolute position information and relative vectors in the same observation is the key design decision. The absolute positions of the end-effector, object, and goal give policy a workspace context, in other words awareness of where those objects are located in relation with table boundaries. The relative vectors `ee_to_obj` and `obj_to_goal` provide the policy with the geometric relationships that directly control the manipulation strategy: the direction and the distance that robot must travel to reach the puck, and the direction and distance puck must travel to achieve the goal position.

By explicitly presenting those geometric relations as observation components, the policy is motivated to learn manipulation strategies grounded in relative spatial reasoning, instead of memorizing fixed coordinates and motor actions. If policy learns how to match the end-effector with `ee_to_obj` vector and subsequently apply force in the `obj_to_goal` direction, it will be able to adapt naturally across new initial configurations because the relative vectors stay unchanged even when absolute position of objects in a workspace change. This design choice is expected to lead to improved generalization in new episode configurations that were not faced during training.

Including of object velocity (`obj_vel`) and end-effector velocity (`ee_vel`) provides the policy more than the current positions, it provides information about motion, which gives opportunity to predict the way arm and puck will move in near future.

This information is highly important for push manipulation, because if the arm does not adjust the force applied on the puck based on the current speed during push, the puck's momentum will make it pass the goal by.

3.4 Action Space and Inverse Kinematics Control

The policy results with two-dimensional continuous action, which represents the expected end-effector velocity in the XY plane: $a = (\Delta x, \Delta y) \in [-1, 1]^2$. This action is scaled to a maximum displacement of ± 3 cm per timestep in simulation, and ± 2 cm per timestep on the physical robot (a 33% reduction applied for safety to limit the kinetic energy of arm motions near the puck and table surface).

The choice of 2D end-effector velocity action space, instead of a 7-dimensional joint torque or joint angle action space, is an intentional simplification called task-space abstraction. The Franka Panda robot has seven degrees of freedom, and all of them must be controlled to achieve end-effector motion. If the policy had to learn right over the joint torques, it would require understanding of robot's kinematics and dynamics, additionally to the push task itself, which is a significantly more complex problem and unlikely to be mandatory for planar push manipulation task [11]. The use of end-effector velocity commands itself, allows the policy to focus on deciding the direction and speed of movement, which is the main task.

The shift from 2D end-effector velocity commands to 7-joint control signals is managed by an inverse kinematics (IK) sub-loop that executes at 10 sub-steps per RL timestep. During each step, the IK controller calculates the joint velocities, which are required for following the desired end-effector's displacement through the Jacobian pseudoinverse approach. After that, it applies it as joint velocity commands in the MuJoCo simulation. The IK problem is essentially a 2D planar tracking problem within the robot's dexterous workspace, because the end-effector's height is fixed at constant value above the table. The sub-step structure provides smooth, consistent arm movement between timesteps, and it maintains the 20 Hz control frequency seen by the RL policy.

During the use of the real robot, the exact same 2D velocity command interface is kept. The real robot's interface converts these commands into Cartesian velocity targets, which are passed to the FrankPy controller. Then the controller runs its own low-level joint impedance control loop at the robot's native control frequency. Preserving the same action interface for both simulation and physical experiment is an intentional design decision. It makes the policy be able to transfer from simulation to real root with zero-shot transfer, without modifying policy network or inference procedure.

3.5 Reward Function Design

Two reward functions are implemented in the FrankaPushEnv, and they can be selected by a configuration parameter. Condition 1 of the ablation study is sparse reward formulation, conditions 2 and 3 are dense progress-based reward formulations.

3.5.1 Sparse Reward

The sparse reward function gives a binary signal:

$$r_t = \begin{cases} 1.0 & \text{if success at timestep } t \\ 0.0 & \text{otherwise} \end{cases} \tag{3.1}$$

Success is defined as $\|P_{obj} - P_{goal}\|_2 \leq 0.05\,\text{m}$ for 10 consecutive timesteps. Using this formulation, the agent does not receive any signal until the task is completed, which makes exploration problem highly complex. The policy is expected to find pushing strategies by random exploration, before any gradient update can be calculated toward the goal.

The sparse reward in the ablation study represents a lower-bound baseline. It is used to show minimal possible reward signal, because it directly rewards only expected behavior. It makes it a reference point in evaluation of rewards shopping advantages. In fact, the sparse reward does not contain any design parameters that

could introduce unintended incentives, so it provides a clean experimental condition.

3.5.2 Dense Progress-Based Reward

The dense reward function combines four components into a single scalar reward at each timestep:

$$r_t = -\lambda_d \cdot d_t + \lambda_p \cdot \Delta d_t - \lambda_e \cdot e_t + \lambda_s \cdot \mathbf{1}[\text{success}], \tag{3.2}$$

where $d_t = \|P_{obj,t} - P_{goal}\|_2$ is the current distance between object and goal, $\Delta d_t = d_{t-1} - d_t$ is reduction of distance from object to goal since the last timestep (positive when the object moves in direction to the goal), $e_t = \|P_{ee,t} - P_{obj,t}\|_2$ is the distance from the end-effector to the object, and $\mathbf{1}[\text{success}]$ is indicator function equal to one on the timestep when the success criterion is first met. Coefficients established as:

$$\lambda_d = 1.0, \quad \lambda_p = 10.0, \quad \lambda_e = 0.1, \quad \lambda_s = 10.0$$

The importance of each component is introduced in Table II.

Table II: Dense Reward Function Components

Component	**Formula**	**Coefficient**	**Purpose**
Distance penalty	$-d_t$	1.0	Penalizes current object-to-goal distance; always negative, encourages proximity
Progress reward	$+\Delta d_t$	10.0	Rewards per-step reduction in distance; provides dense signal throughout episode
Proximity penalty	$-e_t$	0.1	Discourages end-effector from wandering away from puck
Success bonus	+10.0	—	Large terminal reward upon achieving goal; reinforces completion

In contrast with relatively simpler dense reward function, which use solely distance penalty, the progress term Δd_t is identified as a significant advancement. An only distance penalty $-d_t$ gives a reward landscape, it is constant along iso-distance

contours. In other words, the policy receives the same reward in episodes where puck is pushed towards the goal or away from it, unless the current to the goal remains the same. That's why the policy does not get a gradient signal about the direction the puck has to move, but only about its current position. This problem is addressed by the progress term rewarding any reduction in distance at every step. It provides the policy a clear incentive for generating an object's movement in goal direction at each step, even in cases when the object is far from the goal and distance penalty is small or changing slowly. For term's domination and ensuring that movement to the goal is a key factor guiding the policy updates, the progress coefficient has 10x scaling.

The secondary role is played by the proximity penalty $-0.1 \cdot e_t$. It gives penalties in situations where the end-effector moves far from the puck to discourage the policy from exploiting strategies involving large arm excursions between consecutive contact events. To make sure that it directly influences policy without overriding the main goal-directed incentive, the term is given a weight ten times smaller than distance penalty coefficient.

The success bonus +10.0 is provided one time, at the time step when the success criteria is satisfied for the first time. Its value is established equal to the maximum progress reward over one episode that can be achieved. In the absence of this terminal bonus, the policy could keep pushing the puck to the goal in small steps without satisfying sustained-proximity criteria receiving high cumulative reward. This would make a mismatch between the reward task objective.

3.6 SAC Algorithm and Hyperparameter Selection

3.6.1 Algorithm Overview

In all three experimental conditions, Soft Actor-Critic [13], [14] is used as a learning algorithm. This algorithm uses a stochastic policy $\pi_\phi(a|s)$ parameterized by ϕ, and two Q-function networks $Q_{\theta_1}(s,a)$, $Q_{\theta_2}(s,a)$ parameterized by θ_1 and θ_2 respectively. The policy is trained for maximizing the entropy-regularized objective of Equation (2.1).

The Q-functions are trained by minimizing the soft Bellman error:

$$\mathcal{L}(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}\left[\left(Q_{\theta_i}(s,a) - y\right)^2\right] \tag{3.3}$$

where $\mathcal{D}$ is a replay buffer and the target value y is calculated with two target networks $\bar{\theta}_1$ and $\bar{\theta}_2$:

$$y = r + \gamma\left(\min_{i=1,2} Q_{\bar{\theta}_i}(s', \tilde{a}') - \alpha \log \pi_\phi(\tilde{a}'|s')\right), \quad \tilde{a}' \sim \pi_\phi(\cdot|s') \tag{3.4}$$

Using the minimum of two Q-functions estimates when the target computation helps minimize overestimation bias, this is a common issue for methods with only one critic. The temperature parameter α is adjusted during training automatically. It is done for keeping policy's entropy near the target value $H_{\text{target}} = -|A| = -2.0$. It follows a method of automatic entropy tuning introduced by Haarnoja et al. [14].

3.6.2 Hyperparameter Configuration

The exact same SAC hyperparameters were used for all three experiments. It can guarantee that any difference in the results was caused only by either reward function or domain randomization, not from learning setup changes. The full hyperparameter configuration is provided in Table III.

The network consists of three hidden layers with 256 units each, which is deeper than the default Stable-Baseline3 that uses 2 layers with 64 units. This decision was caused by the nature of push tasks, involving contact between arm and the object. In this task, relationship between observed space and optimal action is non-linear and has discounting transitions during the contact between arm and the object, so it can require larger capacity compared to simpler continuous control tasks. Actor and two critics networks use the same architecture with ReLU activation functions.

Eight parallel environments are run at the same time using SubprocVecEnv, with each worker running in a separate MuJoCo process. The number of gradient update steps per environment step is 8, which is equal to the number of parallel environments. It keeps the ratio between collected experience and policy updates balanced. This

Table III: SAC Hyperparameter Configuration

Hyperparameter	**Value**	**Rationale**
Learning rate	3×10^{-4}	Standard value for SAC in manipulation tasks [13]
Replay buffer size	1,000,000	Large buffer for off-policy experience diversity
Batch size	512	Larger than SB3 default (256) for stable gradient estimates
Network architecture	[256, 256, 256]	Three hidden layers; deeper than default [64, 64] to accommodate contact-rich dynamics
Gradient steps per env step	8	Matched to number of parallel environments
Target entropy	-2.0	Equal to $-\|A\|$
Discount factor γ	0.99	Standard for episodic tasks with horizon 200
Polyak coefficient τ	0.005	Standard soft target network update rate
Parallel environments	8	SubprocVecEnv for true multiprocessing
Total training timesteps	2,000,000	Sufficient for convergence on this task

setup means that each time when 8 new transitions are collected (one from each parallel environment), 8 gradient updates are performed. It sets a 1:1 ratio for data collection and updates, which provides decent sample efficiency, while not overfit the policy to the current batch of experience.

3.7 Domain Randomization Strategy

The domain randomization is used in Condition 3 in the ablation study. In the beginning of each episode, sampling multipliers or offsets from specified distributions randomly change the simulation's physics parameters, as shown in Table IV. These changes are applied separately in each episode to make policy experience different physical configuration every episode.

Table IV: Domain Randomization Parameters

Parameter	**Nominal Value**	**Randomization Range**	**Real-World Motivation**
Object mass	$\sim$100 g	$\times[0.9, 1.1]$ ($\pm$10%)	Manufacturing tolerances; unknown true puck weight
Surface friction	Nominal μ	$\times[0.85, 1.15]$ ($\pm$15%)	Table surface wear, dust, contact area variation
Observation noise	0 mm (std)	$+\mathcal{N}(0,\ 2\,\text{mm}^2)$	AprilTag localization uncertainty under real camera conditions

The choice of three parameters was motivated by physical examination of the key differences that can arise between simulation and real world in the context of push task. Object mass directly influences its inertia and momentum during the contact, it determines how far it travels when the force is applied. Surface friction controls the deceleration of an object while sliding, and the amount of force needed to initiate object motion. Observation noise highlights the fact that in the real experiment object and goal positions are detected by the ApriTag system compared to simulation, where they are taken directly from the physics engine, and are therefore subject to localization errors.

The randomization ranges are intentionally kept conservative. A mass variation in ±10% aligns normal manufacturing deviation in a real puck and uncertainty in its actual weight during conducting the experiment. A friction variation of ±15% represents realistic variability in the table surface caused by dust or wear, without introducing the extreme values that would make the pushing task dynamically inconsistent across episodes. The 2 mm standard deviation in the observation noise is selected according to expected accuracy of AprilTag system along with the RealSense camera, as characterized in camera calibration.

Extreme domain randomization including changing mass or friction by ±50% was avoided on purpose. Too large variation can make training unstable, because it can create some episodes, where dynamics is so different that no policy can achieve success in this distribution, so it can lead to low quality training or complete training failure [2]. This conservative approach is focused on the realistic potential variations, instead of worst-case scenarios. It reflects the principle that domain randomization should embrace the real uncertainties in the deployment environment, not arbitrary expansion of the training distribution.

The influence of DR on training is discussed in detail in Chapter 4. Shortly, the DR condition had a lower success rate in comparison with no DR conditions – 67% and 79%, respectively. It aligns with the expected trade off between simulation performance and real deployment robustness, as reported in studies [15], [22].

3.8 Sim-to-Real Transfer Pipeline

The sim-to-real transfer pipeline translates the MuJoCo trained policy into executable actions for the real Franka Panda robot. The pipeline contains three key components: a perception module for estimating object and goal positions in the robot workspace, a camera calibration procedure that establishes the geometric relationship between camera coordinates and robot coordinates, and a real robot interface that exposes the identical observation and action API as the simulation environment.

3.8.1 Perception: AprilTag Detection

The positions of the object and the goal in the workspace are estimated by April-Tags markers, and then detected by an Intel RealSense RGB-D camera, which is mounted overhead and faces downward the table surface. AprilTag is a system of visual markers, in which printed square markers encode unique IDs. The corners of these markers are detected and then decoded from camera image for estimation of the six-degree-of-freedom pose of the marker relative to the camera [23].

There are two AprilTag markers used in the workspace: Tag 0 - is attached to the object, Tag 1 - is placed on the table surface, indicating the goal position. Every control step, the RealSense camera captures the image using 1280x720 resolution and 30+ fps. The pupil-apriltags library for detection, processes the image using four parallel detection threads, and returns the detected pose of each tag in camera coordinates. Then the XY position of each tag is converted into robot base frame coordinates using the calibrated camera-to-robot extrinsic transform (Section 3.8.2).

Both markers are from the tag36h11 family, because it provides good balance between marker size, detection range, and robustness to partial occlusion and motion blur. With the camera being set overhead, markers are clearly visible for the camera in most cases, with the sole exception when the end-effector moves right over a marker during contact. However, this problem was quickly solved by reattaching the tag from above to the side of the object.

Figure 2: The real robot setup, the Franka Panda arm, the table workspace, the object with AprilTag on top, the goal position marker, and the overhead Intel RealSense camera mounting.

3.8.2 Camera Calibration

A planar homography was performed for mapping camera coordinates right to robot base frame XY coordinates. The calibration process uses the existing AprilTag marker system: the cube with attached Tag 0 is placed on 5 different locations on the workspace which are measured relative to the robot's base with a ruler. At each po-

sition the detector reports the pixel coordinates of the tag center, which the operator confirms interactively on the live camera feed before recording. The five calibration positions are listed in Table V.

Table V: Camera Calibration Reference Points

Point	**Robot X (m)**	**Robot Y (m)**
1	0.45	+0.15
2	0.45	−0.15
3	0.60	+0.15
4	0.60	−0.15
5	0.52	0.00

Based on 5 pixel-to-robot correspondences, a 3×3 planar homography matrix H is calculated with the use of OpenCV's `findHomography` function, so that $(x_r, y_r) \sim H(u, v, 1)^T$. A complete SE(3) extrinsic transform is not required, because the camera is mounted overhead at a fixed height and all objects stay on the flat table surface. In this setup, a pixel-to-table is exactly a planar homography. The calibration accuracy is verified by reprojecting each of the five points through H and reporting the Euclidean error in centimetres. The resulting matrix is stored in `camera_calibration.yaml` and loaded at the start of every real robot session.

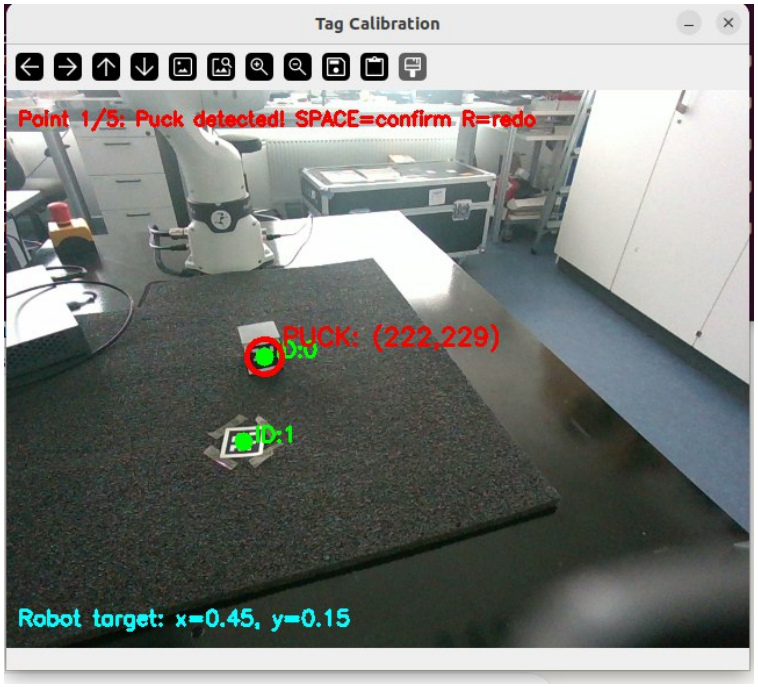

Figure 3: Calibration script interface showing AprilTag detection during the camera-to-robot coordinate calibration procedure. Tag ID 0 (puck, red circle) is detected at pixel coordinates (222, 229), mapped to robot frame position $x = 0.45\,\text{m}$, $y = 0.15\,\text{m}$ (shown bottom left). Tag ID 1 (goal marker) is simultaneously visible. Point 1 of 5 is shown here awaiting operator confirmation.

3.8.3 Real Robot Interface

The real robot environment is implemented in `src/real_robot_env.py` as a class that provides the same Gymnasium-compatible step-reset API as the simulation environment. Remaining the identical interface is an important design solution that enables zero-shot policy transfer. The policy network receives observation and outputs actions in the same format in real robot deployment as it did in simulation training, without changing in policy weights or inference procedure.

The mapping between simulation quantities and their real robot counterparts is summarized in Table VI.

On the physical robot, end-effector and object velocities are estimated taking differences between consecutive position measurements at 20 Hz, which matches the

Table VI: Simulation vs. Real Robot Interface Comparison

Aspect	**Simulation**	**Real Robot**
Observation space	14-dim (same)	14-dim (same)
Action space	$2D\ \Delta x/\Delta y \in [-1, 1]^2$	$2D\ \Delta x/\Delta y \in [-1, 1]^2$
Action scaling	$\pm 3\,\mathrm{cm/step}$	$\pm 2\,\mathrm{cm/step}$ (safety reduction)
Object position source	MuJoCo qpos	AprilTag detection + calibration
End-effector position source	MuJoCo site_xpos	FrankaPy get_pose()
Object velocity	MuJoCo qvel	Finite difference of AprilTag positions
End-effector velocity	MuJoCo qvel	Finite difference of FrankaPy poses
Observation noise	Simulated $\mathcal{N}(0,\, 4\,\mathrm{mm}^2)$	Real camera and detection noise

2 mm observation noise used during training with domain randomization. In the robot interfaced, several safety constraints have been applied: action scaling is reduced from ± 3 cm/step to ± 2 cm/step, the end-effector height is fixed at 3.5 cm above the table with a hard lower bound of 2.5 cm, and workspace boundary constraints prevent motions outside the 40 cm $\times$ 60 cm workspace. These safety constraints are applied by clipping, making it transparent to the policy.

The policy selected for real robot deployment is a domain randomized dense reward from Condition 3. Despite the fact it achieved the lowest success rate - 67%, it was specifically trained to succeed under variations of physical condition, compared to the Condition 1 and 2 that are overfitted to the exact nominal simulation dynamics and are expected to fail handling real experiment conditions. The choice of policy with domain randomization is a test of hypothesis that simulation robustness translates to real-world transfer, which is described in Chapter 4.

The full source code for the simulation environment, training pipeline, evaluation scripts, and real robot interface is publicly available at: `https://github.com/Dal3l/franka-push-rl` [34]

Chapter 4

Experiments and Results

This chapter demonstrates experimental evaluation of three trained policies. Section 4.1 explains the design of ablation study. Section 4.2 describes computational setup and training process. Section 4.3 shows convergence analysis and training curves. Section 4.4 gives quantitative evaluation results. Section 4.5 examines failure mode structure. Section 4.6 shows real robot deployment results.

4.1 Ablation Study Design

The ablation study is organized as an experiment with three different conditions with reward function and domain randomization modified separately. While other factors – SAC hyperparameters, network architecture, observation space, action space, training budget, and random seeds are held the same. This controlled setup isolates any differences from other factors except the manipulated variables. The three conditions are defined in Table VII.

Condition 1 established a lower bound, because it presents the hardest learning problem with sparse reward and no shaping. Condition 2 introduced the dense progress-based reward with identical hyperparameters as Condition 1, so it excluded the effect of reward design. Condition 3 adds the domain randomization strategy on the top of Condition 2, so it represents the complete training pipeline for the transfer to the real robot. The progression from Condition 1 to Condition 3 provides an

Table VII: Experimental Conditions

Condition	**Config File**	**Reward Type**	**Domain Rand.**	**Purpose**
1 — Sparse	`baseline_sparse.yaml`	Sparse (0/1)	No	Lower bound baseline
2 — Dense	`baseline.yaml`	Dense progress	No	Isolates reward design effect
3 — Dense + DR	`baseline_DR.yaml`	Dense progress	Yes	Full pipeline for real deployment

opportunity to assess the independent contribution of each modification.

4.2 Training Procedure

All three conditions were trained for 2,000,000 timesteps with the use of eight parallel SubprocVecEnv workers. With gradient_steps set equal to the number of parallel environments (8), each collected timestep corresponds to one gradient update, resulting in 2,000,000 gradient updates per condition. Training was performed on a workstation equipped with an NVIDIA GeForce RTX 2080 Ti GPU (11 GB VRAM), an Intel Core i9-7900X CPU (3.3 GHz, 20 logical cores), and 32 GB of RAM. Policies were evaluated during the training by a separate evaluation environment that ran 20 episodes each time, and results were recorded in TensorBoard. For all quantitative evaluations, which was mentioned in this chapter, the final policy checkpoint at 2,000,000 timesteps was used. All conditions were evaluated at 100 episodes with new sampled initial states for achieving reliable performance estimation.

4.3 Training Curves and Convergence

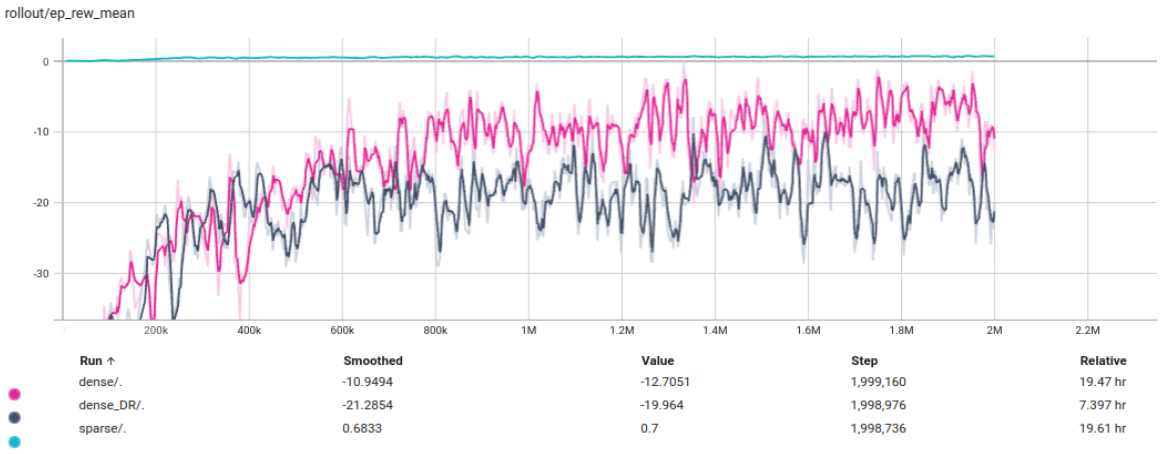

Figure 4: Training reward curves (`rollout/ep_rew_mean`) for all three conditions over 2,000,000 timesteps. Sparse reward (cyan) remains near zero throughout training. Dense reward without DR (pink) and dense reward with DR (dark) both converge, with dense-no-DR achieving higher mean episode reward in simulation.

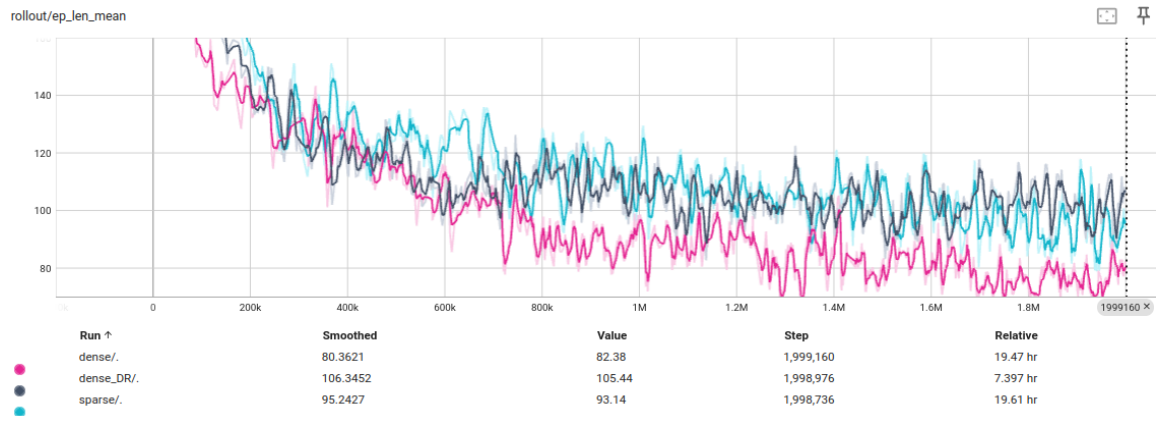

Figure 5: Mean episode length (`rollout/ep_len_mean`) for all three conditions over 2,000,000 timesteps.

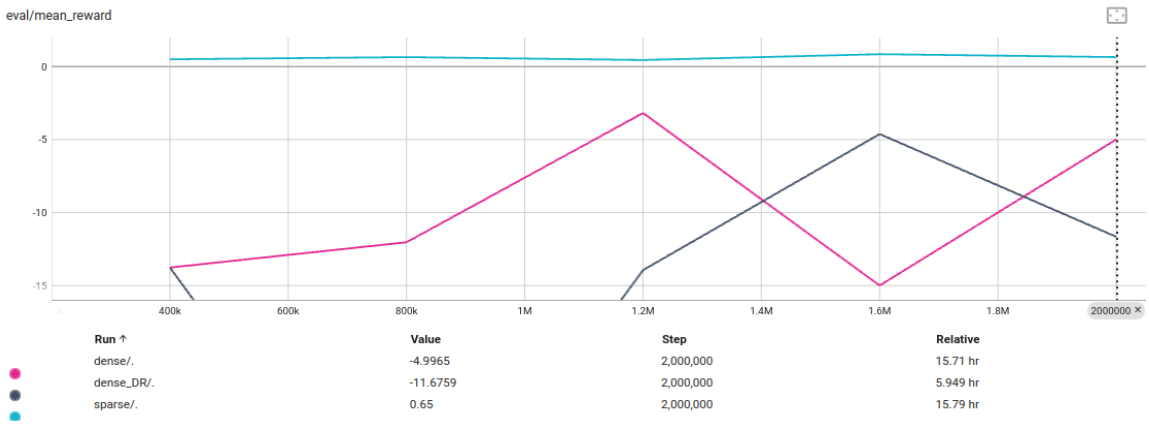

Figure 6: Evaluation mean reward (`eval/mean_reward`) recorded every 50,000 steps for all three conditions.

The training curves demonstrated visibly different learning behavior in three conditions. Condition 2 and 3, which used dense reward, showed earlier and more smooth convergence. It is clearly seen in the climbing rollout/ep_rew_mean curves starting at 200,000 timesteps. It happened due to progress-based reward, which gives a denser gradient signal at each step, even in cases when goal has not been reached. In contrast, the sparse reward condition produces a reward curve that stays near zero throughout 2,000,000 timesteps. However, it does not mean learning failure, it happens because of binary reward structure. The policy receives 0 every step and 1 only on success, that's why the mean reward is numerically close to zero when the policy succeeds often. The rollout/ep_len_mean curve, which is independent from reward scale, proves that sparse reward is learning. The episode length drops from approximately 145 steps to 95 throughout the training, it means that policy learns to achieve the goal progressively faster. Despite the curve staying near zero, the sparse reward condition achieves the same success rate as the dense no-DR condition – 79%. It assumes that SAC replay buffer stores enough sparse reward signal to maintain full convergence in 2,000,000 timesteps, given the relatively simplified geometry of the push task.

Condition 3 with domain randomization achieved lower mean reward compared to Condition 2. It is observable on rollout/ep_rew_mean curves, where the Condition

3 curve is kept below the curve of Condition 2 from the start of training until its end at 2,000,000 timesteps. This output matches with an expectation of domain randomization use. The policy cannot adapt to one fixed set of dynamics because physics parameters such as object mass, surface friction, and observation noise change randomly in the beginning of each episode. Behaviors that work effectively under one set of physics parameters can perform poorer or be penalized under another. Therefore, there is a limit on the mean reward achievable under the randomized distribution. The policy learns a more conservative strategy that is reliable across randomized settings rather than aggressively exploiting the nominal simulation dynamics. It causes lower rewards during training compared to dense condition with no domain randomization, however it creates policy expected to generalize better to real-world variations in physics.

4.4 Quantitative Evaluation Results

Table VIII presents the full quantitative evaluation results across all three conditions, estimated over 100 evaluation episodes per condition.

Table VIII: Quantitative Evaluation Results (100 Episodes)

Metric	**Sparse (C1)**	**Dense No DR (C2)**	**Dense + DR (C3)**
Success Rate	79%	79%	67%
Mean Final Distance	0.118 m	0.156 m	0.112 m
Median Final Distance	0.033 m	0.026 m	0.035 m
Mean Steps to Success	44.4	48.2	48.7
Training Timesteps	2,000,000	2,000,000	2,000,000

Confidence intervals were computed using the Wilson score method. The 95% CI for Condition 1 and 2 (both 79%) is [70%, 86%], and for Condition 3 (67%) is [57%, 75%]. The intervals overlap, indicating the difference in simulation success rates between DR and no-DR conditions is not statistically significant at the $p<0.05$ level. However, the directional finding — that DR reduces simulation performance

— is consistent with domain randomization literature and is evaluated in practice through real robot deployment.

Finding 1. Sparse reward matches dense reward in final success rate. Condition 1 and Condition 2 achieved the same 79% success rate, although there is significant discrepancy in reward function complexity. This result is consistent with prior findings that sparse rewards can support policy learning in combination with efficient off-policy methods and large replay buffers for constrained manipulation tasks [16]. However, it contrasts with expectations for more complex multi-step tasks, suggesting the planar push geometry is sufficiently constrained for SAC to accumulate adequate sparse signal within 2,000,000 timesteps. Specifically in this task, the simplicity of the problem – only one object and planar workspace, appears to make it easy, so SAC replay buffer accumulates enough sparse reward signal to guide learning to the same final performance as dense reward condition. Remarkably, the sparse condition performs successful episodes in less steps on average – 44.4 versus 48.2. It suggests that it directly learns more pushing strategy, while dense policy may spend extra steps for small adjusting, motivated by proximity penalty in the reward.

Finding 2: Domain randomization reduces simulation success rate. Condition 3 achieved a success rate of 67%, which is 12 percentage points less than Condition 2. This result corresponds with domain randomization literature [15], [22] and shows trade of between specialization and generalization. It means that the policy trained across varied physics cannot use the dynamics of a single simulated condition, so its peak performance on a fixed evaluation environment. However, this drop in simulation success does not mean that DR policy is inferior. It shows that simulation success is not an incomplete evaluation criterion for policy deployed to the real robot. The correct evaluation criterion is performance on a real robot, which is reported in Section 4.6.

Finding 3: Mean versus median distance reveals failure mode structure. In all experimental conditions the mean final distance is around 0.112–0.156 m, which significantly larger than median final distance that is 0.026–0.035 m. This indicates that

distribution of final distances is right-skewed, which means that most episodes finish with puck very close to the goal, within or near 5 cm radius, while a smaller number of episodes catastrophic failures, which ends with puck far away from the goal, pull the mean reward. It does not mean that policy fails uniformly, but that it succeeds in most of configurations, and fail drastically in a small subset. This structure is examined further in Section 4.5.

4.5 Failure Mode Analysis

4.5.1 Simulation Failure Modes

Based on examination of failed episodes in simulation, failures in the DR condition, which is 33% failure rate, were organized as three primary modes, represented in Table IX.

Table IX: Simulation Failure Mode Analysis (Condition 3, DR)

Failure Mode	**Estimated Frequency**	**Description**
Catastrophic miss	∼15%	Puck pushed out of the goal region entirely; arm overshoots contact
Approach failure	∼10%	Arm fails to make contact with puck from initial position
Oscillation	∼8%	Policy pushes puck back and forth near goal, never holding for 10 consecutive steps

The catastrophic mode is the key reason for the elevated mean final distance compared to the median and it accounts for the largest part of failures. These episodes finish with the puck far from the goal, therefore, it significantly inflates the mean, explaining the mean-median divergence observed in Table VIII. The second important failure mode is an oscillation mode. It happens when the puck is brought close to the goal, but it does not satisfy the sustained ten-step proximity criterion. Instead, the puck goes back and forth that repeatedly enters and exits the success radius. This behavior indicates that policy has not completely learned the temporal persistence

requirement of the success criteria.

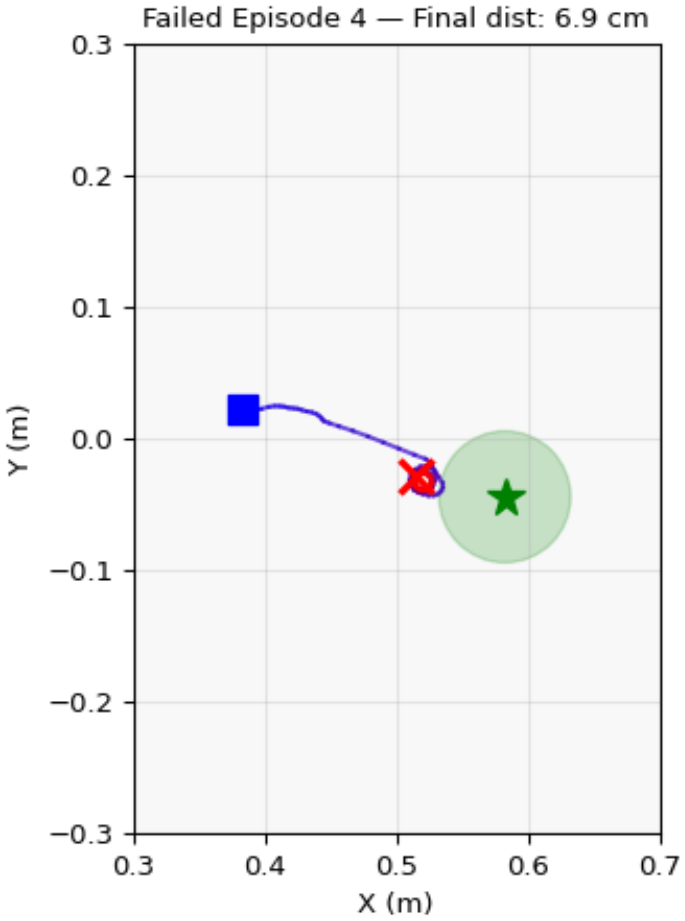

Figure 7: Near-miss failure trajectory of the Dense + DR policy. The puck (path colored blue to red) reaches within 6.9 cm of the goal but the policy fails to hold it within the 5 cm success radius for the required 10 consecutive timesteps, resulting in episode truncation. This oscillation pattern near the goal accounts for a significant portion of the 33% failure rate observed in simulation.

4.5.2 Real Robot Failure Modes

During 20 episodes evaluation on the real robot, four different types of failure modes were highlighted and presented in Table X. These categories were identified before the real experiment based on the sim-to-real gap analysis, and then were confirmed and evaluated using the observed results from testing.

Table X: Real Robot Failure Mode Analysis.

Failure Mode	**Episodes**	**Frequency**	**Root Cause**	**Diagnostic Evidence**
Final-approach stall	2, 12, 14	30% of straight-line	Policy outputs weak actions near goal; impedance compliance reduces effective force	Cube pixel position frozen at ∼6–7 cm from goal for 150–180 steps; action magnitude ∼0.2 normalized
Large Y-offset — no contact	3, 5, 7	60% of large-offset trials	Policy approaches from wrong direction; EE misses cube entirely	Cube pixel coordinates unchanged from step 1 to 200; EE moving but never reaching cube
Small Y-offset — inconsistent	16, 18	50% of small-offset trials	Policy near boundary of training distribution; high-variance outcome	Ep. 16: progress to 7.8 cm then stall; Ep. 18: no movement at all
Cube drift (edge contact)	15	5% overall	EE contacts cube edge rather than face center, causing rotation	Cube visibly rotates and moves laterally away from goal direction
Excessive initial distance	8	5% overall	Policy not trained on $x < 0.38$ initial positions; arm retreats	EE target goes to x = 0.30 at step 1 and remains there for entire episode

The final-approach stall appeared as the most significant failure mode for aligned setup, making 30% of failures in a straight-line motion. In these episodes, the cube is very close to the goal – around 6–8 cm, however policy keeps sending actions with normalized magnitude approximately 0.15–0.30, which is not enough to overcome the

surface friction and the compliance of the Cartesian impedance controller. It is different from a simulation experiment, where the physics engine behaves proportionally to even small commanded forces. Impedance control of the real robot deflects under resistance instead of applying full commanded force. This type of compliance was not included in domain randomization strategy, which included only mass and friction randomization, excluding actuator compliance [27]. The large Y-offset failure mode represents the limitation of policy's generalization across the space. When the lateral displacement of the cube is greater than ± 5 cm, the policy approaches it from the wrong direction, and fails to make contact with it during all five large off-set episodes. It means that diagonal push configurations were not included enough in the training distribution, although the workspace is two-dimensional.

4.6 Real Robot Deployment Results

The Condition 3, which is policy with domain randomization, was deployed to the physical robot Franka Emika Panda using the robot interface described in Section 3.8.3. Structured evaluation of 20 episodes was conducted after fixing all engineering issues. The most important pre-evaluation issue was in the way observations were conducted. The incorrect order of variable updates caused the object's velocity to always be reported as zero. It means that policy systematically received incorrect values of velocity or indices [4:6] and [12:14] of the 14-dimensional observation. Other issues such as a calibration reloading at every step, camera warmup being repeated per detection call, and a control frequency of 0.5 s per step (2 Hz effective) were resolved as well. All these issues were fixed in advance of a physical experiment. The final deployment was run at 10 Hz with correct observations throughout. The control frequency of 10 Hz in the deployment is different from the 20 Hz frequency used in the simulation, representing another sim-to-real factor that was not modeled.

The camera calibration was verified before the evaluation, it achieved a mean reprojection error of 0.46 cm and a maximum reprojection error of 1.10 cm across five calibration points. In order to account for this remaining error the radius of success

on the real robot was established as 6 cm compared to 5 cm in simulation. Also this larger threshold compensates for the systematic offset between tag detected position and actual cube center, which happens because tag is attached to the side of the cube instead of its top surface.

Figure 8: Photo of the complete real robot setup with the Franka Panda arm over the workspace, the black cube with AprilTag on its side face placed on the craft paper surface, the flat goal marker on the table, and the overhead Intel RealSense D435 camera.

4.6.1 Evaluation Protocol

In total, twenty episodes were conducted. In the start of every episode, the cube and the goal marker were placed at specific (x, y) positions by the operator. The robot

adjusted itself to a home position, then verified detection of both tags with a retry loop and started executing the policy. Episodes lasted up to 200 steps, which corresponds to 20 seconds at a control frequency of 10 Hz. The positions of the cube and the goal were randomized across four different configuration types to evaluate policy's generalization. These types are: straight-line (cube y $\approx$ 0, goal y $\approx$ 0), small Y-offset ($\pm$2 cm lateral), large Y-offset ($\pm$5 cm lateral), and one far-puck configuration with initial distance exceeding 20 cm.

4.6.2 Full Episode Results

The complete results of all 20 episodes are presented in Table XI.

Table XI reveals several clear patterns across the 20 episodes. The straight-line configurations (Episodes 1, 4, 6, 9–11, 13, 17) produced consistent successes with final distances of 0.042–0.060 m and short completion times of 24–29 steps, demonstrating reliable performance for aligned pushes. In contrast, all five large Y-offset trials (Episodes 3, 5, 7 and the goal-offset Episode 7) resulted in failures with large final distances of 0.143–0.244 m, indicating the policy never made contact with the cube. The final-approach stall pattern is clearly visible in Episodes 2, 12, and 14: the robot reaches within 6–7 cm of the goal but exhausts the 200-step limit without achieving the sustained 6 cm proximity criterion. Episode 8 stands apart as the only far-puck failure, with the arm retreating immediately rather than attempting contact. The two small Y-offset trials (Episodes 16 and 18) produced inconsistent outcomes – one near-success stall and one total non-contact – reflecting the policy operating near the boundary of its training distribution.

Table XI: Real Robot Evaluation — All 20 Episodes.

Ep	**Puck (x,y)**	**Goal (x,y)**	**Result**	**Final Dist.**	**Steps**	**Note**
1	0.45, 0.00	0.60, 0.00	SUCCESS	0.051 m	25	—
2	0.40, 0.00	0.60, 0.00	FAIL	0.069 m	200	Final-approach stall
3	0.45, +0.05	0.60, 0.00	FAIL	0.195 m	200	Large Y-offset, no contact
4	0.45, 0.00	0.60, 0.00	SUCCESS	0.052 m	24	—
5	0.45, −0.05	0.60, 0.00	FAIL	0.143 m	200	Large Y-offset, no contact
6	0.43, 0.00	0.60, 0.00	SUCCESS	0.060 m	27	—
7	0.45, 0.00	0.60, +0.05	FAIL	0.155 m	200	Large Y-offset goal, wrong direction
8	0.35, 0.00	0.60, 0.00	FAIL	0.244 m	200	Excessive initial distance
9	0.45, 0.00	0.60, 0.00	SUCCESS	0.042 m	24	—
10	0.40, 0.00	0.60, 0.00	SUCCESS	0.060 m	29	—
11	0.45, 0.00	0.60, 0.00	SUCCESS	0.059 m	29	—
12	0.40, 0.00	0.60, 0.00	FAIL	0.068 m	200	Final-approach stall at 6.8 cm
13	0.43, 0.00	0.60, 0.00	SUCCESS	0.056 m	28	—
14	0.42, 0.00	0.60, 0.00	FAIL	0.062 m	200	Final-approach stall at 6.2 cm
15	0.45, 0.00	0.60, 0.00	FAIL	0.075 m	200	Cube drifted sideways on edge contact
16	0.40, +0.02	0.60, 0.00	FAIL	0.078 m	200	Small Y-offset, stalled near goal
17	0.43, 0.00	0.60, 0.00	SUCCESS	0.054 m	27	—
18	0.45, +0.02	0.60, 0.00	FAIL	0.160 m	200	Small Y-offset, no contact
19	0.45, −0.02	0.60, 0.00	SUCCESS	0.060 m	29	—
20	0.45, 0.00	0.60, +0.02	SUCCESS	0.056 m	26	—

4.6.3 Results by Configuration Type

Table XII presents a summary of performance on a configuration type. The overall success rate is 50% (10 episodes out of 20). For straight-line configuration, which is primary in training distribution, the success rate is 80% (8 out of 10). It is greater than 67% success rate in simulation, where the same domain randomization policy was used. This is the main quantitative finding in real robot evaluation.

This comparison should be interpreted with caution, however, as the real robot evaluation used a 6 cm success radius compared to 5 cm in simulation — a threshold 44% larger in area — to account for calibration uncertainty. The results are best understood as comparable rather than strictly superior performance.

Table XII: Real Robot Results by Configuration Type.

Configuration	**Description**	**Trials**	**Successes**	**Rate**
Straight-line	Puck y=0, goal y=0	10	8	80%
Small Y-offset (±2 cm)	Puck or goal ±2 cm lateral	4	2	50%
Large Y-offset (±5 cm)	Puck or goal ±5 cm lateral	5	0	0%
Far puck (x=0.35)	Initial distance >20 cm	1	0	0%
Overall	**All 20 episodes**	**20**	**10**	**50%**

Note: configurations labeled 'Puck Y-offset' test the policy's ability to approach and contact a laterally displaced object. Configurations labeled 'Goal Y-offset' test the policy's ability to push toward a laterally displaced target. These represent distinct generalization challenges. Episode 7 (goal Y-offset +5 cm) is categorized separately from puck Y-offset trials in Table XII.

Figure 9: A sequence of 4 frames from a successful straight-line episode. Frame 1: initial state with cube at approximately x=0.45, arm at home position. Frame 2: arm completing its approach arc to position behind the cube. Frame 3: arm in contact with cube mid-push. Frame 4: final state with cube close to the goal marker.

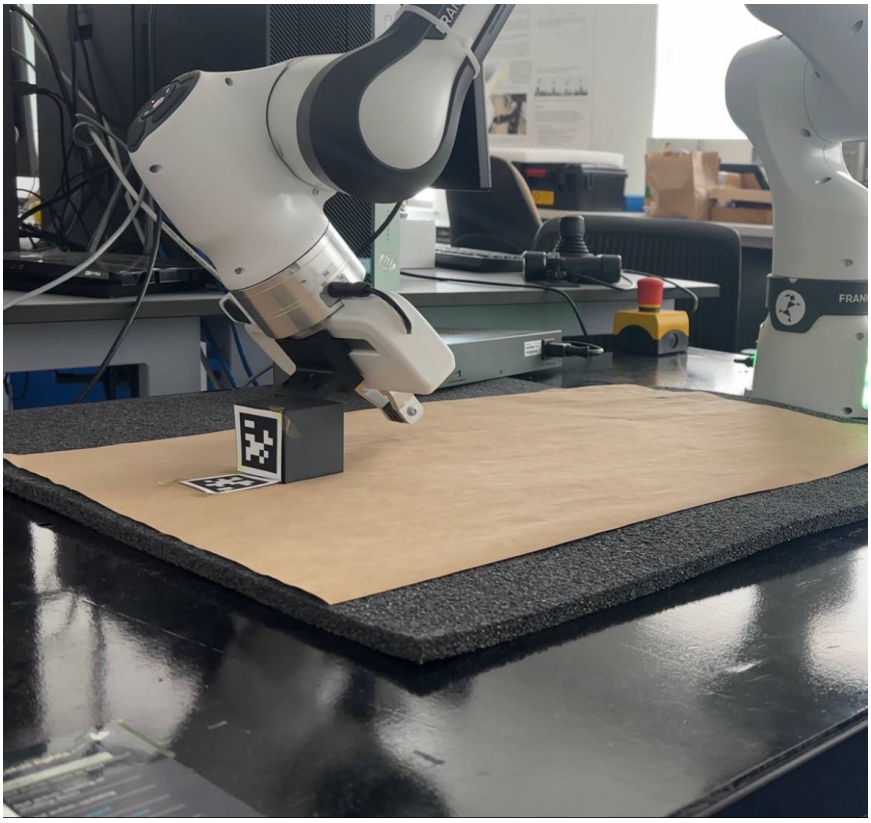

Figure 10: A photo of a successful episode outcome showing the black cube positioned close to the flat goal marker on the table. The cube edge is visibly near the goal tag, confirming the 6 cm success threshold is met physically.

Chapter 5

Discussion

This chapter explains combined results of simulation and real robot performances in terms of three research questions described in Chapter 1. Section 5.1 presents an overall interpretation of the findings. Section 5.2, 5.3 and 5.4 analyze each research question in detail. Section 5.5 discusses what the sim-to-real gap analysis means for the results. Section 5.6 provides comparison of these findings with existing related work. Section 5.7 highlights the limitation of the current approach.

5.1 Interpretation of Key Findings

The experiments conducted in this thesis led to three key findings. First, both sparse and dense reward functions obtain the same (79%) final success for the push-to-goal task. The differences are observable solely in training, not in the final performance. Second, the domain randomization drops the success rate in simulation by 12 percentage points, but it enables real robot deployment that exceeds the simulation baseline for aligned configurations. Third, the main limitation of real robot deployment is not a physics transfer, but generalization across space. The policy demonstrates reliable performance along the training axis, but it fails for lateral configurations outside the training distribution. Overall, these results imply that the most important design decisions in sim-to-real pipeline engineering are the training distribution coverage and the choice of evaluation metric, instead of the specific reward formulation used.

5.2 Conditions Under Which Sparse Reward Achieves Comparable Performance

The fact that Condition 1 (sparse reward) and Condition 2 (dense reward with no DR) achieved the same final success rate is the most theoretically interesting finding from simulation. Generally in RL literature, sparse rewards are considered problematic in exploration in large state spaces, because the probability of randomly reaching a rewarding state decreases rapidly as the task becomes more complex [26]. A common engineering response is a reward shaping, where intermediate signals are added to help guide the policy toward the goal [20]. The result experienced in this thesis that shaping is not necessary, is specific to the constrained geometry of a pushing task and to the properties of the SAC algorithm.

The push-to-goal workspace is 40 cm by 60 cm and success radius is 5 cm. It means that goal takes around 3% of the whole workspace. Despite the relatively small size of the target, the task has a constrained geometry. The object and goal are confined to a two-dimensional surface, which substantially reduces the effective exploration space compared to higher-dimensional manipulation tasks. This geometric structure makes the exploration problem for sparse rewards less severe than it would be in a more complex task or less-structured workspace [26]. Moreover, SAC's large replay buffer allows it to store up to a million of transitions. Over the 2,000,000 timesteps it accumulates a sufficient number of sparse reward signals to make policy learn successfully, even without shaped intermediate rewards.

It is seen on training curves that the dense reward condition converges faster and smoother, which confirms theoretical prediction that reward shaping accelerates the learning [20]. However, the essential point here is that in this specific task, the advantage of dense rewards is only faster training, not better final performance. This distinction should be considered by practitioners in the choice between sparse and dense rewards designs, because dense rewards require extra design parameters, which can cause reward hacking [16].

5.3 Why Domain Randomization Reduces Simulation Performance

The fall of success rate in simulation from 79% to 67% when the domain randomization was applied is a direct consequence of trade-off between specialization and generalization, which is fundamental to machine learning [15], [22]. A policy trained without domain randomization can learn behaviour that is precisely tuned for a fixed conditions of the simulation, taking advantage of the friction coefficient, exact puck mass, and exact noise model to succeed at maximum on the evaluation environment. When domain randomization is introduced and it randomizes these parameters at the beginning of every episode the policy can no longer rely on any specific parameter configuration, it must learn behavior that achieves acceptable performance across the whole randomization distribution.

From the perspective of value function learned by SAC, domain randomization effectively expands the distribution of the environment over which the Bellman consistency constraint must be satisfied. It means that learned Q-function must be precise throughout the whole distribution of randomized parameters. As a result, it brings additional variance into gradient estimates, and it prevents the policy from converging to a sharply tuned strategy [13], [14]. The result is a policy that performs relatively well across different physics configurations, instead of performing perfectly in one particular task. This is an intended engineering choice, whose advantage is clear in a real-world experiment rather than in simulation results.

This mechanism also explains why policy with domain randomization completes succeeded episodes slightly slower than policy without domain randomization (48.7 and 48.2 steps respectively). The behavior learned with domain randomization is more conservative, and involves more cautious approach trajectories and softer contact forces than policy optimized for fixed conditions. As a result, the task is completed slower, but in a more stable and reliable way.

5.4 Simulation Success Rate as a Misleading Metric

One of the most essential practical findings in this thesis is demonstrating that success rate in simulation is not a reliable indicator of how well policy will perform in the real robot. The policy with domain randomization showed the lowest performance in simulation, but it was the only policy suitable for deployment to the real robot. It achieved an 80% success rate in a straight-line motion and surpassed its simulation performance of 67%. This comparison should be interpreted with caution, however, as the real robot evaluation used a 6 cm success radius compared to 5 cm in simulation — a threshold 44% larger in area — to account for calibration uncertainty. The results are best understood as comparable rather than strictly superior performance. The Condition 1 and Condition 2 policies that achieved 79% success rate in simulator were not tested on the robot because they were overfit to the exact conditions of simulation and were expected to fail immediately on the hardware, where values of mass and friction are different.

These results broadens earlier ideas in the domain randomization literature. Peng et al. [22] showed that policies trained in simulation can successfully be transferred to a real robot when dynamics are randomized, but they did not directly compare domain randomization policy real-world performance and its simulation score. Chebotar et al. [27] demonstrated that transfer can be improved with adapting simulation randomization to real-world data. It implicitly assumes that static simulation evaluation does not fully predict real performance. The present work compares directly and highlights it clearly: the policy with the lowest success rate in simulation, is the best on the real robot. This results means that real robot testing, even though on a small scale, as it is done in this work, is a necessary component of any robotics research aimed at the physical deployment, instead of considering simulation result as a final evaluation criterion.

5.5 Implications of the Sim-to-Real Gap

The detailed results from the real robot testing showed that the sim-to-real gap for push manipulation has a specific spatial structure. For straight-line configuration that matches the main training distribution, the gap is effectively negative, the real robot shows significantly better performance than simulation baseline. This result implies that physical push dynamics including contact behavior, object motion, friction, are precisely captured in combination of MuJoCo physics engine along with conservative domain randomization strategy. This outcome matches with classical mechanics of planar pushing, where object movement under sustained contact is highly sensitive to friction, but still predictable when the friction is well-characterized [33]. The 2 mm observation noise model, calibrated with the AprilTag detection system, also matches well, because systematic observation errors were not identified in successful episodes.

In contrast, the gap is substantial and positive for lateral configurations, where the policy achieved 0% success rate from ± 5 cm offsets. This is not a failure of physics transfer, but rather a failure in a coverage of training distribution. The training distribution did not provide enough coverage of diagonal push configurations, so policy could not develop a reliable lateral push strategy. This distinction is crucial for further research. The appropriate approach is not to make simulation more accurate physically, but to expand the range of configurations in training, so the policy can learn to handle these off-axis situations [30].

The final-approach stall represents the real physics-related sim-to-real gap. The compliance of the Cartesian impedance controller decreases the effective force per control step compared to simulation, where commanded joint velocities translate directly into end-effector forces without compliance losses. This misalignment can be addressed by including controller compliance as a randomized parameter in the domain randomization strategy [27], or by training using a smaller action scale that expects the compliance effect.

5.6 Comparison with Related Work

The result of sim-to-real transfer obtained in this work is consistent and comparable to wider domain randomization literature. OpenAI et al. [15] showed successful transfer for dexterous in-hand manipulation using large randomization of multiple parameters at the same time, and large-scale distributed training setup. In comparison, this work demonstrates that targeted randomization of three physical parameters is enough for push manipulation on a standard workstation, assuming that overhead of large-scale domain randomization may not be mandatory for simpler planar manipulation tasks. Another method of bridging the sim-to-real gap without large randomization is randomized-to-canonical adaptation, which trains a separate network to convert real observations into canonical representation as in simulation [32]. Although this approach is effective for vision-based tasks, it introduces additional network complexity, which is not required by the state-based pipeline used in this work.

In contrast with approaches that use Hindsight Experience Replay (HER) for sparse reward manipulation [16], this work shows that particularly for the push task, HER is not necessary for sparse reward convergence when using SAC with a large enough replay buffer. This complements, and does not contradict the HER results, HER is still an appropriate approach for tasks with sparser rewards or multi-goal settings [30], while the present result highlights the conditions under which the standard SAC is enough.

The comparison of dense and sparse rewards provided here fills the specific gap in the literature. While in earlier work, DDPG-based methods [24] and TD3 [25] have been evaluated under conditions of dense rewards for manipulation tasks, and PPO [28] has been evaluated under sparse rewards, direct comparison between two reward types under identical training conditions are not common. The present ablation study provides this controlled comparison keeping the same algorithm, network architecture, hyperparameters, and training budget particularly for a push manipulation task. The result showed that choice between them does not affect the final

performance in these settings.

The fact that policy completes real robot episodes faster (27.1 steps) than simulation episodes (48.7 steps) matches with findings in the domain randomization literature. Randomized policy usually acts conservative under the worst-case variations in simulation, but shows better performance in a more predictable real environment. Although this effect has been observed qualitatively in earlier sim-to-real studies [22], it is rarely measured directly. This work's results provide a clear example of this phenomenon.

5.7 Limitations of the Current Approach

Lateral generalization boundary. The most significant limitation is policy's restricted ability to generalize laterally. Total failure occurred for large Y-offsets of ± 5 cm and the high-variance outcome were observed for ± 2 cm, indicating that the policy's effective operating range is narrower than the whole workspace. This limitation is related to training distribution, not to algorithm, and it can be addressed by increasing the training goal distribution to include diagonal push configurations more uniformly.

Final approach stall. In 30% of straight-line configurations, the policy constantly stalls 6–8 cm away from the goal. It happens because SAC value flattens near the goal and the impedance controller compliance. The SAC policy joins the area where multiple actions appear approximately equivalent, sending small action magnitudes that are not sufficient to overcome the table friction when weakened by the impedance compliance. Neither the reward function nor domain randomization strategy was designed specifically to solve this close-range precision problem.

Control frequency mismatch. The real robot was operated at 10 Hz, while simulation was trained at 20 Hz. It means that every step of a real robot lasts two times longer, and it affects how policy's learned action timing maps to physical object dynamics. Despite this mismatch, the policy was transferred successfully, which shows that it can handle moderate frequency variations, but it is still an unmeasured sim-to-real factor [29].

Object geometry mismatch. The cylindrical puck was used in simulation, however in a real-world experiment the solid cube was used. A cube's flat faces can provide more consistent contact with the end-effector, but its edges can cause contact events when the end-effector is slightly misaligned, which can be seen in Episode 15. For future work, it is recommended to use the identical object geometry in simulation and physical experiments.

Evaluation scale. The real robot testing contained 20 episodes. Despite the fact that results are consistent and failure modes are reproducible across episodes, larger scale evaluation, specifically for the small Y-offset configurations with high variance of outcome, would provide more reliable estimates of policy's generalization.

Chapter 6

Conclusion

6.1 Summary of Contributions

This thesis studied the deep reinforcement learning for robotic push manipulation, followed by deployment to a real Franka Panda robot. The work was organized as an ablation study of three conditions in simulation with sim-to-real transfer of the best candidate policy to the real robot. The key result is that conservative and targeted domain randomization strategy enables sim-to-real transfer with almost no loss for aligned push configurations. At the same time, the results demonstrate that simulation success rate is not a reliable indicator of potential performance in the real experiment. The particular contributions are described below.

Contribution 1: Empirical ablation study with non-obvious findings. A controlled three-condition ablation study was conducted, where the only different factors were reward type and domain randomization. The study produced two non-obvious findings. Firstly, sparse rewards managed to achieve the identical final success rate as dense rewards in simulation (both 79%). Secondly, adding domain randomization strategy decreased simulation performance by 12 percentage points, but enabled the real robot performance of 80% on straight-line episodes, which exceeds the simulation baseline. These results are empirical evidence that indicate that simulation success rate is not sufficient as a sole evaluation criterion for policies intended for sim-to-real transfer.

Contribution 2: Dense progress-based reward function. The design of the reward function combines the distance penalty, a per-step progress reward scaled by a factor of 10, an end-effector proximity penalty, and a terminal success bonus. The progress term was defined as the most significant part, due to providing the informative learning signal during the whole episode, including moments when the object is far from the goal.

Contribution 3: Relative observation design for spatial generalization. The 14-dimensional observation space embraces both absolute positions and relative vectors (from the end-effector to the object, from the object to the goal), which allows policy to learn geometric manipulation based on spatial relationships, instead of memorizing fixed positions. This design is expected to improve policy's ability for generalization to new initial configurations in the training distribution.

Contribution 4: Conservative domain randomization targeting push-specific uncertainty. The developed and validated domain randomization strategy focuses on the main factor that affects pushing: object mass ($\pm 10\%$), surface friction ($\pm 15\%$), and observation noise (2 mm). Instead of using large, random variations, these ranges were chosen to align with the real world conditions. This conservative approach was demonstrated as sufficient for the sim-to-real transfer, at the same time keeping the training stable.

Contribution 5: End-to-end sim-to-real pipeline with real robot validation. A full pipeline was created and tested from customized MuJoCo environment through SAC policy training, AprilTag system based perception, planar homography camera calibration, with mean reprojection error 0.46 cm, and transfer to the physical Franka Panda robot. As a result, pipeline achieved zero-shot policy deployment with 80% success rate in straight-line configurations, and total 50% success rate after 20 evaluation episodes. The primary failure modes were characterized and analyzed.

6.2 Future Work

Expanding lateral generalization. The main limitation discovered in a physical experiment is that policy can generalize effectively only within a range of ±2 cm of lateral offset. This can be addressed by extending training goal distribution, adding diagonal push configurations, or applying the goal-conditioned curriculum that gradually introduces more difficult lateral configurations throughout the training [30]. Also, the use of goal-conditioned reinforcement learning, in which the goal position is provided explicitly as a policy input, could positively influence the spatial generalization. It can condition the policy across the workspace rather than encoding goal position in training distribution [16].

Addressing the final-approach stall. The consistent stalling in the last 6–8 cm from the goal can be addressed by adjusting the reward function to make it give stronger incentive for progress when the object is close to the goal. Another way is adding a special training stage, which is focused on a close-range completion. Also, it may help to include the controller compliance as a parameter in domain randomization, so the policy learns better how the real robot behaves [27]. Alternatively, the task can be split: the reinforcement learning policy handles most of movement in the goal's direction, while a simple proportional controller handles close-range precision.

Control frequency alignment. Future sim-to-real deployment training should include matching the actual robot's control frequency, or randomization of control timestep. This approach would prevent the 10 Hz versus 20 Hz mismatch, which was identified as an unmodeled sim-to-real factor in this work [29].

Object geometry consistency. In order to remove the contact mismatch and cube edge drift failure observed in Episode 15, the same object shape should be used in simulation and physical experiment, it can be done by fabricating cylindrical puck for real-world tests, or modifying simulation model to use the cube.

Extension to task-adaptive manipulation. The initial motivation of this thesis was a task-adaptive reinforcement learning framework for single-arm push manipulation that can generalize across different object shapes without additional training, ex-

tending the dual-arm work of Cui et al. [1]. The design of observation space, domain randomization strategy and sim-to-real pipeline created in this work are compatible with task-parameter conditioned policy architectures. The main remaining challenge is the problem in stabilization of multi-task training, which was identified in the early experiments. It would require a larger training budget, modular network architecture, or a curriculum-based schedule for tasks. This is still an open research direction with practical relevance for flexible robotic manipulation systems [27], [31].

Bibliography

[1] Y. Cui, S. Han, and D. Liu, "Task-Adaptive Deep Reinforcement Learning for a Dual-Arm Robot," *IEEE Trans. Autom. Sci. Eng.*, 2025, doi: 10.1109/TASE.2024.3352584.

[2] W. Zhao, J. P. Queralta, and T. Westerlund, "Sim-to-Real Transfer in Deep Reinforcement Learning for Robotics: A Survey," in *Proc. IEEE Symp. Series Comput. Intell. (SSCI)*, 2020, pp. 737–744, doi: 10.1109/SSCI47803.2020.9308468.

[3] A. A. Shahid, L. Peternel, and E. Coronado, "Continuous Control for a Robotic Grasping Task Using Deep Reinforcement Learning," in *Proc. IEEE Int. Conf. Syst., Man, Cybern. (SMC)*, 2020, pp. 3309–3314, doi: 10.1109/SMC42975.2020.9282951.

[4] J. Hlavaty and A. Kozakova, "SAC-Based Control Strategy Design," in *Proc. IEEE Int. Conf. Commun. Inf. Technol. (COMSCI)*, 2023, pp. 1–5, doi: 10.1109/COMSCI59259.2023.10315824.

[5] Y. Liu, X. Zhang, and H. Wang, "Robot Manipulation of Dynamic Object with Vision-Based Reinforcement Learning," in *Proc. IEEE Int. Conf. Control Robot. (ICCRE)*, 2024, pp. 1–6, doi: 10.1109/ICCRE61448.2024.10589748.

[6] Z. Bai, Y. Chen, and L. Wu, "Deep Reinforcement Learning for Robotic Whip Targeting," in *Proc. IEEE Int. Conf. Simul., Model. Progr. Auton. Robots (SIMPAR)*, 2025, pp. 1–6, doi: 10.1109/SIMPAR62925.2025.10979109.

[7] A. Avhad, R. Desai, and P. Kulkarni, "Adaptive Deep Reinforcement Learning for Robotic Manipulation in Dynamic Environments," in *Proc. IEEE Int. Conf. Data Sci. Netw. Security (ICDSNS)*, 2024, pp. 1–6, doi: 10.1109/ICDSNS62112.2024.10690981.

[8] Z. Li and Y. Wang, "Robot Arm Simulation with Model-Free Reinforcement Learning," in *Proc. IEEE Int. Conf. Artif. Intell. Comput. Appl. (ICAICA)*, 2021, pp. 1–5, doi: 10.1109/ICAICA52286.2021.9498063.

[9] S. Vadlamudi and P. Lakshmi, "Reinforcement Learning for Robotic Arm in Ball-Balancing Task," in *Proc. IEEE Int. Conf. Comput. Commun. Cyber Security (IC3SE)*, 2024, pp. 1–5, doi: 10.1109/IC3SE62002.2024.10593154.

[10] M. Elsaman, A. Hassan, and K. Elbayoumi, "Deep Reinforcement Learning for Continuum Robot Manipulation," in *Proc. IEEE Int. Conf. Novel Intell. Lead. Emerg. Sci. (NILES)*, 2025, pp. 1–6, doi: 10.1109/NILES68063.2025.11232046.

[11] A. Kim, F. Bonsignorio, and E. Coronado, "Reinforcement Learning Environment for a Mobile Manipulator Using Robo-gym," in *Proc. IEEE Int. Conf. Robot Comput. (IRC)*, 2022, pp. 338–343, doi: 10.1109/IRC55401.2022.00056.

[12] M. Farooq and M. Iqbal, "Survey of Reinforcement Learning for Optimization in Automation," in *Proc. IEEE Int. Conf. Inf. Sci. Commun. Technol. (ICISCT)*, 2020, pp. 1–7, doi: 10.1109/ICISCT50599.2020.9351502.

[13] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor," in *Proc. 35th Int. Conf. Mach. Learn. (ICML)*, vol. 80, pp. 1861–1870, 2018. [Online]. Available: `https://proceedings.mlr.press/v80/haarnoja18b.html`

[14] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, and S. Levine, "Soft Actor-Critic Algorithms and Applications," *arXiv preprint arXiv:1812.05905*, 2018. [Online]. Available: `https://arxiv.org/abs/1812.05905`

[15] OpenAI, M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, J. Schneider, S. Sidor, J. Tobin, P. Welinder, L. Weng, and W. Zaremba, "Learning Dexterous In-Hand Manipulation," *Int. J. Robot. Res.*, vol. 39, no. 1, pp. 3–20, 2020, doi: 10.1177/0278364919887447.

[16] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba, "Hindsight Experience Replay," in *Advances in Neural Inf. Process. Syst. (NeurIPS)*, vol. 30, pp. 5048–5058, 2017. [Online]. Available: `https://papers.nips.cc/paper/7090-hindsight-experience-replay`

[17] E. Todorov, T. Erez, and Y. Tassa, "MuJoCo: A Physics Engine for Model-Based Control," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, pp. 5026–5033, 2012, doi: 10.1109/IROS.2012.6386109.

[18] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-Baselines3: Reliable Reinforcement Learning Implementations," *J. Mach. Learn. Res.*, vol. 22, no. 268, pp. 1–8, 2021. [Online]. Available: `https://jmlr.org/papers/v22/20-1364.html`

[19] D. Wang, T. Zhou, and C. Wang, "Multi-Stage Reinforcement Learning for Non-Prehensile Manipulation," *IEEE Trans. Ind. Electron.*, 2024, doi: 10.1109/TIE.2024.3408522.

[20] A. Y. Ng, D. Harada, and S. Russell, "Policy Invariance Under Reward Transformations: Theory and Application to Reward Shaping," in *Proc. 16th Int. Conf. Mach. Learn. (ICML)*, pp. 278–287, 1999.

[21] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement Learning in Robotics: A Survey," *Int. J. Robot. Res.*, vol. 32, no. 11, pp. 1238–1274, 2013, doi: 10.1177/0278364913495721.

[22] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-Real Transfer of Robotic Control with Dynamics Randomization," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pp. 3803–3810, 2018, doi: 10.1109/ICRA.2018.8460528.

[23] E. Olson, "AprilTag: A robust and flexible visual fiducial system," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pp. 3400–3407, 2011, doi: 10.1109/ICRA.2011.5979561.

[24] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous Control with Deep Reinforcement Learning," in *Proc. Int. Conf. Learn. Representations (ICLR)*, San Juan, Puerto Rico, 2016. [Online]. Available: `https://arxiv.org/abs/1509.02971`

[25] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing Function Approximation Error in Actor-Critic Methods," in *Proc. Mach. Learn. Res.*, vol. 80, pp. 1587–1596, 2018.

[26] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, MA: MIT Press, 2018.

[27] Y. Chebotar, A. Handa, V. Makoviychuk, M. Macklin, J. Issac, N. Ratliff, and D. Fox, "Closing the Sim-to-Real Loop: Adapting Simulation Randomization with Real World Experience," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, Montreal, Canada, 2019, pp. 8973–8979, doi: 10.1109/ICRA.2019.8793789.

[28] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," *arXiv preprint arXiv:1707.06347*, 2017. [Online]. Available: `https://arxiv.org/abs/1707.06347`

[29] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI Gym," *arXiv preprint arXiv:1606.01540*, 2016. [Online]. Available: `https://arxiv.org/abs/1606.01540`

[30] M. Plappert, M. Andrychowicz, A. Ray, B. McGrew, B. Baker, G. Powell, J. Schneider, J. Tobin, M. Retkowski, W. Zaremba, and P. Abbeel, "Multi-Goal Reinforcement Learning: Challenging Robotics Environments and Request for Research," *arXiv preprint arXiv:1802.09464*, 2018. [Online]. Available: `https://arxiv.org/abs/1802.09464`

[31] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-End Training of Deep Visuomotor Policies," *J. Mach. Learn. Res.*, vol. 17, no. 39, pp. 1–40, 2016.

[32] S. James, P. Wohlhart, M. Kalakrishnan, D. Kalashnikov, A. Irpan, J. Ibarz, S. Levine, R. Hadsell, and K. Bousmalis, "Sim-to-Real via Sim-to-Sim: Data-Efficient Robotic Grasping via Randomized-to-Canonical Adaptation Networks," in *Proc. IEEE/CVF Conf. Comput. Vision Pattern Recognit. (CVPR)*, Long Beach, CA, 2019, pp. 12627–12637, doi: 10.1109/CVPR.2019.01291.

[33] M. T. Mason, "Mechanics and Planning of Manipulator Pushing Operations," *Int. J. Robot. Res.*, vol. 5, no. 3, pp. 53–71, 1986, doi: 10.1177/027836498600500303.

[34] D. Abekenov, "franka-push-rl: SAC policy for robotic push manipulation with sim-to-real transfer," GitHub, 2026. [Online]. Available: `https://github.com/Dal3l/franka-push-rl`