Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Reinforcement Learning from Simulation for Real Robot Manipulation

Doctoral thesis
with the aim of achieving a doctoral degree at the
Faculty of Mathematics, Informatics and Natural Sciences
Department of Informatics
of Universität Hamburg

submitted by
Lin Cong
Matrikelnr.: 7135320
Hamburg, 2021

Doctoral Committee Member:

Prof. Dr. Stefan Wermter (Chair)
Prof. Dr. Jianwei Zhang (Examiner)
Prof. Dr. Janick Edinger (Examiner)

Date of Disputation:

27/01/2022

Abstract

A broad application of robots in both industry and everyday life needs an agile and stable control strategy when faced with variable circumstance. Among all the everyday tasks, object pushing and grasping are two basic manipulations. The robot control process can be divided into perception and decision-making, which are usually dealt with separately in traditional control frameworks. Especially when confronted with multimodal sensing, different mathematical methods may be necessary to extract features from data of different modalities. The prevalence of deep learning brings a lot of new thoughts to robot research fields. The flexibility and scalability of the neural networks promote the end-to-end learning methods, which can do the perception and control together. The main content of this thesis is the study of pushing and dexterous grasping with deep reinforcement learning.

This thesis starts from modeling the planar object pushing. Recurrent model predictive path integral, a sampling-based method, is proposed accordingly to push the unknown object to a target pose. Real-world experiment results show that the policy can be well transferred to the real world without any fine-tuning after training only in simulation. Comparatively, the policy trained with reinforcement learning cannot be easily generalized to a differently shaped object, but the agent can make intelligent pushing side switching decisions according to the relative positions of the current object pose and target pose.

In order to endow the reinforcement learning policy with generalization ability to different shaped objects, the necessity of taking images as input is analyzed. A vision-proprioception model is proposed, which can extract pose and shape features from an object mask and fuse with the robot end-effector position into a low-dimensional latent space as the state of the agent. Through domain randomization, the policy trained in simulation is again well transferred to the real world. However, the real robot setup is very complex because of the sensitive light requirement.

The question is raised: whether the robot can still push if we simplify the robot setup. However, taking high-dimensional RGB images as the input increases the perception difficulty. A novel self-supervised attention mechanism is proposed to highlight task-related information from the images, which can help learn the valuable representation. Then the attention module is integrated into the reinforcement learning framework. The pre-trained pushing model from the previous chapter is used as the teacher policy to guide the agent during the training process.

The last part of this thesis continues to study reinforcement learning through a dexterous grasping case. A multimodal grasping policy that fuses joint torques, joint angles and tactile sensing is trained in the simulator. To reduce the exploration space of the agent, postural synergies are generated through principal component analysis on a dataset covering various grasp types. The hand motion is planned in the latent space. Through sensor mapping, the trained policy is successfully transferred to a real robot platform.

Zusammenfassung

Eine breitere Verwendung von Robotern sowohl in der Industrie wie auch im Alltag einiger Menschen benötigt dynamische und stabile Kontrollmechanismen, wenn man die Roboter komplexen und sich verändernden Umgebungen aussetzt. Unter allen alltäglichen Aufgaben sind das Verschieben von Objekten und das Greifen zwei essenzielle Manipulationen. Der Roboterkontrollprozess kann in die Bereiche Wahrnehmung und Entscheidungsfindung aufgeteilt werden, welche normalerweise separat von traditionellen Kontrollframeworks behandelt werden. Besonders wenn man mit multimodaler Wahrnehmung konfrontiert ist, mögen unkonventionelle mathematische Methoden nötig sein, um signifikante Merkmale der unterschiedlichen Modalitäten zu extrahieren. Die Verbreitung von Deep Learning bringt eine Reihe neuer Ideen in den Forschungsbereich der Robotik. Die Flexibilität und Skalierbarkeit künstlicher neuronaler Netze unterstützt End-to-End Learning Methoden, welche Wahrnehmung und Kontrollmechanismen vereinen. Der Hauptinhalt dieser Arbeit ist die Untersuchung von Schieben und geschicktem Greifen mit Deep Reinforcement Learning.

Die Arbeit beginnt mit der Modellierung von planarem Schieben von Objekten. Der Recurrent Model Predictive Path Integral, eine Sampling-basierte Methode, wird anhand der Aufgabe, ein unbekanntes Objekt an eine Zielpose zu schieben, vorgestellt. Experimente in der Echtwelt belegen, dass die so nur im Simulator gelernten Strategien ohne Anpassungen gut in die Realität übertragen werden können. Im Vergleich dazu kann eine Strategie, welche mit Reinforcement Learning gelernt wurde, nicht so leicht für andersförmige Objekte generalisiert werden. Jedoch ist der Agent in der Lage, intelligente Entscheidungen zum Wechseln der Schiebeseite basierend auf den relativen Objekt- und Zielposen zu treffen.

Die Notwendigkeit zur Verwendung von bildbasiertem Input wird analysiert, um die Reinforcement Learning basierte Strategie mit einer Generalisierung über diverse Objektformen hinweg auszustatten. Ein bildbasiertes Wahrnehmungssystem, welches Positions- und Formmerkmale aus einer Objektmaske extrahiert und mit Informationen über die Pose des Roboterendeffektors in einem niederdimensionalem latenten Raum zu einer Zustandsrepräsentation fusioniert, wird präsentiert. Durch Randomisierung in der Trainingsumgebung ist die gelernte Strategie gut für den Einsatz auf einem echten Roboter transferierbar. Besonders aufgrund einer hohen Empfindlichkeit des Systems in Bezug auf die Ausleuchtung der Umgebung ist der Versuchsaufbau jedoch sehr komplex.

Folgend wird die Frage gestellt, ob ein Roboter die Aufgabe trotz einer Vereinfachung des Aufbaus noch lösen kann. Generell ist der Aufwand der Verarbeitung von hochdimensionalen RGB-Bildern als Input höher. Ein neuartiger selbstkontrollierender Aufmerksamkeitsmechanismus wird präsentiert, welcher die aufgabenrelevanten Informationen hervorhebt, was hilft, eine aussagekräftige Repräsentation zu lernen. Der Mechanismus wird in das Reinforcement Learning Framework integriert. Das vortrainierte Netzwerk aus dem vorherigen Kapitel wird als Lehrstrategie verwendet, um den Agenten im Trainingsprozess zu führen.

Der letzte Teil der Arbeit setzt anhand von geschicktem Greifen mit der Untersuchung von Reinforcement Learning fort. Eine multimodale Greifstrategie, welche Ge-

lenkdrehmomente, Gelenkwinkel und taktile Wahrnehmung vereint, wurde im Simulator trainiert. Um den Exploration Space des Agenten zu reduzieren, werden Synergien mithilfe von Hauptkomponentenanalyse auf einem Datensatz, welcher diverse Greifstrategien enthält, generiert. Die vollständige Handbewegung wird im latenten Raum geplant. Durch Sensor Mapping wird die trainierte Strategie erfolgreich auf eine Roboterplattform in der Realität übertragen.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Throughout the history of humankind, one accidental revolution is enough to change the course of human history. What is exciting about living in the current era is that we are going through another revolution: the Deep Learning Revolution, transforming the AI research. Deep learning is reshaping our life in many important ways. Self-driving cars empowered with deep learning knowledge are changing the way of traveling; AlphaFold [126] is a new scientific breakthrough, which can predict 3D models of protein structures and has the potential to accelerate research in most fields of biology; GPT3 [21], which can be used in conversation and text completion, powers the next generation of Apps. Besides, deep learning also brings new ideas to the field of robot control. The problem of intelligent robot control can be taken as a *decision-making process* based on the current environment state to reach the final goal of the specific task. Reinforcement Learning (RL) is the study of how intelligent agents ought to take actions in an environment in order to maximize the notion of cumulative reward [51]. The combination of RL and deep learning, namely Deep Reinforcement Learning (DRL), extends the application scope of traditional RL by using a deep neural network to process unstructured data without manual engineering of the state space.

One of the most notable breakthroughs brought by DRL is AlphaGo, developed by Silver et al. [128]. AlphaGo is a DRL algorithm with an actor-critic structure, which learns both a value network (critic) and a policy network (actor) through self-play games. Before the success of AlphaGo, the game of Go was regarded as a challenging problem in machine learning and was expected to be out of research for the technology of the time. In October 2015, it became the first program to defeat a professional human player, and several months later, it defeated the top human player (Lee Sedol) at the time. Since then, the potential of DRL to solve highly complicated strategy problems entered into researchers' horizons.

When using the traditional Model Predictive Control (MPC) method to control the robot, the performance heavily relies on the accuracy of the model. However, because of the stochastic nature of the actual physical world, an accurate dynamic model of a robot or robot-object interaction during the control process is hard to get. Researchers

have proposed various modeling methods to overcome the uncertainty during the control process, including analytical and data-driven models. Accordingly, choosing different control methods for different models is also critical to reach a satisfying control effect, which is always not trivial. One significant advantage of using DRL for robot control is that the control process is end-to-end, which means the algorithm is model-free, no explicit dynamic model is necessary. The agent can directly map the current state to actions.

Figure 1.1: Scalable RL training with multiple robots in the real world [59].

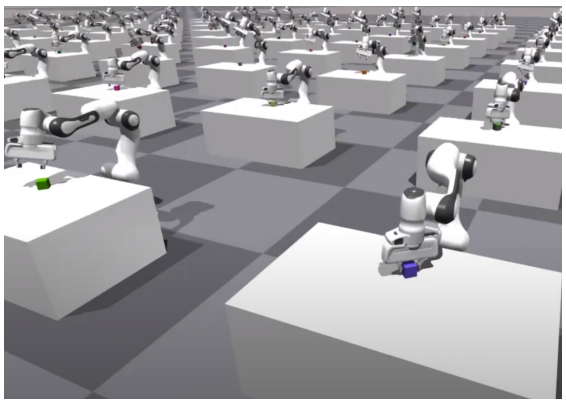
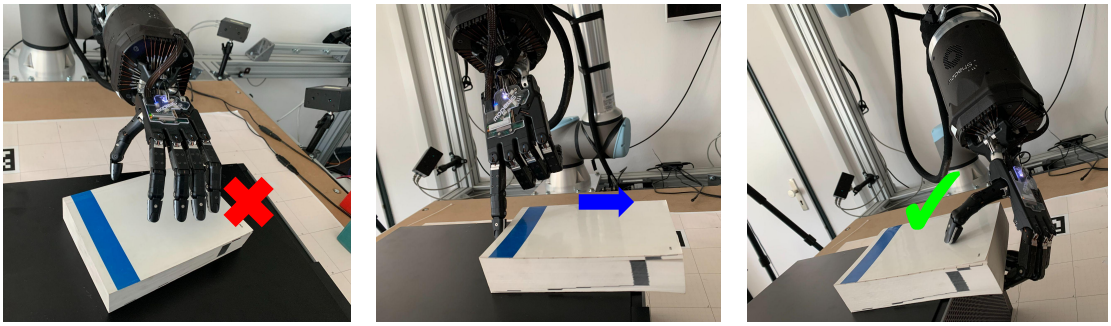


Figure 1.2: Isaac Gym (a learning platform to train policies on GPU) developed by Nvidia [83].

In most of the real robot manipulation scenarios, the model of the environment is unknown. Therefore, the only way to collect samples about the environment is to interact with it. At the beginning of the training process, the actions applied by the agent are random, leading to massively ineffective interactions (One extreme case is all the rewards are zero, the agent learns nothing from the trajectory). Random exploration without any prior model of the environment causes the notorious low sample efficiency of RL. Therefore, training a satisfied agent usually requests large amounts of interaction data, making the training process long and expensive. Especially when the training is

performed on a real robot platform, collecting enough interaction data is always time-consuming and laborious. Kalashnikov et al. [59] propose a scalable DRL framework to train robot grasping, in which a group of robot arms is used together to collect the data (Figure 1.1). One feasible option to reduce the experiment cost and training time is to train the robots in simulation. Training in simulation is faster for two main reasons: 1) execution in a simulator is often faster than execution in the real world; 2) experiment reset in simulation is more convenient and faster. Isaac Gym (Figure 1.2) is a new physics simulation environment for reinforcement learning which can use fewer GPUs to replace thousands of CPU cores. Besides running experiments in parallel to collect more data, the sample efficiency can also be improved by modifying the algorithm. Hindsight Experience Replay (HER) [4] is proposed to increase the sample efficiency for off-policy RL algorithms. However, training in simulation brings a sim-to-real gap, leading to the policy trained in simulation usually being hard to apply in the real world directly. How to bridge this gap is also widely studied in the robotic community.



(a) (b) (c)

Figure 1.3: Using shadow hand to grasp a book. (a) The book is too big to grasp from top. (b) Pushing the book to the edge of the platform. (c) Grasp from a different side.

To summarize, using DRL in robot tasks is a newly arisen research direction. Even though the training usually calls for excessive computation resources, but the potential of RL to solve long-term tasks in the future has emerged. The core content of this thesis is to study the application of DRL in real robot control. Two elementary manipulation tasks are chosen as the research context, which are pushing and grasping. Pushing can be executed as a pre-grasping action, usually for a better grasping pose (Figure 1.3). Both of the two manipulations are regular human daily operations but are still challenging for autonomous robots.

1.2 Contributions

This thesis focuses on training robots to do object manipulation with RL in simulation and then transferring to real robot platforms. The main contributions of this thesis are summarized as follows:

- **New pushing model and the corresponding control method:** A novel self-adapting model for object push is proposed. Both the analytical and data-driven

aspects can be reflected in the model. The model assumes no prior knowledge of the pushing object and can be generalized to different objects by performing online self-adapting according to the real-time interaction trajectories. Further, a sample-based path integral control algorithm is proposed to achieve object pushing on a real robot.

- **Vision-proprioception fusion model for RL:** To encode the visual information into the state of a Markov Decision Process (MDP), a new vision-proprioception model is proposed. The model can fuse all task-related information from the environment (object mask, goal pose mask, and end-effector position) into one low-dimensional vector. Variational Autoencoder (VAE) is used in the model to compress an image mask into a latent space. In order to train the robot pushing objects with RL in a simulator, a Mujoco [134] simulation environment is built based on our UR5 robot platform. One feasible policy is obtained through training in the simulation by taking the feature from the fusion model as the agent's state. A GUI tool is also built to visualize the latent vectors from the VAE.

- **Attention Augmented RL:** Learning a representation of the high-dimensional RGB image and the corresponding control policy at the same time is quite challenging. A self-supervised top-down attention mechanism is proposed to extract task-related regions from an image. The highlight parts of the image are allocated higher weights, enabling the network to pay more attention to the vital part. The attention mechanism can be pre-trained and easily integrated into an RL framework, improving the training performance to a large degree.

- **Multimodal RL for dexterous grasping:** A multimodal RL framework is proposed for multi-finger grasping. Different modalities (including joint angles, joint torques, tactile sensing) are tested to reach the best perception performance. Principal Component Analysis (PCA) is used to reduce the action dimension and also generate human-like hand motions, which turns out to be effective in compressing the exploration space and avoiding weird hand postures. A sensor mapping method is also proposed to transfer the model to a real robot platform.

1.3 Research Questions

The following problems are studied in this thesis:

- **Pushing Object-Based on Dynamic Model**: How to learn a dynamic object pushing model in simulation which can be used in the real world and also generalized to objects with different physics parameters? Given the dynamic model, how to design a controller which can perform online self-adapting according to the historical interaction trajectory?

- **Visual Pushing with RL**: How to endow the robot with intelligent side switching ability, and at the same time can generalize to different objects. How to deal with the high-dimensional image input in the problem setting of goal-based RL?

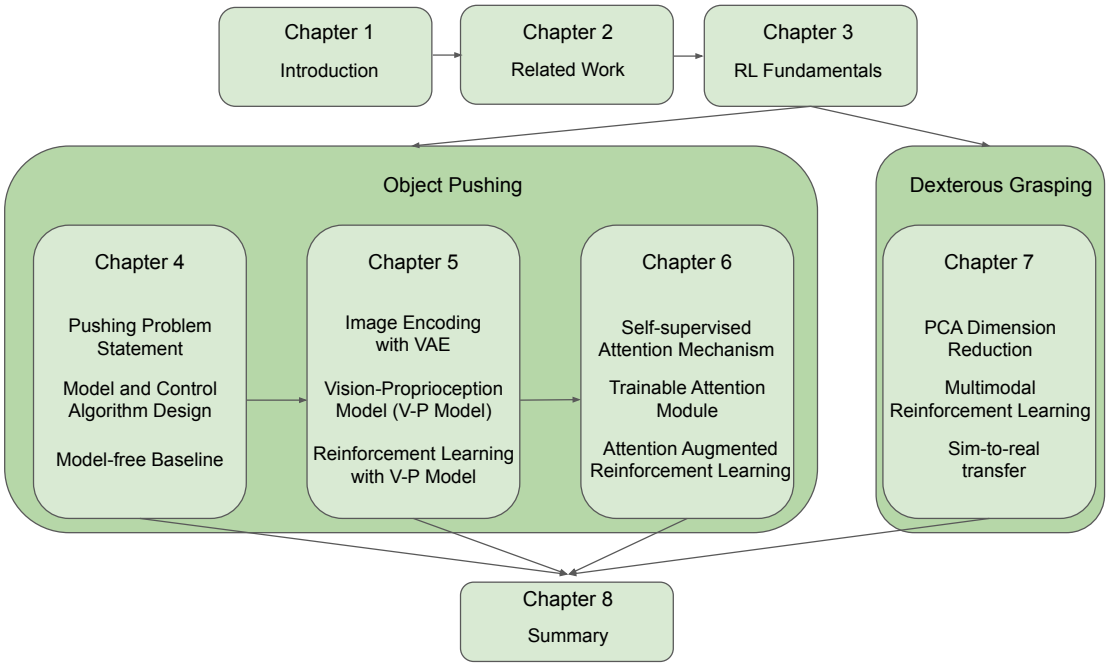


Figure 1.4: Structure of the thesis. The main theoretical contributions of each chapter are listed.

- **Attention Augmented RL framewrok**: Is it possible to simplify the robot setup requested by the method proposed in the previous chapter by modifying the algorithm framework? How to integrate the attention mechanism into the RL framework to improve learning performance?
- **Multimodal RL for Dexterous Grasping**: How to train an arm-hand system to do multi-finger grasping with RL in simulation and transfer to a real robot platform? How to increase the robustness of the policy by using multimodal sensor data?

1.4 Thesis Structure

The structure of this thesis is shown in Figure 1.4. The main contents are organized as follows:

- Chapter 1 gives an overview of the thesis from motivation, research question, and contribution.
- Chapter 2 presents the most recent researches related to the thesis.
- As RL is used as the core solution to all the robot tasks in this thesis, chapter 3 details all the necessary math fundamentals and concepts to fully understand the RL algorithms. Besides, some applications of RL in other industries are also introduced.
- Chapter 4 first introduces the problem of the robot planar pushing and some existing solutions, including the analytical motion model and the data-driven model

of object pushing. Then a novel combination of analytical and data-driven models, which can adapt the prediction according to the recent sequence of interaction data, is proposed. Given the dynamic model, a new MPC method is presented accordingly. Apart from the model-based pushing method, an RL policy is also trained as a model-free baseline. After that, how Automatic Domain Randomization (ADR) is applied during the data collection to bridge the gap between real-world and simulation is explained. Finally, the real robot platform setup is described, and both the proposed model and the RL policy are verified on the real robot.

- Chapter 5 tries to solve the pushing problem from pixel inputs. The state input used in chapter 4 lacks object shape and size information, which is essential for the algorithm's generalization to push different objects. A vision-proprioception model is proposed based on Variational Autoencoders, and the corresponding RL framework to train the agent is also presented in detail. At last, to verify the model's performance in the real world, the same real robot set up as in the simulation is built.

- Chapter 6 continues with the visual RL but attempts to solve the pushing problem with a simpler robot setup (getting rid of the transparent table and the bottom camera). Taking the image from the front camera as the only input increases the training difficulty of the agent for two reasons: the network has to learn the high-dimensional representation and the control at the same time; the ground-truth robot state is no longer available directly but has to be inferred from the RGB image. The novel top-down self-supervised attention mechanism, which can pay attention to the task-relevant part of the image, is presented and integrated into the RL framework.

- Chapter 7 presents a multimodal RL algorithm for robot dexterous grasping. Inspired by the idea of fusing visual information and proprioception states and planning in the fusion space proposed in chapter 5, further in-depth research on multimodal RL is performed. The algorithm includes feature fusion and action control in two parts. After enough episodes of training in simulation, the model is verified on a real robot platform.

- Chapter 8 first summarizes the scientific contributions of the thesis, then some limitations of the current state are also analyzed. Finally, an outlook of the upcoming future research is given to close the whole thesis.

1.5 Thesis Related Publications

During the four years of studying, several publications were contributed to different conferences and journals. Most of them are the results of collaboration with colleagues. The works listed here are directly related to the core of this thesis. Till the submission of this thesis, three of them are already published or accepted papers:

- **Lin Cong**, Michael Görner, Philipp Ruppel, Hongzhuo Liang, Norman Hendrich, and Jianwei Zhang. "Self-Adapting Recurrent Models for Object Pushing from Learning in Simulation," In *International Conference on Intelligent Robots and Systems* (IROS) 2020, Las Vegas, USA. (published)

- Hongzhuo Liang, **Lin Cong***, Norman Hendrich, Shuang Li, Fuchun Sun, and Jianwei Zhang. "Multifingered Grasping Based on Multimodal Reinforcement Learning." In *Robotics and Automation Letters* (RA-L). (accepted)

- **Lin Cong,** Yunlei Shi, and Jianwei Zhang. "Self-supervised Attention Learning for Robot Control", In: *International Conference on Robotics and Biomimetics*, (ROBIO) 2021, Sanya, China. (accepted)

One paper is submitted to *Frontiers in Neurorobotics* and under review:

- **Lin Cong**, Hongzhuo Liang,Philipp Ruppel, Yunlei Shi, Michael Görner, Norman Hendrich and Jianwei Zhang. "Reinforcement Learning with Vision-Proprioception Model for Robot Planar Pushing."

Chapter 2

Related Work

DRL can learn high-dimensional visual representations (usually RGB image inputs) for common robotic tasks and the corresponding control policy at the same time. However, trajectory collection and policy exploration via running an agent in the real world is always extremely time-consuming. Training in simulation and transferring the policy to a real robot is an alternative to applying the RL method. The problem is that there is always a gap between the physics simulator and the real world. Recently, many methods have been proposed to bridge the sim-to-real gap in robotic researches.

The components in this chapter are organized as follows: Section 2.1 gives an introduction on the recent progress of bridging the sim-to-real gap. From the perspective of concrete tasks, this thesis takes solving real-world object pushing and dexterous grasping as examples. Therefore, the second and third parts of the chapter briefly summarise the object pushing and dexterous grasping problem and some existing solutions. Because RL is used as the primary method for solving the manipulation task in the thesis, Section 2.2.2 introduces some state-of-the-art RL-based robot control applications. At last, Section 2.4 gives a supplementary introduction to attention mechanism, which is used in chapter 5 to improve the performance of visual reinforcement learning.

2.1 Robot Learning from Simulation to Real Environment

Compared to training robots in the real world, learning in a simulated environment is an easy alternative, which reduces the time spent on the expensive data collection on real robots, and we do not have to consider safety problems during the training process. However, transferring results or policies trained in simulation to a real environment is always challenging due to the gap between visual and dynamics. This reality gap has been addressed in numerous works. Saxena et al. [122] learn to grasp novel objects with simulator rendered objects. Rusu et al. [116] propose using progressive networks to bridge the reality gap, and transfer learned policies from simulation to the real world. Three frequently-used methods are mainly introduced in this section (Figure 2.1): visual domain adaptation, domain randomization (can be used for both domains), and actual data actuator modeling (used for the dynamics domain).

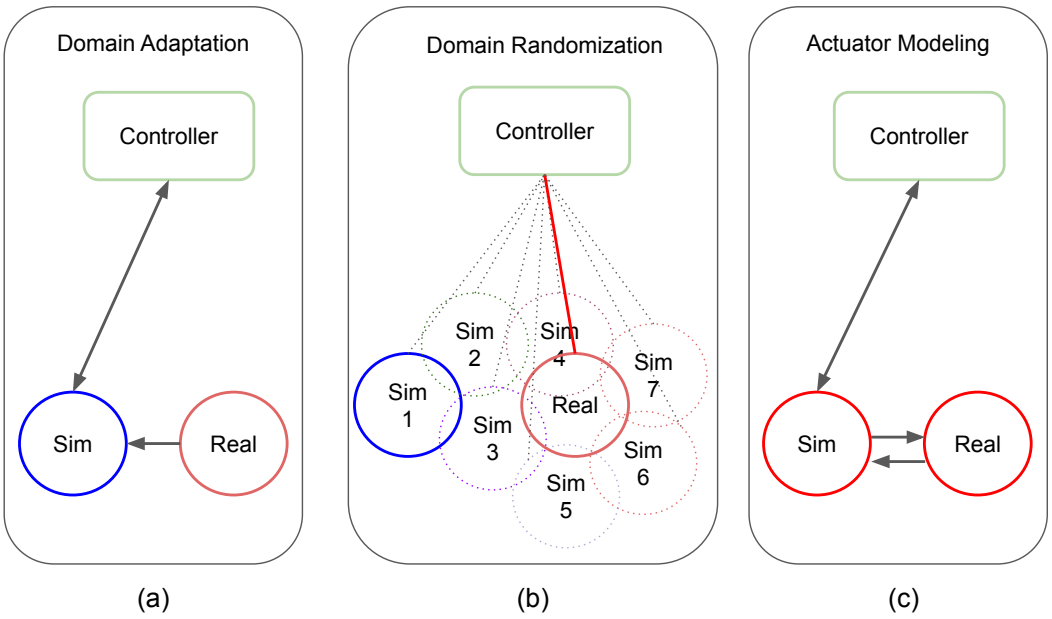


Figure 2.1: Illustration of different sim-to-real transfer methods. (a) The training process in domain adaption usually happens in one specific simulation domain. When applied to real robots, the real domain observation is translated into the simulation domain, where the controller is effective. (b) The controller is trained in various simulation domains, in which the real domain may be one of the variants among them. (c) Modeling the actuator in simulation with real data collected from the real domain to make the simulation and real domains match with each other.

2.1.1 Visual Domain adaptation

Visual domain adaptation is a process that allows a model trained with samples from a source domain to generalize to a target domain [98]. It has been an effective method for solving the problem of adapting vision-based models trained in a source domain to a previously unseen target domain [133], widely used in visual application scenarios like action recognition [29], object detection [145], and image classification [118]. Various approaches have been proposed, including model-retraining [147], model weights adaptation based on the feature distributions of source and target domains [73], and learning invariant features between domains [41, 135]. Typically, large amounts of unlabelled real-world data are necessary to cross the visual reality gap. In [56], a novel approach Randomized-to-Canonical Adaptation Networks is proposed to cross the visual reality gap, which uses no real-world data by translating randomized rendered images into equivalent non-randomized, canonical versions, in which way, real images are also translated into canonical sim images (Figure 2.2). During real robot application, the observation is translated into the canonical domain, in which the agent is trained. However, the real grasping performance relies heavily on the adaptation quality. Recent research has also shown that the learning of domain adaptation and the learning of control policy can be performed in one framework and mutually benefit from each other. Zhu et al. [159] propose a joint learning framework that integrates adversarial feature adaptation and policy mimic together to solve 3D indoor navigation. Ho et al. [48] in-

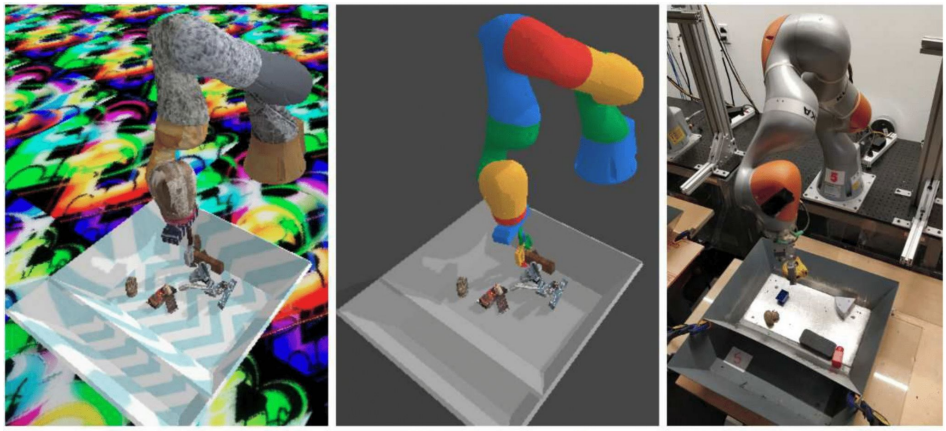


Figure 2.2: An example of visual domain adaptation. (a) The randomized version of simulated environment. (b) The canonical version of the same observation. (c) The real-world observation [56].

troduce the perception consistency loss into the Generative Adversarial Network (GAN) to adapt simulated images to realistic ones. The algorithm is tested on real robot tasks like object grasping, pushing, and door opening.

2.1.2 Domain Randomization

Domain randomization, first proposed in [55] as an idea to develop robust controllers by randomizing all aspects of the simulator, is a valuable technique that randomizes the simulator parameters [117, 144, 154] (can be appearance properties like object textures, a field of view of the camera, lighting conditions; or dynamics parameters [8, 132, 150] like object mass and size, surface friction coefficients, robot joint damping coefficients) in the hopes to improve the transferability of learned policies. In [101], Peng et al. use Domain Randomization to bridge the gap by randomizing the dynamics parameters in simulation during training. Tobin et al. train a real-world object detector using only data on simulated images by randomizing rendering in the simulator (Figure 2.3). OpenAI et al. [6] randomize most of the aspects, including dynamics and visual properties of the simulated environment, in order to learn both a policy and a vision model that generalizes to reality. The policies show excellent robustness on real robots. However, both of the two algorithms rely on a high-performance machine. The former takes approximately 8 hours for the policy to do exploration on a 100 core cluster, and the second policy is trained on a pool of 384 worker machines, each with 16 CPU cores. All the randomization processes mentioned above are independent of the training of the agent. ADR [6] automates the parameter tuning by correlating the parameter randomization with the learning process. However, the correlation needs carefully manual design to avoid getting stuck into suboptimal solutions.

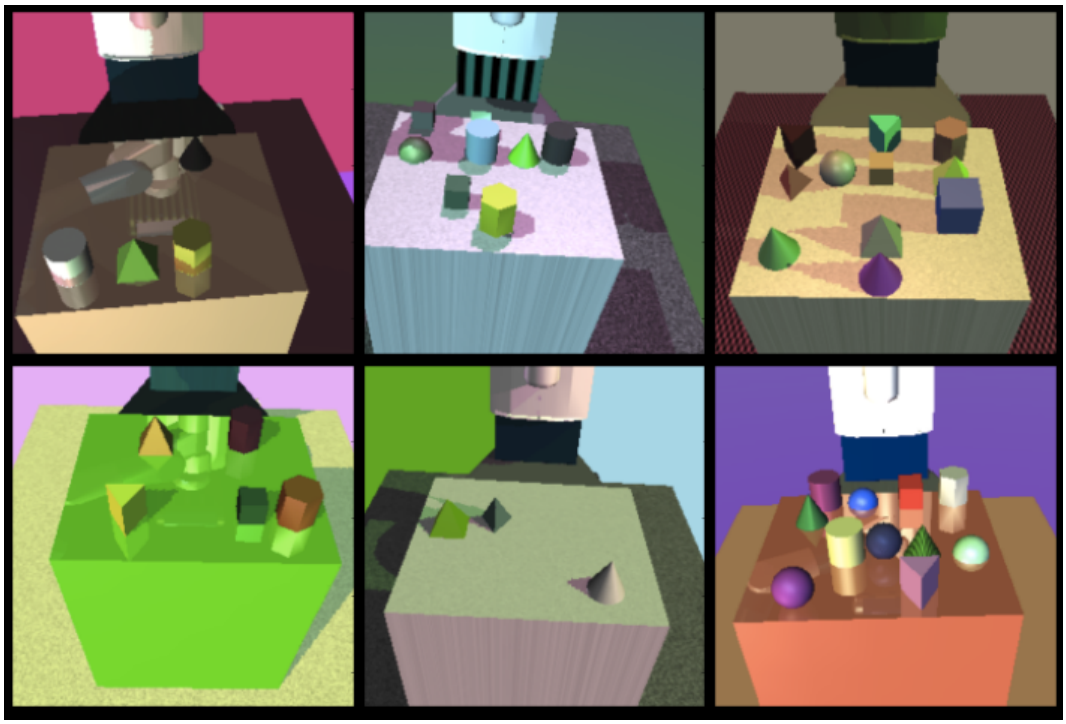


Figure 2.3: An example of visual domain randomization for fetch robot [133].

2.1.3 Actuator Modeling from Real Data

Building dynamic models for simulators from time-serial data of real systems has widely been studied in recent years. Actuator modeling makes the simulator closely match the physical reality by modeling the simulator from real data. In general, the simulator models can be classified into 3 different kinds, namely analytical model [28,52,70,134], data-driven model [74, 120], and hybrid model [1, 39, 45, 54], which is a fusion of the former two methods. Different from visual domain adaptation (Section 2.1.1) or domain randomization (Section 2.1.2), actuator modeling is always used to learn a forward model of the dynamics, which is a mapping or transition distribution from the current state and the action to the next state.

Heiden et al. [45] propose a network-based differentiable hybrid simulator, which can model complex effects involving frictional contacts and viscous friction from real data. They also show that inserting networks can also accelerate model-based control architectures. Golemo et al. [39] introduce a method for training a Recurrent Neural Network (RNN) on the differences between simulator and real robot and use this model to augment the simulator. The neural augmented simulator can train an agent, which transfers better to the real world than the agent trained in the original simulator.

2.2 Object Manipulation

Object pushing is an active research topic in robot manipulation, the essence of which is the single contact underactuated control. Pushing can be taken as a pre-action before

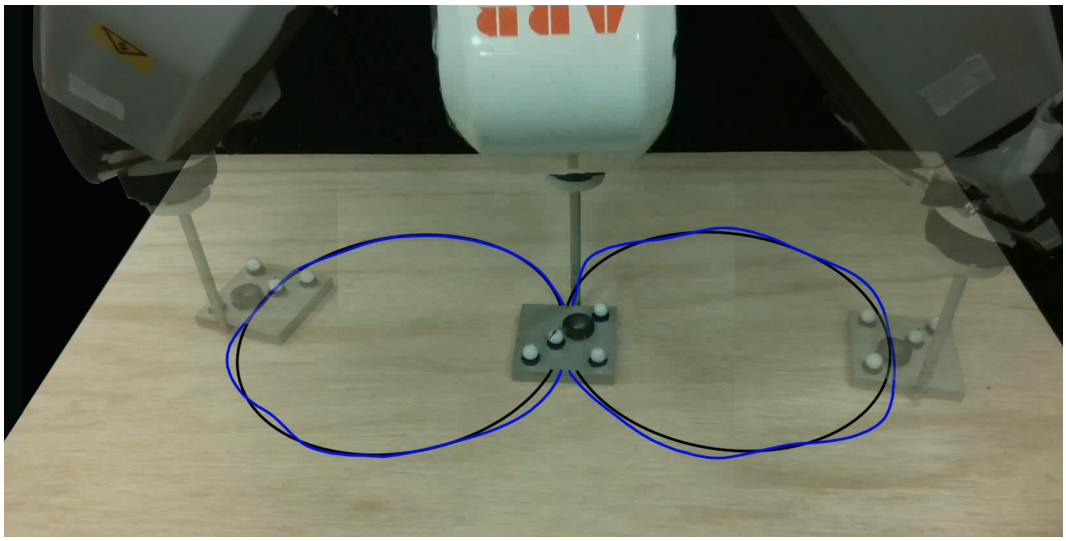


Figure 2.4: One example of object planar pushing research from [14]: using ABB arm to push an object doing trajectory tracking. Black and blue lines represent the desired trajectory and the motion followed by the object's geometric center respectively.

some other action, such as grasping. In general, the solutions to most kinds of robot manipulation can be divided iMPC, is widely used in various robotic tasks. After Mason introduced the modeling problem of pushing decades ago [85], much research has been done around the modeling of planar pushing. Modeling methods for pushing dynamics can be divided into analytical modeling, which is detailed in Section 2.2.1 and data-driven modeling in Section 2.2.1. In [14], Bauza et al. study both methods from the perspective of data-complexity required for control (Figure 2.4).

Given a correct general model, the feedback mechanism of control theory can achieve satisfactory control results. In many robot tasks that involve complex interactions like manipulating a cube with a robotic hand [5] or the problem of using tools [110], it is impossible to model precisely for every contact situation. In these cases, a model-free method such as deep reinforcement learning, which can map from high-dimensional state space to action space directly, is more suitable. However, deep reinforcement learning suffers from the necessity for large amounts of experience data, which is difficult for real robots. Recently some literature has also been combining model-based with model-free methods [58] to improve exploring efficiency.

2.2.1 Planar Pushing

A traditional method to predict an object's motion is to describe the dynamics as an analytical model [62]. An analytical method is always the right choice when the parameters of the specific object are known for sure. Parameters in the analytical model have specific physical meanings. Therefore, the model can be easily transferred to similar problems given specific system parameters. However, getting a stable dynamic model in a real environment is never an easy task. Therefore, generalizing the analytical model to unknown objects is even more difficult, especially when the object's properties, such as

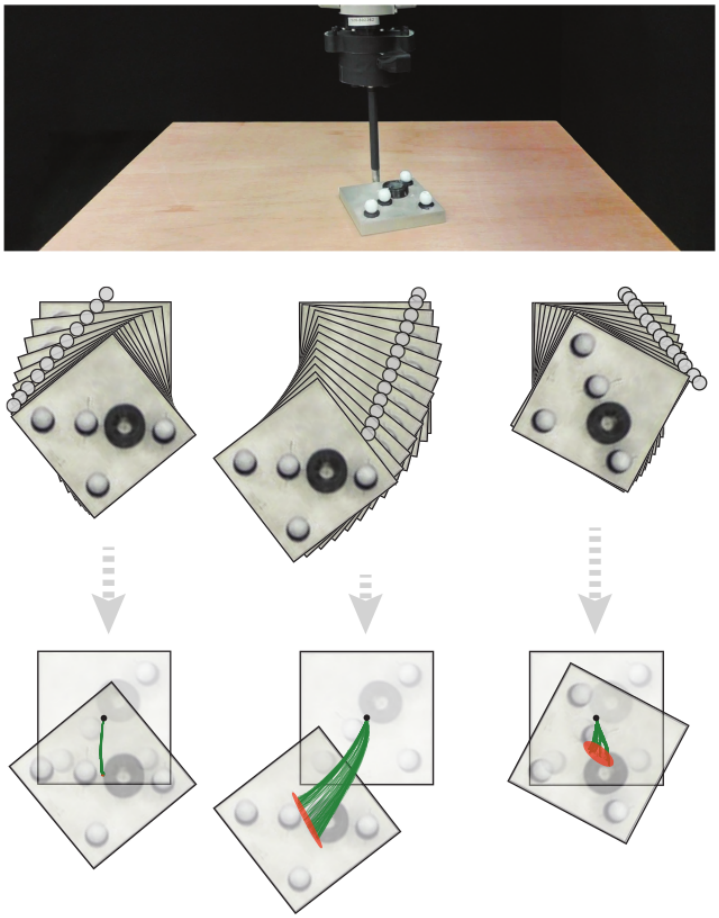


Figure 2.5: Illustration of different distributions under three different pushes (repeated 100 times for each push). The trajectories of the center of mass of the block are shown in green; orange ellipse areas represent the distribution of final poses [15].

friction and inertia, are hard to get.

After the first analytical object pushing model was proposed by Mason [85], which introduces the voting theorem to predict object rotation under an external force, several different practical models emerged. One notable model was proposed by Lynch et al. [82] in 1992, which used the concept of limit surface [40] to model the relation between the pusher and the object velocities.

Pushing in the real world is proven to be a stochastic process, as is shown in Figure 2.5. Data-driven methods [13, 15, 63] for building a dynamic model for complicated nonlinear systems have increasingly attracted attention from researchers. Gaussian Process Regression (GPR) [113] and deep neural network [79] are two typical data-driven modeling methods widely used. The general model-based solution is to build a data-driven model with large amounts of robot-object interaction datasets and then apply MPC as a robot control strategy.

However, collecting real interaction trajectories is very time-consuming, while physics parameters in real situations can only be approximated, both making the model-based methods hard to apply. In [14], Bauza et al. use Gaussian Process to fit the dynamic

model with less than 100 data points from the real environment and get a very stable trajectory tracing performance. The approach is highly efficient. However, the model cannot deal with new objects. In [9], Arruda et al. use GPR and an Ensemble of Mixture Density Networks and get both mean and variance, which is a good evaluation of uncertainty in their predictions. As mentioned in Section 2.2.1, Lynch et al. [82] built the analytical based on the assumption of quasi-static interactions, which neglects inertial forces. To circumvent the approximations made by Lynch et al., Bauza et al. [15] employ a data-driven approach to fit the system dynamics, taking uncertain parameters such as friction coefficients and uneven distributions into consideration.

2.2.2 Dexterous Grasping

Grasping is widely studied in the robot research field. According to the mechanical structure of robot gripper, grasping can be divided into two-fingered grasping and multi-fingered grasping. From the perspective of the controller, due to the high Degree of Freedom (DoF) of dexterous hand joints, multi-fingered grasping is more complicated. Even without considering object dynamics, the grasping pose synthesis of a high DoF dexterous hand is still a challenging task [114]. Brahmbhatt et al. [19] present a novel framework for grasp synthesis. They collect human-demonstrated contact maps on various objects and take the overlapping of contact maps with demonstration data as a target to optimize and refine the grasp candidates with GraspIt! [26]. However, contact maps are hard to generate for unknown objects in the real world. Ficuciello et al. [32] propose a synergy-based strategy; the weakness is that the grasping performance heavily depends on the quality of hand preshaping. They further propose a hand-arm grasp system [31], but the shape of the objects used in their experiments are not diversified enough. Besides, the training is performed in the real world, which is always time-consuming for robotic tasks. Kumar et al. [66] use human hand motion demonstration to initialize a multi-fingered hand in order to reduce the search space. However, pose estimation is required to get an object bounding box, which is already difficult in real robot applications. Wu et al. [140] discretizes the finger action space. The finger motion resolution is low at the beginning and increases during the training process to reduce training difficulty. The hand motion is binary: the robot makes decisions whether to close a finger or open it. However, the agent is easy to train because of the low DoF of the robot hand. Object pose and visual feedback are assumed to be unknown in the above work. In [86] both the contact forces and object pose belong to the robot observation, and they conclude that contact forces can improve the grasping robustness specifically under pose uncertainty. However, the theory lacks real experimental verification, as accurate contact force is hard to get in the real world.

2.3 Reinforcement Learning Based Methods for Robot Control

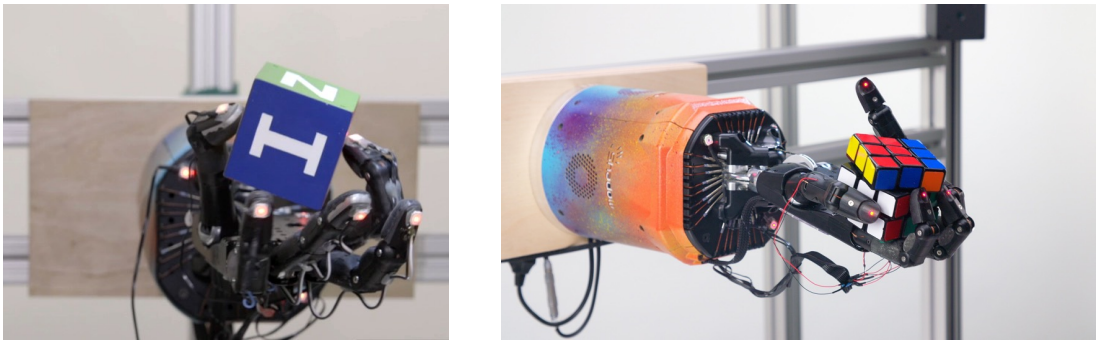
Learning-based methods are becoming more and more popular in robotic manipulation researches recently. Especially deep learning methods, based on artificial neural networks, have reached state-of-the-art performance among standard robot tasks such as grasping from the complex scene, object in-hand manipulation, and components assembling. However, most of the simple dexterous manipulations that are regular tasks for humans in our everyday lives, such as opening a door with the key, flipping through a book, screwing a bottle lid, are still challenging for a robot to perform. The main challenges include the perception of the environment and the corresponding control policy. Human perception of the environment is multimodal, usually including visual, tactile, proprioception, and even audio. The human brain can make accurate decisions through multimodal perception and information fusion, a very complicated neural process. Inspired by humans' neural processing mechanism and the ability to learn from both positive and negative rewards from the exploration to a task, using deep reinforcement learning methods to solve robot manipulation tasks receives more and more attention [10].

RL can be used to get actions directly from the ground truth state [101], or raw pixel images [152]. However, generalization of the model to different manipulation objects is hard for state-based inputs (pose, velocity, joint angles) while extracting useful features such as object shape, size, and the robot's relative position from raw images for a subsequent policy network is always difficult. Convolutional Neural Network (CNN) [68] is always used as an encoder in modern RL algorithms to get spatial features from images. Recent work achieves impressive results on DeepMind Control Suite and OpenAI Gym benchmarks with learning trick-like image reconstruction [42, 146], data augmentation [67] and contrastive learning [67]. However, low data sample efficiency during exploration makes it hard to train directly on real robot platforms. In our work, ADR [5] is applied to bridge the gap between simulation and the real world.

Using dense (pixelwise) visual description as the representation has been proven effective in visual correspondence estimation [25, 123]. Florence et al. [34] propose a self-supervised system to learn consistent dense representation for robotic manipulation. Contrastive learning [44, 94] is always used as an unsupervised learning approach to extract useful representations from high-dimensional data, which has been applied successfully to image recognition [46] and RL [2, 67]. Contrastive loss is also used in Time Contrastive Networks [127], which learn state representation using temporal information from unlabelled demonstration videos. Other methods rely on generative reconstruction loss like VAE [61] and its variations to compress images to latent vectors. In [42] the latent vector representing what the agent sees at each time frame is then fed into recurrent neural networks (RNN) to predict the future. Finn et al. [33] train a deep spatial autoencoder to map camera images to latent features and perform a motion skill directly in the latent space.

2.3.1 In-hand Manipulation

One of the most notable breakthroughs in deep reinforcement learning on robot applications in recent years is from OpenAI [5]. Andrychowicz et al. use RL to train Shadow Dexterous Hand learning manipulating a block (Figure 2.6(a)) and the following research, solving Rubik's cube [5] (Figure 2.6(b)). Their works demonstrate that models trained only in simulation can be used on real robots to solve complex manipulation tasks. Moreover, from the control policy perspective, RNN based policy with memory mechanism trained with ADR shows noticeable sim-to-real transfer performance improvement.



(a) (b)

Figure 2.6: Using reinforcement Learning to do robot in-hand manipulation. (a) Block rotation. (b) Solving Rubik's cube [5].

2.3.2 Quadrupedal Locomotion

Another impressive application of deep RL in robotics is the quadrupedal locomotion control. Lee et al. propose an RL-based solution to use only proprioceptive information to control legged robot walking in challenging natural environments (Figure 2.7). In contrast to the conventional controllers, which are usually based on elaborate state machines, the RL policy shows more robustness and better generalization to unseen conditions.

Figure 2.7: Using reinforcement learning to control quadrupedal over challenging terrain [71].

2.4 Attention Mechanism in Deep Learning

Laskin et al. [67] have also shown that RL consistently achieves better learning performance given ground truth states compared with pixel inputs. One reason is that an image input usually needs a more complicated network structure like CNN to process spatial features, which increases training difficulty. Also, it is challenging to uncover attended regions and eliminate interference from unrelated pixel perturbations without any privileged information on the image. Top-down attention mechanisms are used in [84, 88] to force the agent to focus on task-relevant information. A partially observable ground truth state can also be used to train the agent with image input. For example, Salter et al. [119] propose to use the asymmetric actor-critic method [104] via access to the real state while providing only images for the actor, improving both the robustness and sample efficiency as a result.

The combination of deep learning and attention mechanism has been studied in various research fields, especially in computer vision, Natural Language Processing (NLP), and the cross-field of both, such as image caption [3, 141, 148], and Visual Question Answering [7, 81, 151]. Different kinds of attention module architectures have been proposed accordingly.

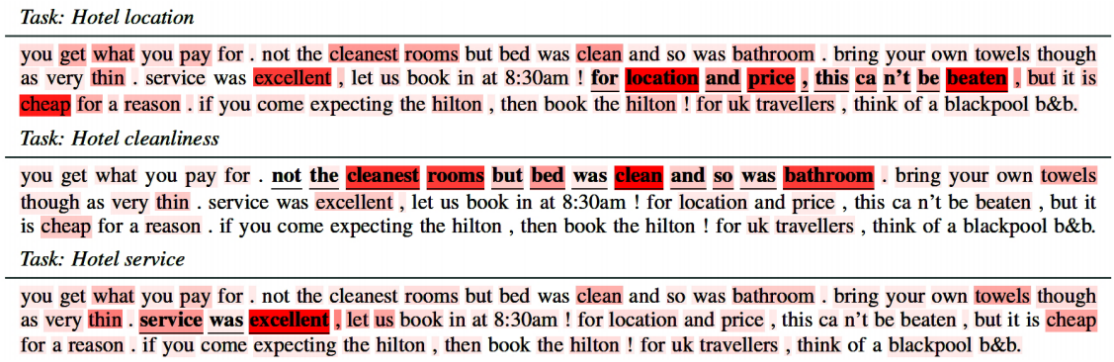


Figure 2.8: Example of attention visualization for a sentiment analysis task [12].

2.4.1 Attention Mechanism in NLP

BERT [30] and GPT-n series [21, 107] (including their variants), which are the most popular and powerful techniques for NLP developed recently, are all transformer-based deep learning models. Before the transformer is invented, the sequence transduction models for NLP are based on complex RNN or CNN, including an encoder and a decoder. The best-performing models also use the attention model to connect the encoder and decoder. The transformer module is proposed in [136], using only the attention module for machine translation.

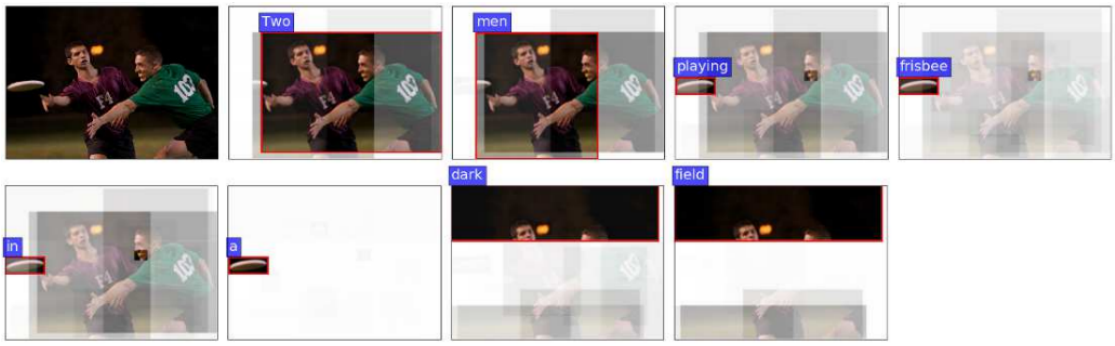


Figure 2.9: An example showing a generated caption for the attended image regions [3].

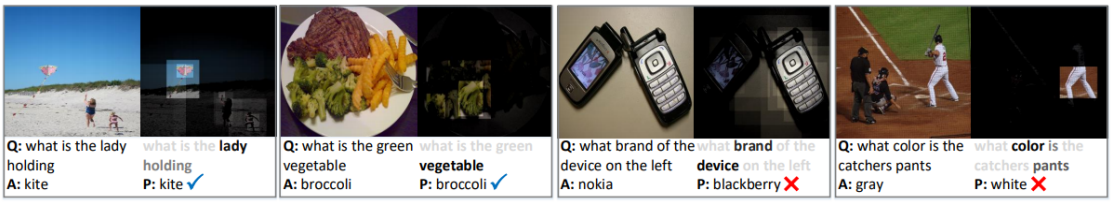


Figure 2.10: Examples of learned image and question attentions [151].

Besides, attention-based models are also used in low-resource scenarios, such as Bao et al. [12] train a domain-invariant representation and derive attention scores for words in a sentence from human rationales (Figure 2.8). The alignment model proposed by Bahdanau et al. [11] can search for parts of a source sentence that are relevant to predicting a target word. Attention mechanism can also be used for image captioning and question answering (Figure 2.9 and 2.10). All these attention modules are designed to deal with sequence modeling and generative modeling tasks in NLP.

2.4.2 Visual Attention Mechanism

Visual attention is the selective process that enables us to act effectively in our complex environment. In general, visual attention can be divided into two different categories: bottom-up attention, which is derived from the conspicuousness of regions in a visual scene, and top-down attention, which is driven by the " mental state" of the subject [35]. In real robotic applications, Region of Interest (ROI) of the same image may vary from different tasks. Attention-based model has been successfully applied in many research fields like visual SLAM [36], image caption [142], image classification and recognition [23, 50, 57], auxiliary disease diagnosis [92], and also question answering [53, 109, 141]. Even though the attention mechanism has achieved impressive results, massive manual annotation works are needed for building training datasets. Xu et al. [142] introduce two different attention models: a soft deterministic attention mechanism that can be trained by standard back-propagation and a hard stochastic attention mechanism that is trained with a reinforcement learning framework. Bello et al. [16] in-

troduce a two-dimensional relative self-attention mechanism and use it as an alternative to convolutional networks for image classification.

The attention mechanism is also used in unsupervised learning, like reinforcement learning. Rao et al. [112] propose an attention-aware model for video face recognition, which can discard misleading frames and find attention focuses. They formulate the attention finding process as an MDP and train the model using a reinforcement learning framework. Mott et al. [88] propose a top-down attention model, forcing the agent to focus on task-relevant information and select actions. They achieve the state-of-the-art performance for some Atari games and endow the agent actions with interpretability. However, the model is challenging to train because of its high complexity.

Chapter 3

Reinforcement Learning Fundamentals

3.1 Introduction to Reinforcement Learning

In this chapter, we present the preliminaries necessary to fully understand the Reinforcement Learning Algorithms in this thesis. RL is a machine learning method to train agents to solve specific tasks by exploration in the given environment. It is one of three basic machine learning paradigms, alongside supervised learning and unsupervised learning. The environment of RL is modeled in the form of MDP [49]. The core idea is that the agent can learn a feasible policy after enough trials and errors. All the algorithms proposed in this thesis are based on DRL. The main contents of this chapter are divided into two parts, the brief introduction to RL in Section 3.1.2 and different kinds of RL algorithms in Section 3.2.

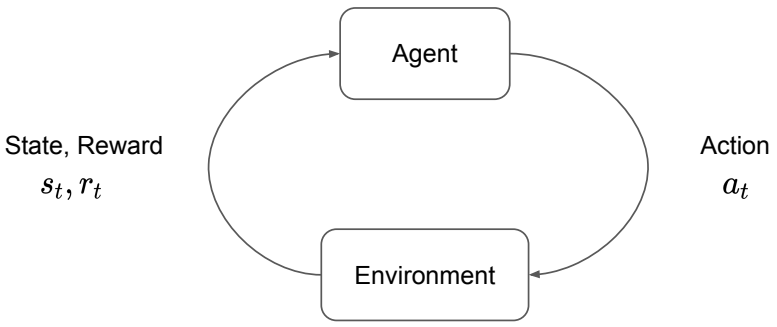


Figure 3.1: The interaction loop of agent and environment.

3.1.1 Real Applications of Reinforcement Learning

RL receives attention from various research fields and has a wide application in industries, ranging from autonomous fields like self-driving and robot control to the chemistry industry like drug design. The recent combination of deep learning and RL has made breakthroughs in the area of strategy games, like Go and DOTA, occupying the top of the ranking list and beating human players [17, 129]. Table 3.1 shows some real applications of RL in different industries.

Table 3.1: Reinforcement Learning Real Applications

Field	Application
Game	Train RL to play MOBA game [17] Train RL to play Go against Human [129]
Animation	Physical skills imitating from videos [103] Physics-Based character skills learning [100]
Robot Control	Bipedal robots locomotion control [75] Quadrupedal robot locomotion control [71] Solving Rubik's cube with dexterous hand [5] Robot Imitating Animals [102]
Resources Management	Home energy management system control [78] Google data centre cooling system control [38]
Trading	News recommendation [156] Sequential Recommendation [137] Online advertising recommendation [155]
NLP	Abstractive summarization [99] Dialogue generation [72]
Health Care	Equipment health status prediction [153] Personalized health recommendation [89]
Chemistry	Chemical reactions optimization [158] Drug design [106]

3.1.2 Key Concepts

Two essential parts of RL are the **agent** and the **environment**. The agent exists in and interacts with the environment. At each time step of the interaction, the agent obtains a whole or partial **observation** of the environment and takes an **action** according to the current policy. The environment also makes some changes during the interaction, and the result is usually stochastic. The agent receives a **reward** from the environment after each time step, which is a value quantifying the current state. The reward always comes from a reward function that needs to be carefully designed according to the task to learn. The goal of the agent is to maximize the cumulative reward.

The frequently used key concepts in RL include:

- Markov Decision Process
- Policy (Deterministic and Stochastic)
- Return
- Value Function

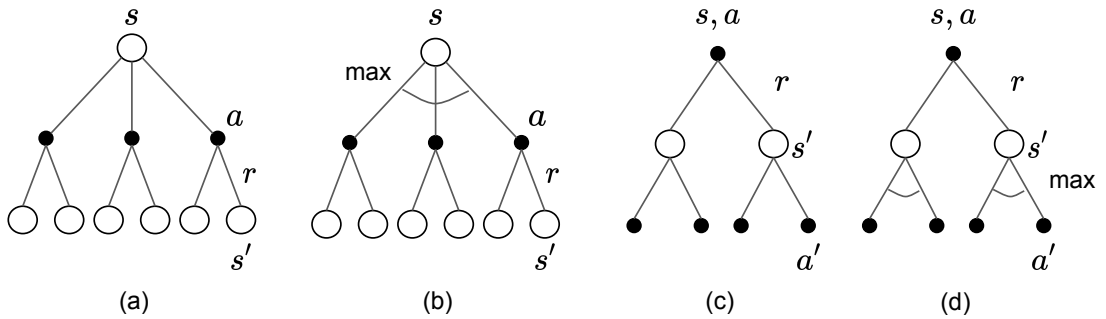


Figure 3.2: Diagrams for (a) $V^{\pi}(s)$, (b) $V^{*}(s)$, (c) $Q^{\pi}(s,a)$, and (d) $Q^{*}(s,a)$ [131].

- Optimal Policy and Optimal Value Function
- Bellman Equation

RL is modeled as a **Markov Decision Process**, which is a discrete-time stochastic control process. A MDP can be represented as a 5-tuple, $\langle S, A, R, P, \rho_0 \rangle$. Where

- S is the set of all valid states,
- A is the set of all valid actions,
- $R : S \times A \times S \to \mathrm{R}$ is the reward function,
- $P : S \times A \to \mathrm{P}(S)$ is the transition probability function,
- ρ_0 is the starting state distribution.

A **policy** in RL is the controller of the agent, deciding the action to take based on the current state. A policy can be **deterministic**, in which case denoted by $a_t = \mu(s_t)$, or **stochastic**, in which case denoted by $a_t \sim \pi(\cdot|s_t)$. As the policies in this thesis are all parameterized (by weights and biases) with trainable networks, the symbols are uniformly denoted as, $a_t = \mu_\theta(s_t)$ and $a_t \sim \pi_\theta(\cdot|s_t)$. A sequence of states and actions experienced by the agent is called a trajectory $\tau = (s_0, a_0, s_1, a_1, ...)$

Return is the cumulative reward over a trajectory τ, denoted as $R(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t$, in which the immediate reward r_t depends on the current state, the action taken at this time step, and the next state of the world: $r_t = R(s_t, a_t, s_{t+1})$. The discount factor $\gamma \in (0,1)$ is necessary for the convergence of the infinite-horizon return, and has an intuitive meaning: immediate reward is better than the reward coming later.

Value function is used to approximate the value of a **state** in every RL algorithm of this thesis. Value is the expected return starting from that state of a state-action pair and then interacting with the environment following a particular policy π. There are two kinds of **value function** frequently used by RL algorithms, namely the value function $V^{\pi}(s)$, which represents the expected return of an agent starting from state s and acts according to policy π:

$$V^{\pi}(s) = \mathbb{E}_{\tau \sim \pi}[R(\tau)|s_0 = s] \tag{3.1}$$

and **action-value function** $Q^{\pi}(s,a)$, which represents the expected return of an agent starting form state s, taking an arbitrary action a (not necessarily from the current policy π), and then acting according to policy π:

$$Q^{\pi}(s,a) = \mathbb{E}_{\tau\sim\pi}[R(\tau)|s_0 = s, a_0 = a] \tag{3.2}$$

Therefore, when action a samples from the current policy π, the two kinds of value functions are unified:

$$V^{\pi}(s) = \mathbb{E}_{a\sim\pi}[Q^{\pi}(s,a)] \tag{3.3}$$

We assume that the environment transitions are stochastic, in which cases, the probability of a T-step trajectory under the decision of policy π is:

$$P(\tau|\pi) = \rho_0(s_0)\prod_{t=0}^{T-1} P(s_{t+1}|s_t,a_t)\pi(a_t|s_t) \tag{3.4}$$

And the expected return is:

$$\mathbb{E}_{\tau\sim\pi}[R(\tau)] = \int_{\tau} P(\tau|\pi)R(\tau) \tag{3.5}$$

The **optimal policy** π^* can be obtained by solving the optimization problem:

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\tau\sim\pi}[R(\tau)] \tag{3.6}$$

The **Optimal value function** $V^*(s)$, which represents the expected return of an agent starting from state s and acting according to **optimal policy** π^*:

$$V^*(s) = \max_{\pi} \mathbb{E}_{\tau\sim\pi}[R(\tau)|s_0 = s] \tag{3.7}$$

Accordingly, the **optimal action-value function** $Q^*(s,a)$, which represents the expected return of an agent starting form state s, taking an arbitrary action a, and then acting according to the **optimal policy** π^*:

$$Q^*(s,a) = \max_{\pi} \mathbb{E}_{\tau\sim\pi}[R(\tau)|s_0 = s, a_0 = a] \tag{3.8}$$

The difference of several mentioned values is illustrated in Figure 3.2.

The Bellman Equations state an idea: The value of the starting state is the expected reward to get from being there, plus the value of the next state to arrive at. Both the **value functions** and the **action-value function** obey the Bellman Equations:

$$\begin{aligned} V^{\pi}(s) &= \mathbb{E}_{a\sim\pi}[r(s,a) + \gamma V^{\pi}(s')] \\ Q^{\pi}(s,a) &= \mathbb{E}\big[r(s,a) + \gamma \mathbb{E}_{a'\sim\pi}[Q^{\pi}(s',a')]\big] \end{aligned} \tag{3.9}$$

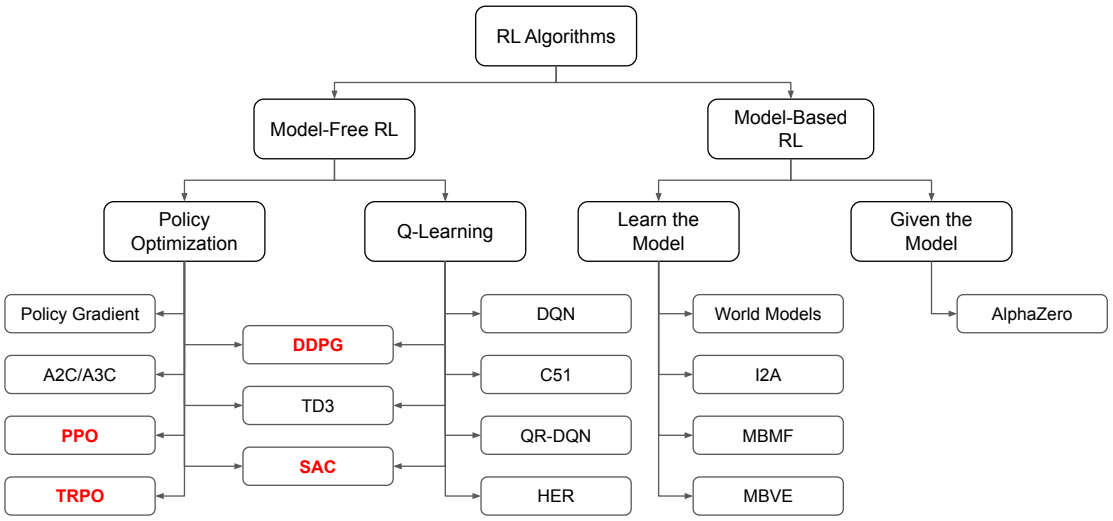


Figure 3.3: A Taxonomy of RL Algorithms [95], algorithms marked in red are used in the thesis.

3.2 Representative Algorithms

There are different ways to classify RL algorithms. From the perspective of complexity, the solutions of the RL problem can be divided into two broad categories, which are **tabular solution methods** and **approximate solution methods** [131]. The former methods are suitable for simple RL tasks, whose state and action spaces are small, and the value function can be represented in table form. In this case, the exact optimal value function and policy to the task can be found. However, when the problem is getting complex, assuming that most states encountered by the agent are never seen before, which is always the case when the state or action space (or both) is continuous. This requires that the value function can produce a good approximation over the whole state and action set.

From the perspective of the access to the model of the environment, which is used to predict the transitions and rewards, the RL algorithms can be divided into **model-free** and **model-based** methods. With the model available, the agent can plan the following action sequence in the future. The decision process is similar to MPC. The advantage of Model-Based methods is that the algorithms are of more sample efficiency. However, a ground truth model of the environment is usually hard to get, limiting applications' scope. Figure 3.3 shows a taxonomy of modern RL algorithms.

All the RL algorithms used in this thesis belong to DRL, which uses Deep Neural Networks to approximate the optimal policy and value function. With the continuous attention and the rapid development of the RL, it goes through a trail of research and finally evolves to powerful algorithms like Proximal Policy Optimization (PPO) (on-policy) and Soft Actor-Critic (SAC) (off-policy), which are state of the art algorithms on reliability and sample efficiency respectively. The RL algorithm selection in this thesis results from careful consideration of both reliability and sample efficiency according to the specific task.

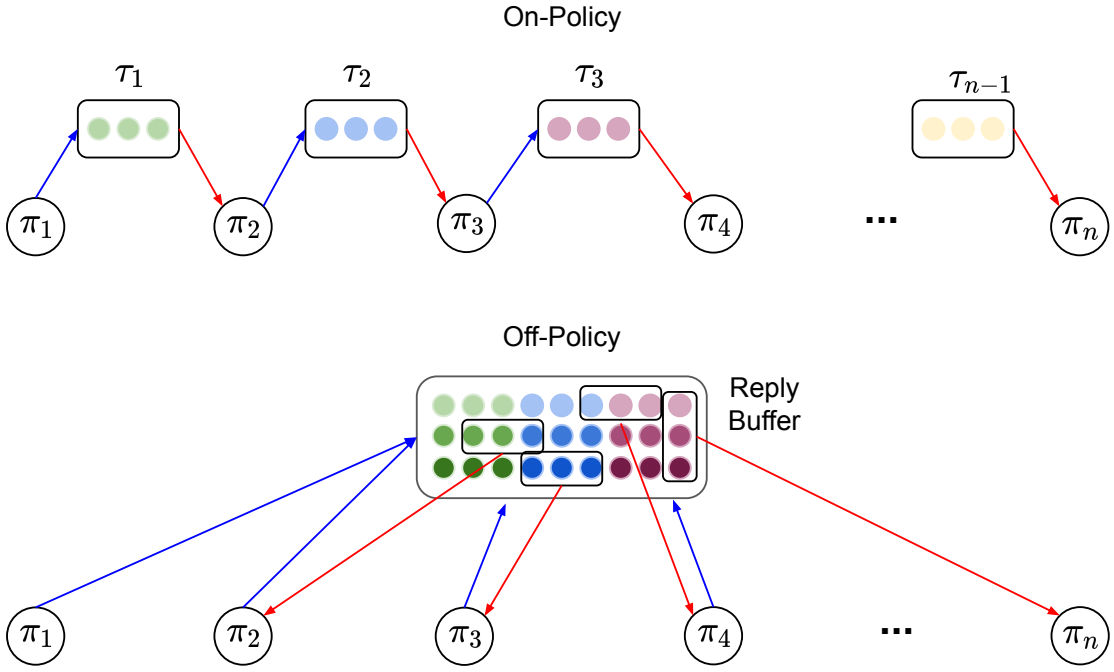


Figure 3.4: Updating of on-policy and off-policy algorithms. Blue and red arrows represent the trajectory collection and policy (or value function) updating process. On-Policy algorithms update policy π_t using the trajectories collected by the policy at the time (τ_t). off-policy algorithms use a large replay buffer to restore all the trajectories collected from both former and current versions of policies and sample some data from the buffer each time to update the policy π_t.

3.2.1 On-policy and Off-policy Algorithms

One pair of frequently mentioned concepts in RL are **on-policy** and **off-policy** (Figure 3.4). The key difference between the two kinds of algorithms is whether they use old data generated from former policies. On-policy algorithms do not use old data because the direct optimization of the policy performance calls for on-policy data (data generated by the current policy) to calculate the updates mathematically, which leads to the low sample efficiency. This kind of algorithm trades off sample efficiency for policy stability. On the contrary, off-policy algorithms can use data collected during training to update the value function efficiently. Bellman Equations is the mathematical basis to optimize the Q-function, in which case all interaction data from both the most recent version of the policy and former policies are effective. However, satisfying Bellman Equation can not guarantee a good policy performance. The disadvantage is that algorithms of this class are potentially unstable, e.g., more sensitive to hyper-parameters.

3.2.2 Policy Optimization and Q-Learning

As is shown in Figure 3.3, from the perspective of optimization approach, model-free RL includes two main kinds: **policy optimization** and **Q-Learning**. Four different model-free RL algorithms are used in this thesis. PPO and Trust Region Policy Optimization (TRPO) belong to policy optimization methods, Deep Deterministic Policy Gradi-

ent (DDPG) and SAC belong to the cross-field of Policy Optimization and Q-Learning. Policy optimization methods update the network weights by optimizing the accumulative expected return $\mathbb{E}_{\tau\sim\pi}[R(\tau)]$ (Equation 3.5) directly in the on-policy way. The primary strength of this approach is that the optimization target (expected return) positively correlates with the agent's performance, which ensures the reliance on the algorithm. By contrast, Q-learning algorithms optimize the policy indirectly by learning an approximator of the optimal action-value function $Q^*(s,a)$ (Equation 3.8) through the Bellman Equation in an off-policy way. The optimal policy is obtained by taking actions that maximizes the action-value function:

$$a(s) = \arg\max_a Q(s,a) \tag{3.10}$$

Q-learning methods reuse data to update networks more efficiently than policy optimization but come with the trade-off of the algorithm stability.

Chapter 4

Self-adapting Recurrent Model for Object Pushing

Planar pushing remains a challenging research topic, where building the dynamic model of the interaction is the core issue. Even an accurate analytical dynamic model is inherently unstable because physics parameters such as inertia and friction can only be approximated. Data-driven models usually rely on large amounts of training data, but data collection is time-consuming when working with real robots.

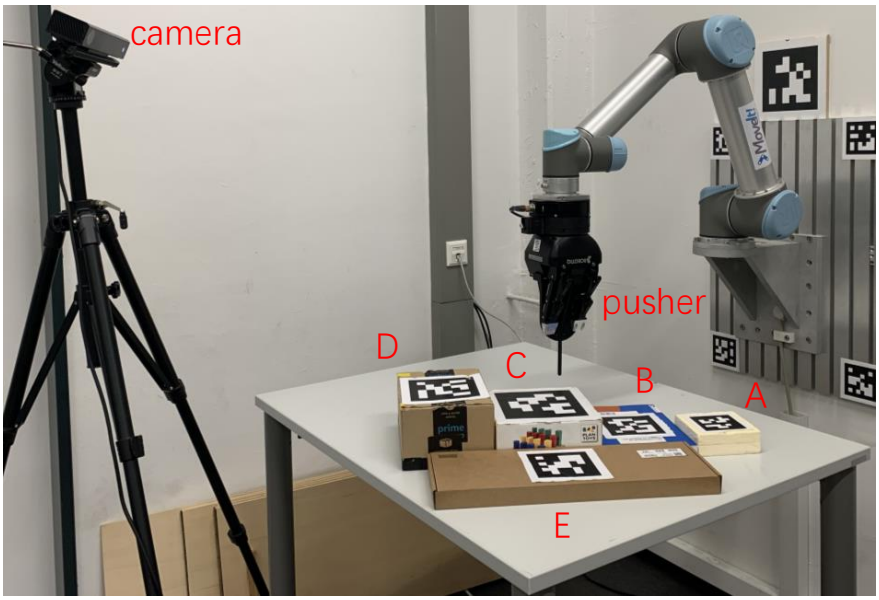


Figure 4.1: Our experiment platform consists of an UR5 robot with Robotiq 3-finger hand which grasps the 3D-printed vertical pusher rod. Its cylindrical part, designed to touch and push the moving object, is 6 mm in diameter. The five objects marked from A to E are test objects with different physical properties (sliding friction, rotating friction, mass, and size). Object position and motion are tracked using AprilTag markers and the camera.

In this chapter, we introduce a model-based control method for object planar pushing. We collect all training data in a physics simulator and build an Long Short-Term

Memory (LSTM) based model to fit the pushing dynamics. Domain Randomization is applied to capture the pushing trajectories of a generalized class of objects. The trained recursive model adapts to the tracked object's real dynamics within a few steps when executed on the real robot. In summary, our work investigates using an LSTM-based dynamic model, trained fully in simulation, to predict motion in a real environment, and apply MPC to control the robot. The whole framework is illustrated in Figure 4.2. The content of this chapter is organized as follows:

Section 4.1 elaborates on the question of object planar pushing and the corresponding dynamic model during the pushing process. Given an unknown object, a human tends to make some pushing primitives, learn the pushing dynamics from the interactive trajectory, and adapt motions according to the online generated trajectories. Inspired by how human beings learn from this kind of interactive process, we propose the self-adapting recurrent model (Section 4.2.1). The corresponding algorithm Model Predictive Path Integral (RMPPI) in the following Section 4.2.2 is a variation of the original Model Predictive Path Integral (MPPI) [138] approach.

As a comparison, we also train an RL policy with DDPG as a model-free baseline (Section 4.3), which is also used as the action generator in the data collection phase. During policy training, HER is used to improve exploration efficiency. Section 4.3.3 gives details on how we collect pushing trajectories in simulation using ADR. At last, real pushing experiments on our UR5 platform demonstrate the model's adaptability and the effectiveness of the proposed framework in Section 4.4.

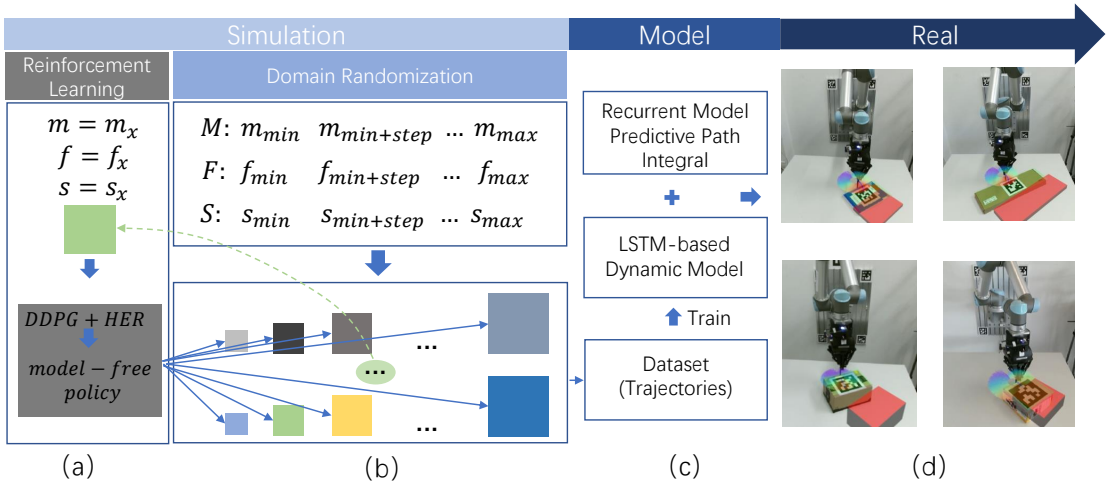


Figure 4.2: Overview: (a) A model-free generator policy (baseline also) is trained using DDPG and HER to push a prototypical object in a physics simulation. (b) The generator policy explores the randomized object with biased object interactions in the simulator. M, F, and S represent mass, friction, and size, respectively. (c) All sampled trajectories are used to train an LSTM-based dynamic model. (c-d) The model provides multi-step rollouts for a RMPPI controller to actuate the robotic system.

4.1 Problem Statement of Planar Pushing

Pushing can be taken as a pre-action before some purposive action, such as grasping. In general, the solutions to most kinds of robot manipulation can be divided into model-based and model-free. The model-based method, combined with MPC, is widely used in various robotic tasks. An analytical method is always the right choice when the parameters of the specific object are known for sure. However, getting an accurate dynamic model in a real environment is never an easy task.

Planar object pushing with a single contact is a typical instance of underactuated robot manipulation. The uncertainty of different physics parameters and the pressure distribution makes building a precise motion model for real interactions difficult. Figure 2.5 in chapter 2 illustrates the stochastic nature of a pushing process. Learning a data-driven model [13, 15] is an effective method due to this. Besides the model-based method, end-to-end policy learning by mapping from state space to action space directly is also becoming a trend [5]. A primary defect of such model-free methods remains the difficulty of collecting sufficient manipulation experience on real robots. Notorious instability of robot behavior during the exploration phase limits the feasibility of training. Policies can be trained in simulation environments to avoid physical training. In order to deploy such policies successfully in the real world, the training framework has to overcome the well-known gap between simulation and reality. In [5], Andrychowicz et al. use domain randomization to transfer their simulation result to a real environment successfully.

4.1.1 Analytical Model

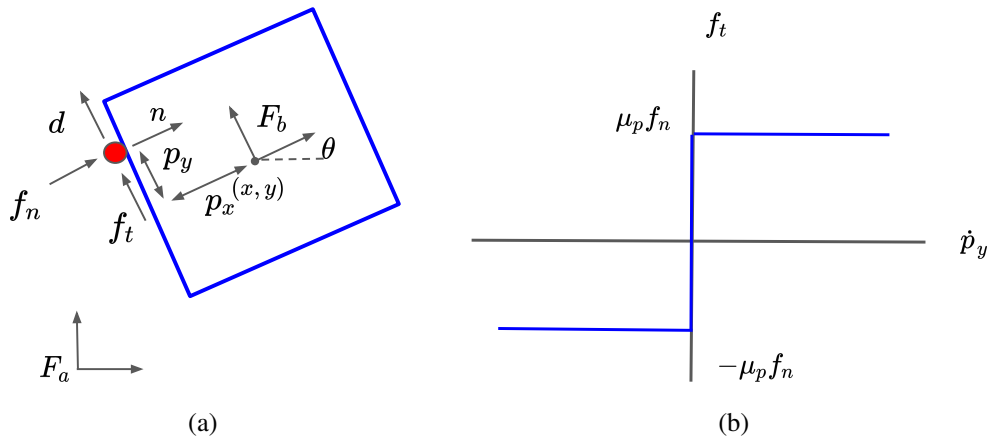


Figure 4.3: Illustration of analytical model. (a) Illustration of planar pushing model in world frame F_a and body frame F_b (b) Coulomb's frictional law. μ_p is the coefficient of friction [14].

A traditional method to predict an object's motion is to describe the dynamics as an analytical model [62]. Parameters in the analytical model have specific physical meanings. Therefore, the model can be easily transferred to similar problems given specific system

parameters. However, generalizing the analytical model to unknown objects is difficult, especially when the object's properties, such as friction and inertia, are hard to get.

According to the analytical model built by Bauza et al. [14], when the pusher interacts with the object, it impresses a normal force f_n, a tangential frictional force f_t, and torque τ about the center of mass (Figure 4.3(a)). The applied force causes the object to move in the perpendicular direction $H(w)$, as defined by Zhou et al. [157]. The object twist in the body frame is given by $t = \nabla H(w)$ where the applied wrench $w = [f_n \ f_t \ \tau]$ can be written as $w = J^T(nf_n + df_t)$ with $n = [1\ 0]^T$, $d = [0\ 1]^T$, and $J = \begin{bmatrix} 1 & 0 & -p_y \\ 0 & 1 & p_x \end{bmatrix}$. The system's motion equations are

$$\dot{x} = f_m(x, u_m) = \begin{bmatrix} Rt \\ \dot{p_y} \end{bmatrix}, \quad R = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.1}$$

where $x = [x \ y \ \theta \ p_y]^T$ is the state vector and $u_m = [f_n \ f_t \ \dot{p_y}]$ is the control input. The applied forces f_n, f_t and the relative contact velocity $\dot{p_y}$ must obey frictional contact laws 4.3(b).

4.1.2 Data-driven Model

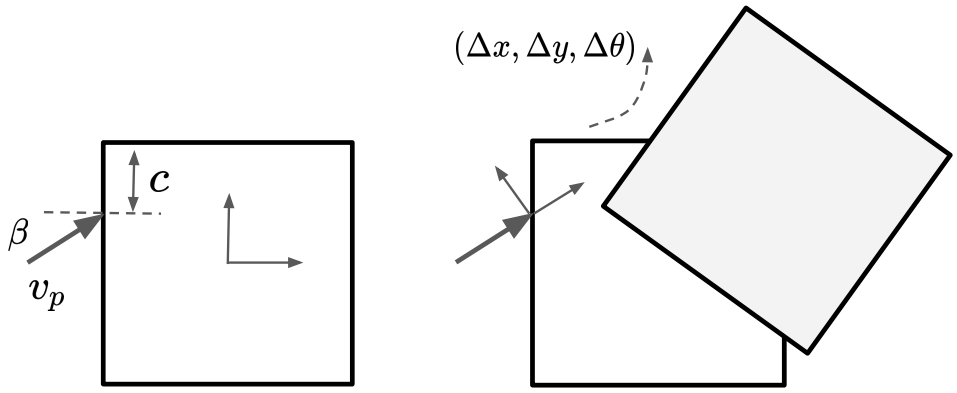


Figure 4.4: Illustration of data-driven model. (a) The pusher action (input of data-driven model). (b) Motion prediction in the body frame [15].

The data-driven model is an alternative to the traditional analytical model, which is more amenable than the analytical model for MPC because it presents continuous differential equations [14]. Bauza et al. presented a Variational Heteroscedastic Gaussian Processes based data-driven model on predicting the outcome of a planar push and its expected variability. The model can learn accurate models that outperform analytical models with no more than 100 samples. As is defined in [15], the object's expected motion and its

variance are denoted as:

$$\begin{aligned} \Delta x &\sim N(\mu_x(u), {\sigma_x}^2(u)) \\ \Delta y &\sim N(\mu_y(u), {\sigma_y}^2(u)) \\ \Delta\theta &\sim N(\mu_z(u), {\sigma_z}^2(u)) \end{aligned} \tag{4.2}$$

where $u = (v_p, c, \beta)$ is the pusher action, $\mu(u)$ and $\sigma^2(u)$ are the expected outcome and variance (Figure 4.4(a)).

4.2 Model Design and the Control Algorithm

4.2.1 Self-Adapting Recurrent Model

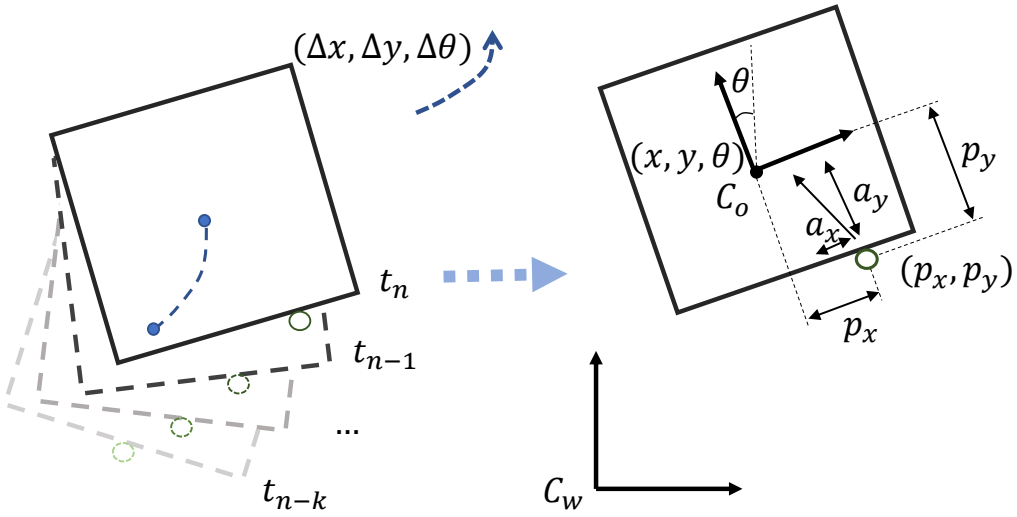


Figure 4.5: The figure illustrates a planar pushing system by showing the time sequence and variable representations. C_w and C_o denote word coordinate and object coordinate, respectively.

Humans can approximate dynamic models from several pushing steps during the interaction with an object and choose the proper pushing direction and velocity given a target pose. With the distinctive ability to recognize patterns in data sequences, the LSTM module is chosen to fit the object motion dynamics during the pushing process. We apply MPC as a control strategy to push the objects to target poses with a single point of contact. A new fusion motion prediction model without interaction force is proposed. Figure 4.5 illustrates the planar pushing system, where p_x, p_y, and a_x, a_y denote the pusher's position and the action in the object's frame. These variables are taken as inputs for the model in [14, 15]. x, y, θ denote the position of the object's geometric center and the object's rotation in the world frame. $\Delta x, \Delta y, \Delta\theta$ are the corresponding increments relative to a previous moment in the object's frame. What is different from the analytical model mentioned above (Section 4.1.1) is that the prediction of the interaction is inferred from its historical motion trajectory with LSTM rather than the current interaction force, which is hard to measure. In this model, the state increments $\Delta x, \Delta y, \Delta\theta$ are

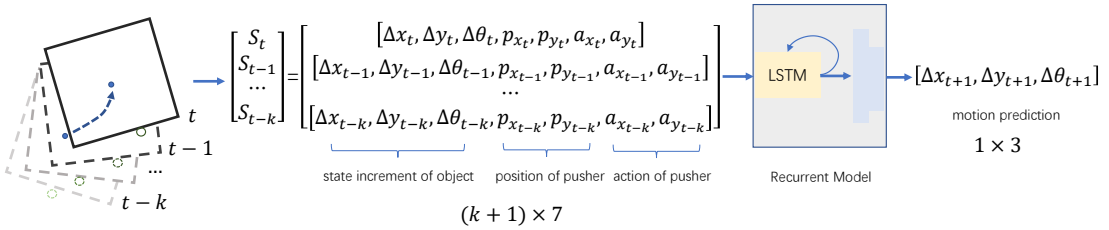


Figure 4.6: The model architecture used in our experiments. The trajectory steps are preprocessed and stacked into the input sequence for the LSTM module, followed by a two-layer fully connected network (the blue bar in the figure). The output from the module is also a motion sequence, the last element of which is the motion prediction for the next step. In the figure, we present only the last element instead of the whole output sequence.

adopted as part of the input, instead of absolute trajectory data x, y, θ to eliminate the influence of possible data distribution drift caused by different world coordinate origins:

$$\Delta x_t, \Delta y_t, \Delta\theta_t = \begin{cases} 0, & \text{if } t = 0 \\ [x_t - x_{t-1}, y_t - y_{t-1}, \theta_t - \theta_{t-1}], & \text{if } t > 0 \end{cases} \tag{4.3}$$

In summary, at moment t, a tensor representing the current state of the system can be denoted as S_t:

$$S_t = [\Delta x_t, \Delta y_t, \Delta\theta_t,\ p_{x_t}, p_{y_t},\ a_{x_t}, a_{y_t}] \tag{4.4}$$

We assume that the model M relies on the previous system state of $k+1$ time steps as the input to predict the motion in the object's frame for the next time step:

$$\Delta x_{t+1}, \Delta y_{t+1}, \Delta\theta_{t+1} = M([S_{t-k}, S_{t-k+1}, ..., S_t]) \tag{4.5}$$

After all necessary states are preprocessed and stacked as input, they are fed into the recurrent module (2-layer LSTM with 128 and 64 units). By preserving sequential information in the recurrent network's hidden state, the LSTM module achieves self-adapting to the real dynamics and outputs a tensor of 64 recurrent features. Then the motion predictor (2-layer fully connected network) takes the recurrent tensor as input and yields the motion prediction in the object's frame. Figure 4.6 illustrates the details of the recurrent model. The dropout rate is set to 0.5 for both layers of the LSTM. We train the model through stochastic gradient descent on the L_2 loss between the prediction from the network and the real object motion using the Adam [60] optimizer.

4.2.2 Recurrent Model Predictive Path Integral

MPPI [138, 139], as a typical MPC method, is investigated for optimizing nonlinear system model control. We focus on the planar pushing problem and set the task as pushing different objects with unknown physics parameters to different target poses. The algorithm has already been successfully applied in [80] for complicated robot manipulation problems in simulation. In order to endow the original MPPI with a memory mechanism, we add a history buffer H into the algorithm. We propose RMPPI, integrating traditional MPPI with recurrent state-dependent models. Algorithm 1 gives details of

Algorithm 1: Recurrent Model Predictive Path Integral (RMPPI)

Given:
N: Number of sampled action sequences;
T: Number of time steps to roll out;
$K+1$: Number of initial action steps;

$M : HistoryBuffer \times Action \rightarrow State$: Dynamic Model;
$C : State \times Environment \rightarrow Cost$: Cost function;

$(u_0, u_1, ..., u_K)$: Initial control sequence;

LSTM Warm Up Stage:
$H = empty\ HistoryBuffer$;
for $t \leftarrow 0$ **to** K **do**
    $S_t \leftarrow GetState(t)$;
    append S_t to H;
    $RobotAction(u_t)$;

Model Based Action Stage:
while not $TaskCompleted()$ **do**
    Sample rollout actions $U_{1...T}^{1...N}$;
    $H^{1...n} = H$;
    for $t \leftarrow 0$ **to** $T-1$ **do in parallel**
        $S_{t+1}^n = M(H^n, U_t^n)$;
        append S_{t+1}^n to H^n;
        $C_{t+1}^n = C(S_{t+1}^n, Env)$;
    $U_{1,...,n}^* = MPPI(C, U)$;
    $RobotAction(U_1^*)$;
    $S_t \leftarrow GetState(t)$;
    append S_t to H;

the whole framework. The results from our real experiments prove the effectiveness of our algorithm.

Before arriving at reasonable predictions, the LSTM module needs several steps to warm up. At the start of each pushing episode, we give $K+1$ steps of the initial control sequence $(u_0, u_1, ..., u_K)$ to the robot, which is the '***LSTM Warm Up Stage***' in Algorithm 1. The length of the initial control sequence depends on the input sequence we need to feed into the network.

After this stage, the memory buffer stores adequate states for the model to predict the object's motion. Then the algorithm goes into the '***Model Based Action Stage***.' For each pushing step, N pushing action sequences are sampled, and the LSTM prediction is calculated in parallel for all N sample trajectories and recurrently for T time steps to do the rollouts. The costs of each sampled action sequence are set as the distance between

the current state and the target.

$$C_t = \|State_t - Target_t\| \tag{4.6}$$

Then, using the path integral formula, an optimal action list $(u^*_{t+1}, u^*_{t+2}, ..., u^*_{t+T})$ is computed. Only the first action u^*_{t+1} is sent to the robot actuator and this process is repeated for every pushing step.

4.3 Model-free Pushing Baselines

In our task, a policy network is trained as a model-free baseline that maps the system's current state to robot action directly using the classic off-policy RL method DDPG [77]. The technique HER [4], which can be combined with an arbitrary off-policy RL to improve the exploration efficiency, is applied during training. The initial idea of training this policy network is to do a comparative experiment as the model-free baseline. The simulation and real experiments show that the model-free method is valid only for the prototypical object used during the training process but cannot be generalized to objects with different physical parameters. Even though the policy is not sufficient for achieving pushing tasks of a different object, we find that the policy can be used to collect pushing data for different objects during the study.

4.3.1 Deep Deterministic Policy Gradient

DDPG is an off-policy RL algorithm, which uses the Bellman equation to learn the Q-function, and then uses Q functions to learn a policy. The optimal action-value function $Q^*(s,a)$ can be represented through the Bellman equation:

$$Q^*(s,a) = \mathbb{E}_{s' \sim P}\left[r(s,a) + \gamma \max_{a'} Q^*(s',a') \right] \tag{4.7}$$

where s and s' are the current system state and next state; s' is sampled by the environment from distribution $P(s'|s,a)$.

A deep neural network Q_ϕ with parameter ϕ is used to approximate $Q^*(s,a)$. As an off-policy algorithm, DDPG takes advantage of the replay buffer $\mathbb{D}$, composed of a set of transitions (s,a,r,s',d), where d indicates whether s' is the terminal state of the episode. All transitions from $\mathbb{D}$ can be used to minimize the mean squared Bellman error function to fit the approximator Q_ϕ:

$$L(\phi, \mathbb{D}) = \mathbb{E}_{(s,a,r,s',d) \sim \mathbb{D}}\left(Q_\phi(s,a) - \left(r + \gamma(1-d) \max_{a'} Q_\phi(s',a') \right) \right)^2 \tag{4.8}$$

in which the term

$$r + \gamma(1-d) \max_{a'} Q_\phi(s',a') \tag{4.9}$$

is called the target. It is the fitting target of the Q function when minimizing the MSBE loss. Because the target depends on the same parameters with the network Q_ϕ, the training process of loss minimization becomes unstable. A time-delayed network (target network) is proposed whose parameters is close to ϕ, but lags the first for several updating time steps. The parameters are denoted ϕ_{targ}, and updated by Polyak averaging:

$$\phi_{targ} \leftarrow \rho\phi_{targ} + (1-\rho)\phi \tag{4.10}$$

where ρ is a hyperparameter: $\rho \in [0,1]$. Then the max over actions in the target

$$\max_{a'} Q_\phi(s',a') \tag{4.11}$$

can be approximated by the target network $Q_{\phi_{targ}}$.

By composing a target policy network $\mu_{\theta_{targ}}$ to compute an action which approximately maximizes $Q_{\phi_{targ}}$. The parameters θ are updated by polyak averaging using the same as target Q-function. To summary, Q-function is trained by minimizing the MSBE loss with gradient descent:

$$L(\phi,\mathbb{D}) = \mathbb{E}_{(s,a,r,s',d)\sim\mathbb{D}}\left(Q_\phi(s,a) - \left(r+\gamma(1-d)Q_{\phi_{targ}}(s',\mu_{\theta_{targ}}(s'))\right)\right)^2 \tag{4.12}$$

And the gradient ascent formula for the policy learning can be derived by solving:

$$\max_\theta \mathbb{E}_{s\in\mathbb{D}}\left[Q_\phi(s,\mu_\theta(s))\right] \tag{4.13}$$

which means searching for the action that maximizes Q-function.

4.3.2 Policy Details

In our task, the pusher mounted on the robot arm is the agent interacting with the stochastic environment E, which is the pushing task scenario. The agent's behavior is determined by policy π, which maps states to actions $\pi : S \rightarrow A$. In the task, we take sparse binary rewards and follow a Goal-Based Reinforcement Learning framework in which the agent is told what to do using additional input [105]. We model it as an MDP with state space:

$$S = [X_o, Y_o, \theta_o,\ X_r, Y_r,\ X_g, Y_g, \theta_g] \tag{4.14}$$

in which the variables denote the current pose of the object, the robot end effector and the goal pose respectively. The action space is:

$$A = [X_a, Y_a] \tag{4.15}$$

in which X_a and Y_a denote the action of the robot end-effector. Other elements in the MDP are: initial state distribution $p(s_0)$; transition dynamics $p(s_{t+1}|s_t,a_t)$, and sparse reward function:

$$r(s_t,a_t) = \begin{cases} -1, & \text{if goal is not achieved} \\ 1, & \text{if goal achieved} \end{cases} \tag{4.16}$$

The return of a state-action pair is defined as the sum of discounted future reward $R_t = \sum_{i=t}^{T} \gamma^{(i-t)} r(s_i,a_i)$ in which γ is a discounting factor $\gamma \in [0,1]$.

Table 4.1: Dynamic Parameters and Their Ranges in Simulation

Parameters	**Range**
Size	[8, 25] cm in length = width
Mass	[0.01, 0.8] kg
Sliding Friction Coefficient	[0.1, 1]
Rotation Friction Coefficient	[0.001, 0.01]
Damping Coefficient	[0.01, 0.015]

4.3.3 Trajectory Collection with ADR in Simulation

Performing robotic learning in simulation has recently become especially promising for reaching human-level performance in various tasks such as using tools [110], object localization [133], and games like Atari [60]. The main advantage of learning in simulation is a faster, lower-cost data collection process. We use domain randomization in simulation to change the physics parameters of the target objects, including mass, size, sliding friction, and rotational friction. In the real-world pushing process, the trained LSTM continuously observes the actual object motion and self-adapts to the object physics within a few (less than 5) pushing steps. We randomize 300 objects with different physical parameters. With enough variability, an object from the real world may appear to be a variation from the randomized domains [101]. As represented in Figure 4.2, objects are generated according to physics parameters from different domains in Gym [20,134], after which the data collection process is performed on these objects. All objects in the simulation have the same size in height (2.5 cm). Table 4.1 details the range of objects' parameters.

Instead of applying random actions to exploring object properties as in [13,149], we use the trained model-free policy to generate actions to push the randomized objects to goal poses. The DDPG policy is trained with a prototypical object with fixed parameters and cannot generalize to different objects. The specific parameter of the prototypical object is shown in table 4.2 as object P. As a result, this policy cannot push the random object to the target pose ideally, and it will generate noisy actions, but the general movement tendency is towards the target pose. This kind of robot motion is especially suitable for exploring the objects' properties. During the whole exploration (data collection) process, the initial and target poses of the objects are set randomly for going through as many different states of the object as possible. An episode is finished when the object reaches the target pose, or the robot executes the 60th pushing step. This data collection approach proves to be sufficient to explore objects with different parameters. The collection result shows that a model-free policy trained with a prototypical object can efficiently bias the exploration of novel objects for subsequent model learning.

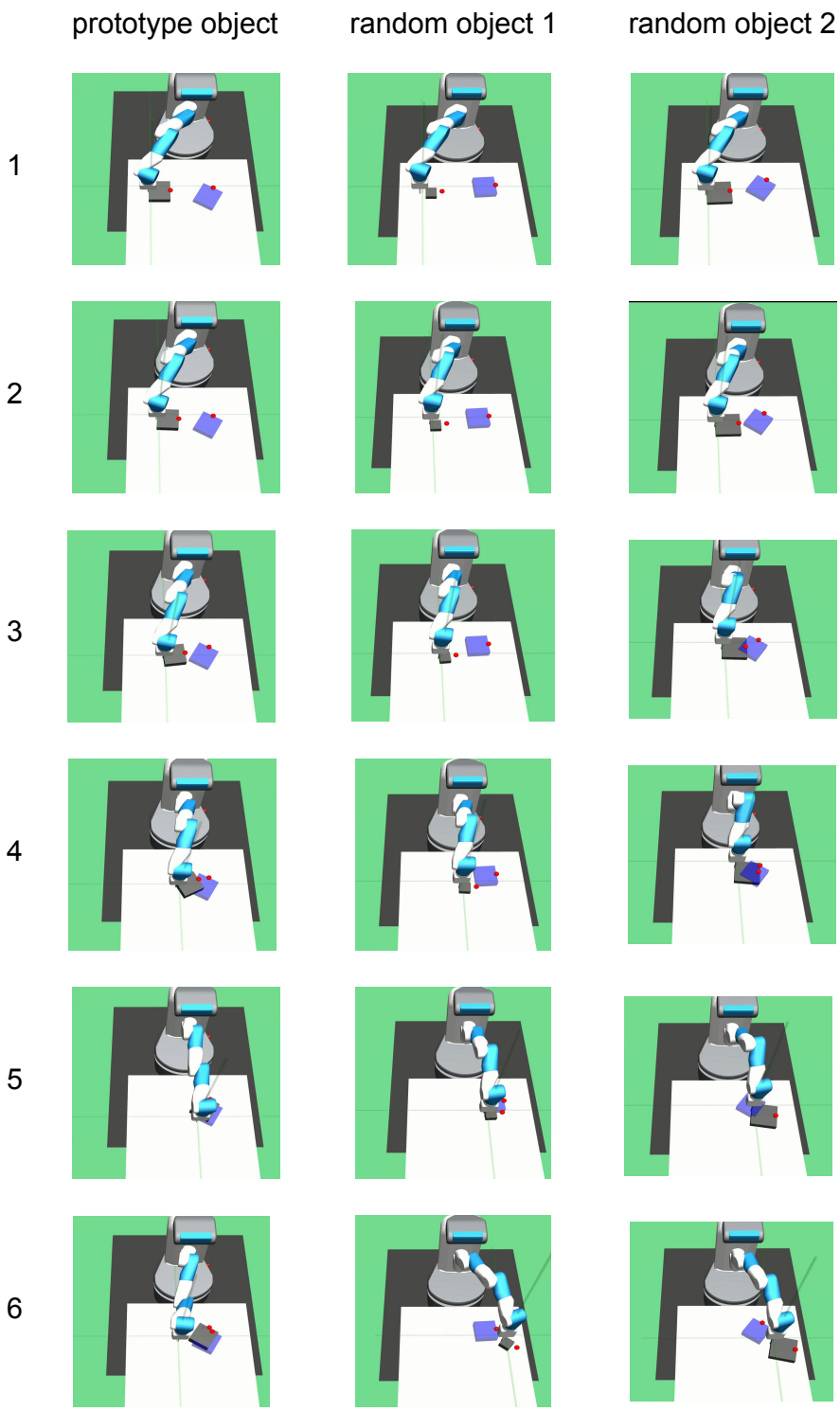


Figure 4.7: The illustration of trajectory collection with domain randomization. Three columns show the pushing process of three different objects (6 steps for each process). The first column is the prototype object used for training, and the others are two random objects with different physics parameters.

4.4 Experiments

Through real experiments, we demonstrate that a state-dependent model trained from simulation can effectively predict the motion of real objects without further fine-tuning.

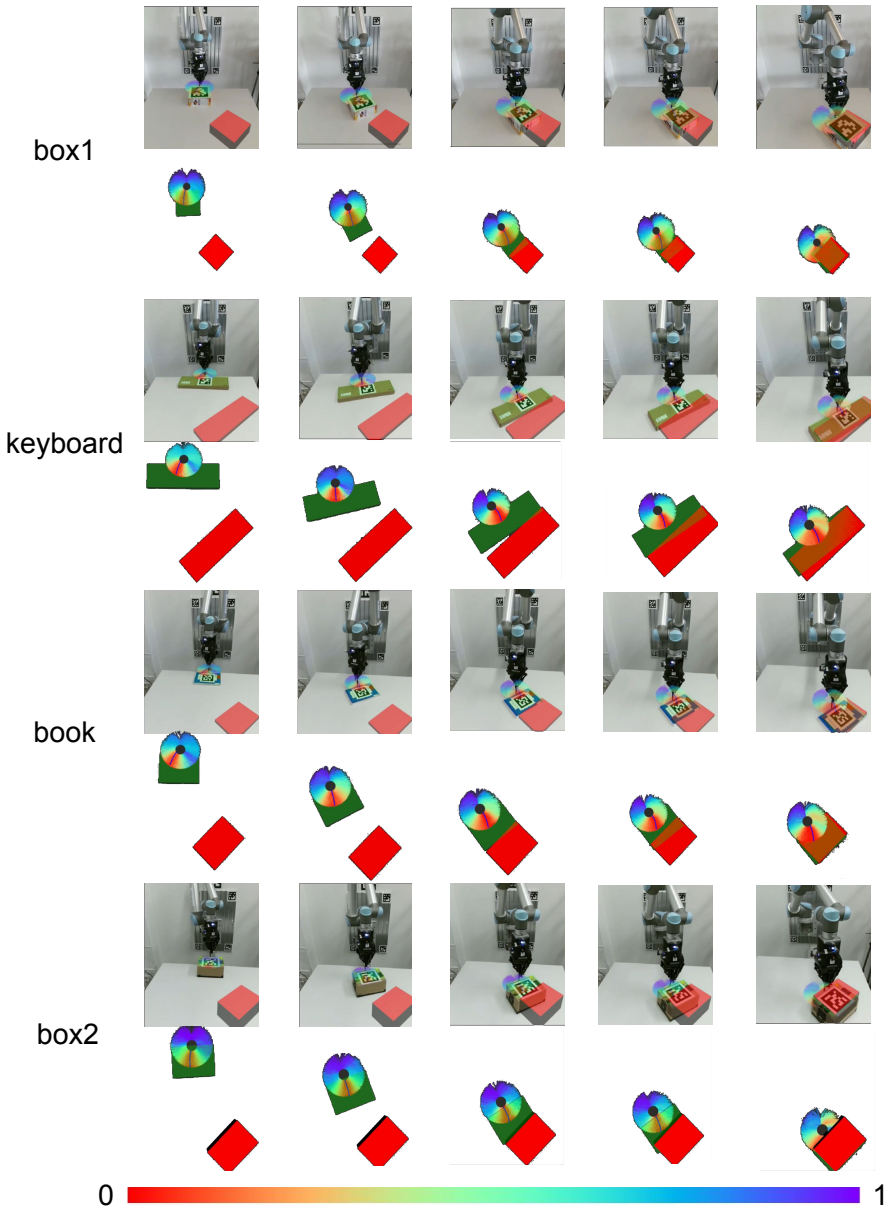


Figure 4.8: Example pushing trajectory of 4 different objects. The pictures in the first-row show camera images from the real robot experiment with visualization overlays for each object. Second row: Visual markers in rviz in which the green marker represents the current pose of the object, and the red marker represents the goal pose. For each pushing step, 1024 sampled rollout action sequences are sampled. The different colors of the trajectories are normalized costs (ranged in $[0, 1]$). One optimal action sequence is computed in every picture. The color bar represents the cost value.

4.4.1 Robot Setup

We test our methods in the real world on a UR5 robot shown in Figure 4.1. The robot is mounted to a wall above a table and holds a pushing rod (pusher) to interact with the objects. Object positions are measured using a camera and AprilTag markers [93]. Pushing commands are interpreted as Cartesian motion goals. These motion goals are translated by an online controller [115] into joint-space robot commands under kinematic and dynamic constraints. End-effector positions are restricted to a rectangular workspace above the table, and the axis along the pushing rod is fixed in an upright position. Motions are constrained by velocity and acceleration limits.

4.4.2 Benchmark Description

We propose a model-based method for pushing different objects, and the model is trained with all the data from the simulation. A model-free method, which is a deep policy network trained by DDPG, is taken as the baseline (Section 4.3.2). The trained policy is also deployed on the real robot platform (Figure 4.9). Both approaches are evaluated in real experiments. Through experiments, we find that the hyperparameters in RMPPI have a significant impact on the pushing performance. The data analysis is detailed in the next section. Figure 4.8 shows selected steps from one pushing experiment using four different objects in rviz. The pictures from each column represent robot and object states of the same time step. Green and red rectangles represent the current pose and target pose, respectively. In each figure, the 1024 rollout action sequences of the pusher are shown. The color of each line is the normalized estimated cost of the action sequence, computed by the cost function we defined acting on the prediction results from the dynamic model. Red means a low-cost pushing direction, while blue stands for a high cost.

Table 4.2: Parameters of Experimental Objects

	Mass (kg)	Length (m)	Width (m)	Sliding Friction (N)
A	0.016	0.116	0.116	< 0.1
B	0.615	0.168	0.237	≈ 1.4
C	0.565	0.198	0.198	≈ 1.1
D	0.587	0.166	0.228	≈ 1.8
E	0.506	0.153	0.462	≈ 0.9
P	0.015	0.120	0.120	≈ 0.05

To demonstrate the adaptability of our model by predicting objects with unknown motion properties, we plot the trajectories of both object and pusher in Figure 4.10, in which (a) and (b) denote trajectories of objects E and C, respectively, during the pushing process towards the three target poses "left," "middle," and "right." Target poses are marked by "red," "blue," and "green" squares, respectively. Light gray squares represent the final poses in the real experiment. Black squares also denote starting positions. (c)

and (d) denote the corresponding trajectories of the pusher generated by RMPPI in the same experiment (a) and (b).

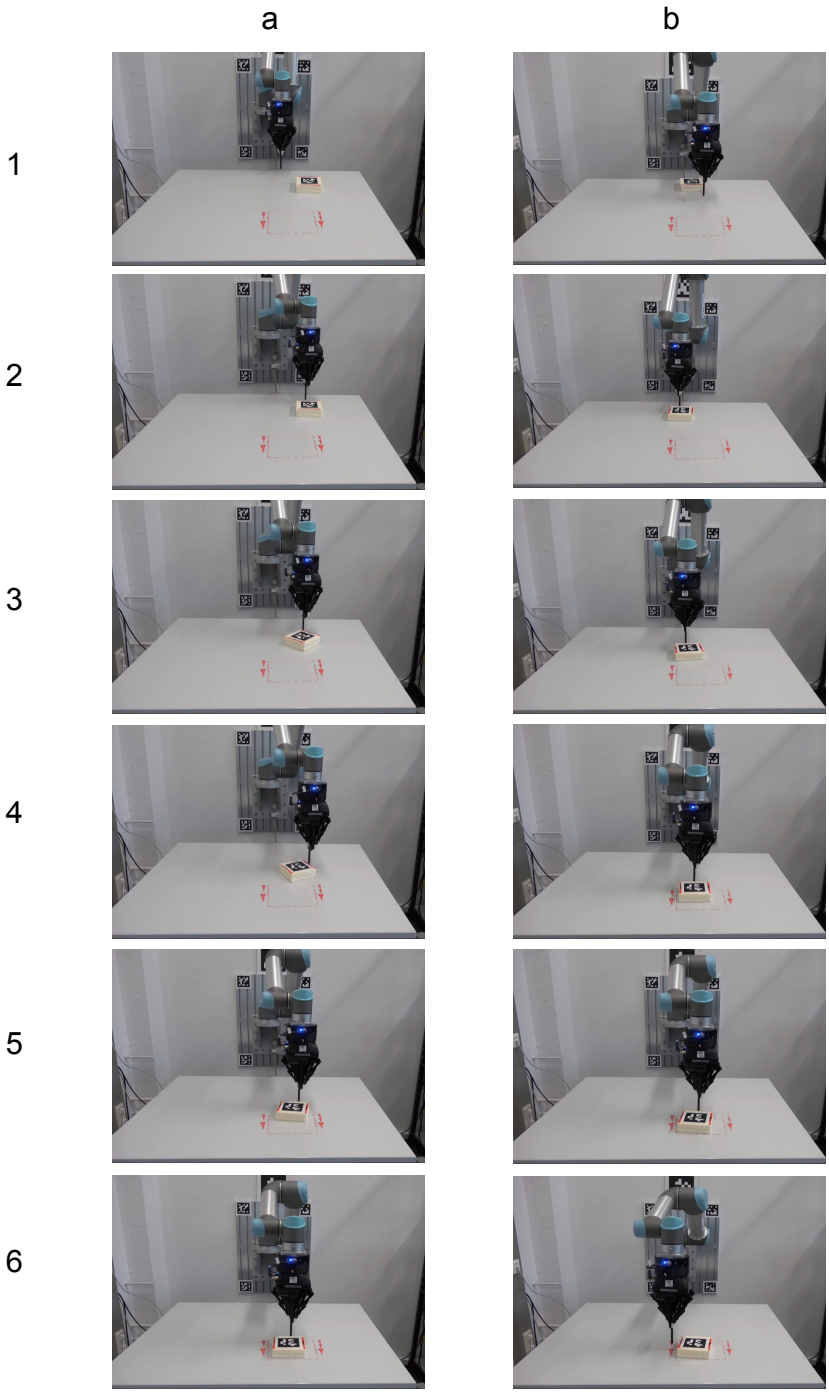


Figure 4.9: model free pushing policy trained with DDPG is applied on a real robot platform. The figure shows two pushing processes of the prototype object used during simulation training (6 steps for each process). The red rectangle on the table is the target pose.

To compare the motion properties of different objects, during the experiment, we set

the same target poses for different objects. From the trajectories of the pusher, we notice that the initial several pushing steps are taken straightforward, which corresponds to the warm-up stage in algorithm 1. Then the robot pushes objects using the computation result from RMPPI. From the pusher trajectories of the "middle" target pose, we notice that tiny direction adjustments are adopted to keep the object facing forward. Comparing the trajectory distribution towards the same target pose between (c) and (d), we can notice that the network can adapt itself to the real dynamics of different objects. The final poses reached are all close to the goal poses, proving that RMPPI is highly robust in the real environment. The length of each pushing is set as 0.5 cm. Because only the first action is executed, the motion is not smooth, with a momentary stop after each pushing action. Executing the sequence's first two or three actions and running the model in parallel can make the motion smooth; this will be done in future work.

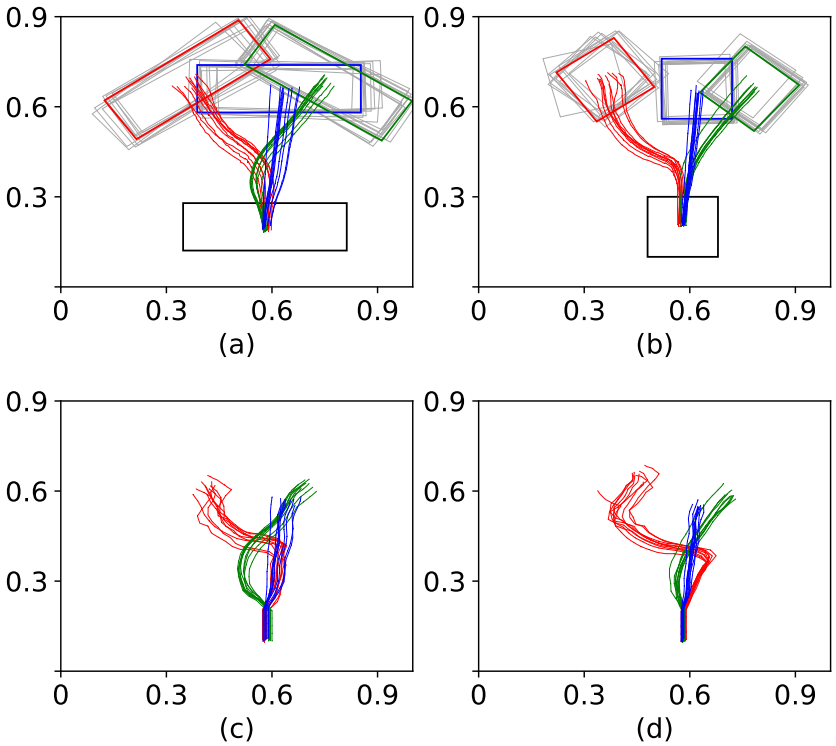


Figure 4.10: Example trajectories from the real robot experiment. (a,c) Trajectories of the keyboard and the robot. (b, d) Trajectories of the square box and the robot.

4.4.3 Analysis

We choose objects with different physical parameters as test objects and measure their parameters after the experiments. The results are shown in Table 4.2. Five different objects (A-E) are used to evaluate our model's adaptability, among which object A has almost the same parameters (size, mass, friction) as the prototypical object (P) we used for training the model-free baseline. Table 4.3 shows the success rate of pushing objects to target poses under different thresholds. in which:

Level 1: $(E_x < 0.025(m))\&(E_y < 0.010(m)\&(E_\theta < 0.052(rad))$

Level 2: $(E_x < 0.035(m))\&(E_y < 0.015(m)\&(E_\theta < 0.087(rad))$
Level 3: $(E_x < 0.050(m))\&(E_y < 0.025(m)\&(E_\theta < 0.17(rad))$

Our model fits all five object motion properties and achieves a reasonable pushing success rate, while the model-free baseline does not work well on any object that differs from the prototypical one trained in simulation.

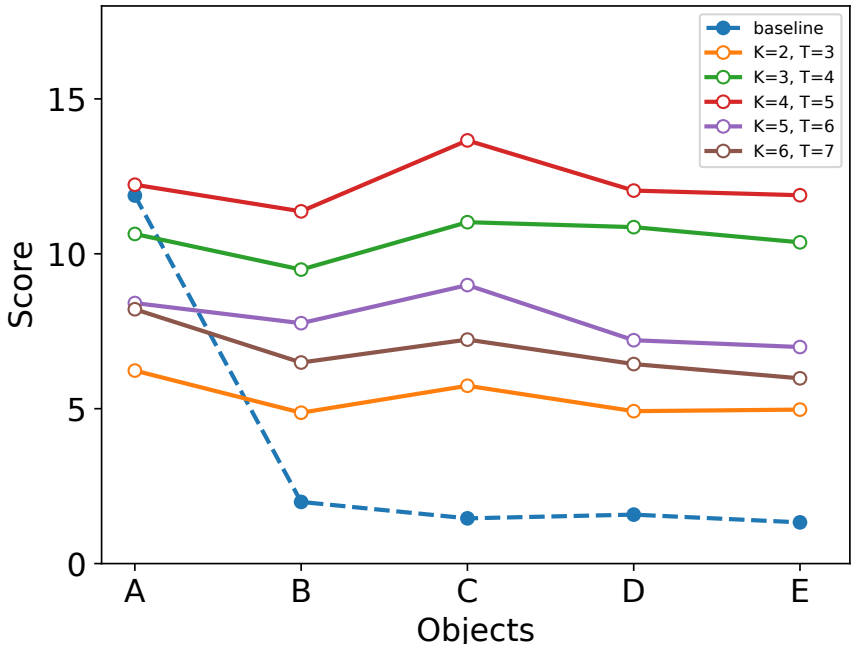


Figure 4.11: The scores computed according to the evaluation formula, $K+1$ is the length of the sequence fed into LSTM for each prediction, which is also the same as that in Equation (4.5). T is the number of steps to roll out in algorithm 1, the baseline is the score of the model-free approach.

In order to get a comprehensive evaluation for each pushing result, we set a score calculation formula with the absolute error in X direction, Y direction, and θ:

$$S = \frac{1}{E_x + 3 \cdot E_y + 0.5 \cdot E_\theta + \sigma} \tag{4.17}$$

in which E_x, E_y and E_θ are the final errors to the target pose, constant $\sigma = 1e-7$. The factors used for the different errors are chosen to keep them on the same magnitude. During the real experiments, we find that the two hyperparameters in RMPPI that influence the pushing performance the most are K in Equation (4.5) and T in Algorithm 1; the number of previous states used as inputs to the LSTM, and the number of timesteps used in the rollouts before calculating the costs. During the LSTM warm-up stage, we set straightforward pushes as the first $K+1$ actions of the pusher. To find the best combination (K^*, T^*), we try different groups of K and T in real experiments and plot part of the combined performance in Figure 4.11. As K increases from 2 to 4, the score keeps increasing and reaches the top at 4. This means the LSTM module needs a sequence of at

Table 4.3: Success Rate Comparison with Model-Free Baseline

Threshold	Method	A	B	C	D	E
Level 1	RMPPI	85.3%	82.8%	87.2%	83.3%	81.4%
	Model-Free	85.5%	22.6%	19.1%	11.7%	12.3%
Level 2	RMPPI	90.5%	88.9%	91.5%	88.3%	87.5%
	Model-Free	87.8%	24.9%	19.6%	12.5%	12.5%
Level 3	RMPPI	**93.5%**	**90.9%**	**93.9%**	**91.6%**	**89.5%**
	Model-Free	92.8%	25.6%	19.3%	14.8%	13.7%

least 4 time steps to get the best prediction; a longer input sequence will not improve the prediction accuracy further. Because the prediction error can be accumulated through action sequence $[u_{step}, ..., u_{step+T-1}]$ from the current step to T later steps, there is also the best prediction step number T^*. Through testing, we find the best combination is $K^* = 4, T^* = 5$.

4.5 Summary

This chapter builds a recurrent model that can adapt to the real interaction dynamics in object pushing tasks. RMPPI is proposed as the controller. Through domain randomization, we bridge the gap between simulation and the real environment. The model is trained in simulation only but can be used in the real environment without any fine-tuning. Although the model requires an initial warm-up stage for adjusting itself, this is exactly what humans do as well when we start working with a novel object. Results show that the new algorithm is of high robustness.

Besides, a model-free RL pushing policy is also trained as the baseline in simulation. The agent can switch the pushing side according to the relative pose of the object and target. However, the weakness is the generalization ability. Through the analysis of the results, our key findings are:

- Given proper state variables, a well-trained LSTM-based model can learn to predict object motions for objects of different sizes and shapes, self-adapting to the actual object dynamics online after only a few pushing steps like a human.
- Recurrent model can be integrated effectively in an MPC framework.
- Domain randomization effectively bridges the gap between simulation and real environment in robotic learning tasks.

Chapter 5

Vision-Proprioception Model for Object Pushing

In the previous chapter, we focused on a model-based method. We built a fusion model which adapts itself to the real interaction dynamics after several pushing interactions using the proposed RMPPI algorithm as the controller. One limitation of this method is that the robot cannot effectively switch pushing sides according to the object's current pose during the pushing process. Another limitation is that we need AprilTag [93] to locate the object in real-time. This chapter trains an RL policy that takes the raw image and the pusher position as input. After enough training episodes in simulation, the trained agent learns to make good decisions on switching the pushing side both in simulation (Figure 5.7) and in the real world (Figure 5.9).

We propose a vision-proprioception model for planar object pushing, efficiently integrating all necessary information from the environment. For the vision part, a VAE is used to extract valuable representations from the task-relevant part of the image into latent space, the same way as the goal information is encoded. With the real-time robot state obtained easily from the hardware system, we fuse the latent representations from the VAE and the robot end-effector position into the state of an MDP. We use SAC to train the robot to push different objects from random start poses to target positions in simulation. During the training process, HER is applied to improve the sample efficiency. Experiments demonstrate that our algorithm achieves a pushing performance superior to a state-based baseline model that cannot be generalized to a different object. Moreover, it outperforms state-of-the-art policies operating on raw image observations only. At last, we verify that our trained model has a good generalization ability to unseen objects in the real world.

The remainder of this chapter is organized as follows: the necessity of visual input (image) is elaborated in Section 5.1.1. Section 5.1.2 introduces the usage of VAE in our work and shows the GUI, which is designed to visualize the latent space of the encoder. The vision-proprioception model and the training method of the policy with the fusion model are detailed in Section 5.1.3 and Section 5.2.2. Both the simulation results and real experiments analysis are given in Section 5.3.

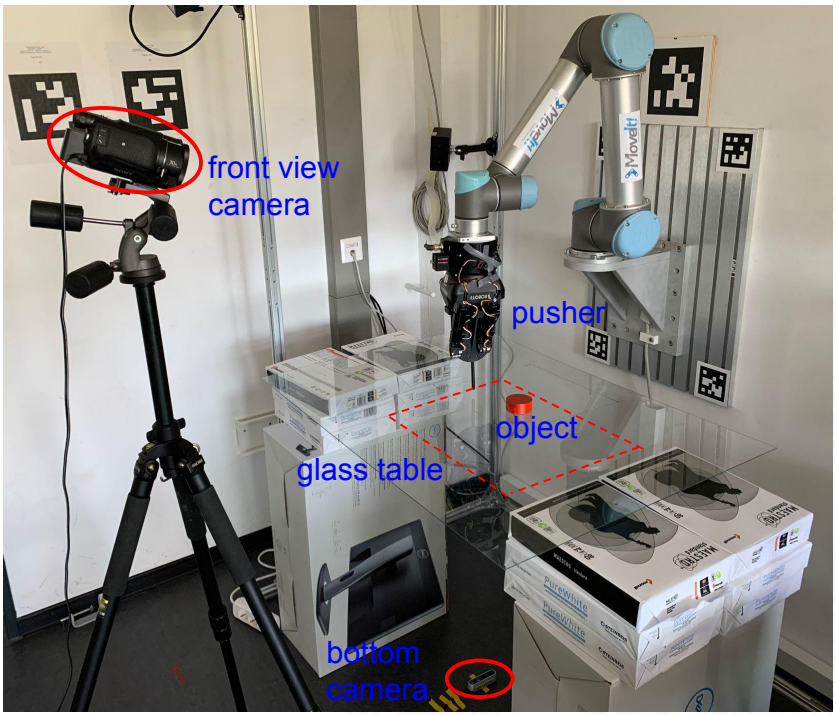


Figure 5.1: Our experiment platform consists of a UR5 robot with a Robotiq 3-finger gripper that grasps the 3D-printed vertical pusher rod. The pushing target placed on the transparent table is painted red to obtain masks easily through color filtering. The bottom camera (to get the input image) is set right below the table, and another front camera is added to record experiment videos.

5.1 Vision-Proprioception Model

5.1.1 Necessity Analysis of Image Input

To enhance the robot's generalization ability to manipulate different objects by extracting task-relevant information such as the object's shape and pose features from image observations. However, in most situations, the raw image from the camera always includes complicated components such as noisy background, which is hard for the robot to understand. This work extracts useful information by segmenting an object mask from the image and constructing a latent representation through a VAE. As the decoder from the trained VAE can roughly reconstruct most of the original masks from the latent features, these features should include the object's pose and shape information.

The robot state could be inferred from the camera image through a deep network, but this is usually unnecessarily difficult. Unlike the object state, the robot state, including the end-effector pose, can be obtained directly from the hardware system. We, therefore, propose a vision-proprioception model to fuse the latent features and robot states as the RL inputs. We train the robot to push objects to target positions using a carefully designed reward function (Section 5.2.3). The models are evaluated by the distance between final real positions and target positions.

5.1.2 Variational Autoencoders

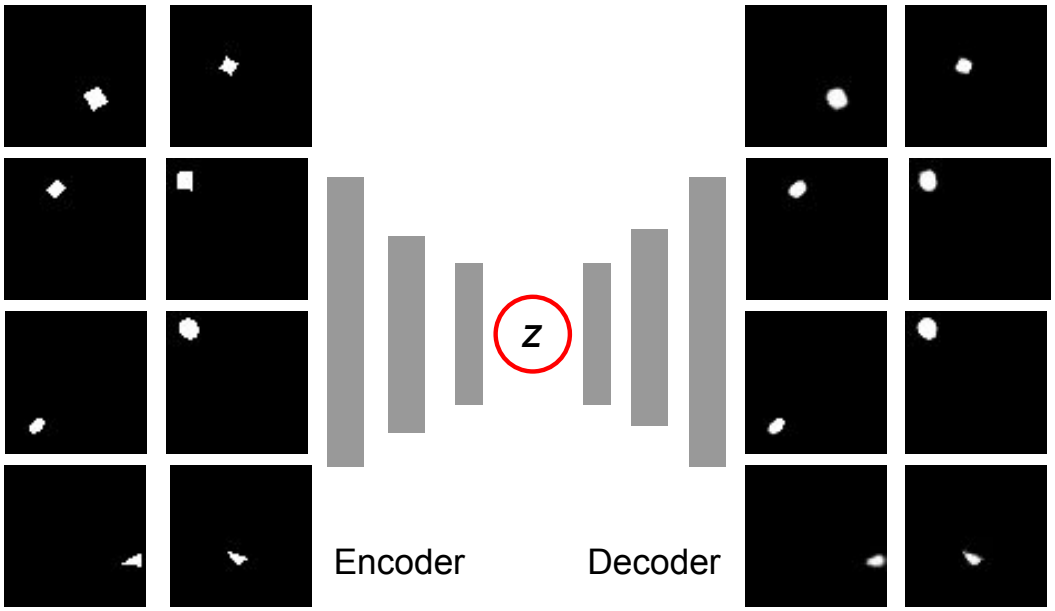


Figure 5.2: The training dataset (left) and reconstruction (right) of the VAE. All of the images are sized 64x64. The dataset includes 20000 images. As we can see, the position and general shape are well reconstructed, even though some details such as the rotation angle and sharp corners are not exactly the same as the originals.

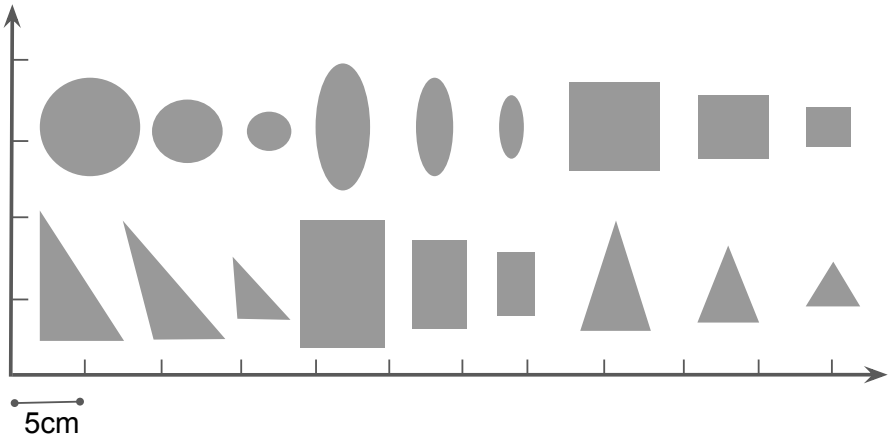


Figure 5.3: We use 18 different objects in our pushing task, all of them of 2 cm height. The physical parameters which can affect the dynamics are randomized from a range in Table 5.2 with ADR during training.

To deal with high-dimensional image inputs, we train a latent representation of the state by VAE. A VAE is a probabilistic generative model composed by an encoder which converts state x into a prior distribution $q_\theta(z|x)$ and a decoder that converts the latent variable z back to a state distribution $p_\omega(x|z)$ as is shown in Figure 5.2. The model

is trained by minimizing the reconstruction loss of original states x (the first term in the Equation 5.1) and forcing the latent representation z to be similar (in the form of KL divergence [65]) to a prior distribution (second term) at the same time. We take Gaussian distribution as prior here.

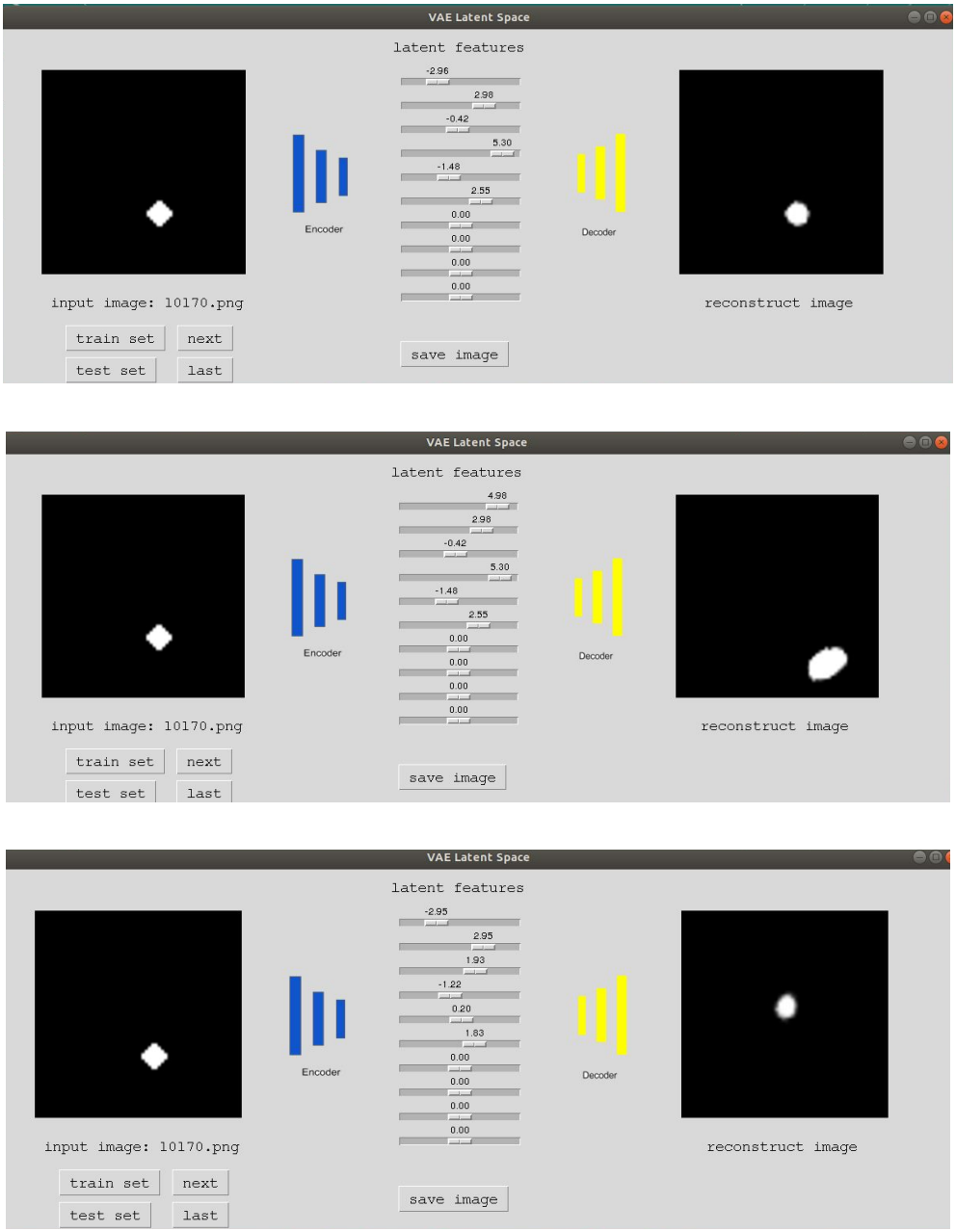


Figure 5.4: We write a GUI for the VAE, which is used to visualize the latent space. The image on the left and right sides are the input and the reconstruction of the latent values. The reconstruction image on the right changes accordingly by resetting the value bar's latent values in the middle.

$$L(\theta,\omega) = -\mathbb{E}_{q_\theta(z|x)}[\log p_\omega(x|z)] + \mathbb{KL}(q_\theta(z|x)||p(z)) \tag{5.1}$$

In this task, we only care about the pose and shape of the object and target in the image; color and texture information can be ignored. We filter the image from the bottom camera by color (red) and obtain the masks (Figure 5.5). The VAE is trained on these one-channel 64x64 masks. We collect a mask dataset by randomizing the pose of different objects on the table. Figure 5.3 shows all the objects used in the dataset.

5.1.3 Model Structure

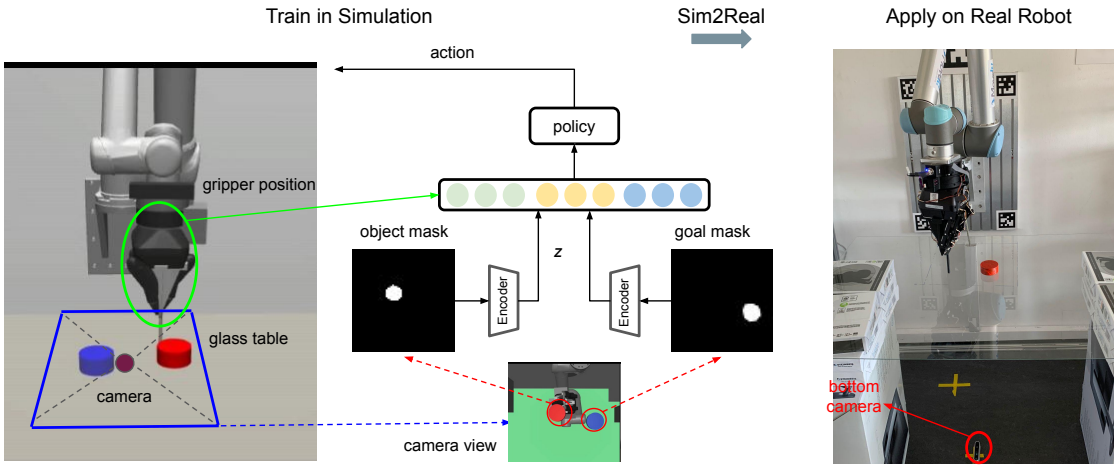


Figure 5.5: Overview: The training process in simulation is shown on the left, and the model is transferred directly to the real platform. The camera is put below the transparent table to avoid possible occlusions between the camera view and the object during manipulation. Color filtering is applied to the image from the bottom camera to obtain both the object (red) and goal (blue) masks. The two masks are fed into the pre-trained autoencoder and then concatenated with the gripper position into a fusion space, where the policy plans.

One of the most challenging parts of visual RL tasks is that the agent needs to simultaneously learn perceptions from high dimensional data and the corresponding control policy. Given an image, a usual perceptron like Multi-Layer Perception (MLP) or CNN encodes all the information from the input, while human beings only pay attention to related information, which is a more efficient way when solving a complicated task. Besides visual input, proprioception is also an essential channel among all the modalities that humans perceive. Inspired by the way human beings solve a task through both vision and proprioception, our vision-proprioception model performs planning in the fusion space.

We embed the object mask m_o and goal mask m_g into a latent space z with the pre-trained encoder e in Section 5.1.2, getting the latent object state $z_o = e(m_o)$ and latent goal state $z_g = e(m_g)$. As the latent variable z samples from Gaussian distribution $q_\theta(z|x) = N(\mu_\theta(x), \sigma^2_\theta(x))$, we take the mean of the encoder $\mu_\theta(x)$ as the state encoding. With the planar position of the pusher $p_r = [x,y]$ easily obtained from forward-

kinematics on the joint-angles during robot manipulation, we construct a fusion state space $S = [p_r, z_o, z_g]$ in which the policy $\pi_\phi(a_t|s, s \in S)$ does its planning. The action $a = [a_x, a_y]$ is a continuous vector, representing the pusher's 2-D velocity in the motion plane. We use squashed action implementation from [108]: $a = \tanh(\bar{a})$, in which $\bar{a} \sim N(\mu_\phi(x), \sigma_\phi^2(x))$.

5.2 Reinforcement Learning with Vision-Proprioception Model

5.2.1 Soft Actor-Critic

SAC is also an off-policy algorithm that uses experienced transitions from previous episodes to update a stochastic policy. SAC is different from DDPG due to the inherent stochasticity of the policy, leading to a smoother target policy. A central feature of SAC is entropy regularization. The policy is trained to maximize the expected return while punishing the decreasing entropy, a measure of policy randomness. This is an exploration-exploitation trade-off: higher entropy means more exploration, which may accelerate learning later. The punishment of a fast decreasing entropy also prevents the policy from converging to a local optimum.

Denote entropy H of variable x as:

$$H(P) = \mathbb{E}_{x\sim P}[-\log P(x)] \tag{5.2}$$

in which $P(x)$ represents its distribution. The optimization object of entropy-regularized reinforcement learning is slightly different from normal ones, because at each time step, the agent gets an extra bonus reward proportional to the entropy of the policy:

$$\pi^* = \arg\max_\pi \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty}\gamma^t\Big(R(s_t, a_t, s_{t+1}) + \alpha H(\pi(\cdot|s_t))\Big)\right] \tag{5.3}$$

In which $\alpha > 0$ is the coefficient. Because of the slight change of the optimization objective, now V^π is updated to include the entropy item:

$$V^\pi(s) = \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty}\gamma^t\Big(R(s_t, a_t, s_{t+1}) + \alpha H(\pi(\cdot|s_t))\Big)\middle| s_0 = s\right] \tag{5.4}$$

In this situation, the relation between V_π and Q_π can be denoted as:

$$V^\pi(s) = \mathbb{E}_{a\sim\pi}[Q^\pi(s,a)] + \alpha H(\pi(\cdot|s)) \tag{5.5}$$

The Bellman equation for Q^π is:

$$Q^\pi(s,a) = \mathbb{E}_{s'\sim P}\big[R(s,a,s') + \gamma V^\pi(s')\big] \tag{5.6}$$

Like TD3 [37], SAC uses the clipped double-Q trick, and takes the minimum Q-value between the two Q approximators. According to the derivations above, the objective to minimize for the Q-networks in SAC is:

$$L(\phi_i, \mathbb{D}) = \mathbb{E}_{(s,a,r,s',d)\sim\mathbb{D}}\left[\Big(Q_{\phi_i}(s,a) - y(r,s',d)\Big)^2\right] \tag{5.7}$$

where y is the target Q-value:

$$y(r,s',d) = r + \gamma(1-d)\left(\min_{i=1,2} Q_{\phi_{targ,j}}(s',\tilde{a}') - \alpha \log \pi_\theta(\tilde{a}'|s)\right), \quad \tilde{a}' \sim \pi_\theta(\cdot|s') \tag{5.8}$$

By maximizing the expected future return and the expected entropy, the optimization objective of the policy net is given by:

$$V^\pi(s) = \mathbb{E}_{a\sim\pi}[Q^\pi(s,a)] - \alpha \log(\pi(a|s)) \tag{5.9}$$

5.2.2 Model Architecture

In this part, we show how to train RL with the model we propose in Section 5.1.3. To improve the sample efficiency of RL, we use the stochastic off-policy algorithm SAC [43] with the goal relabelling trick HER [4]. With an entropy regularization as part of the optimization, the policy is trained to maximize the expected return and the policy entropy, which is a randomness of the policy. The general training process is similar to other off-policy RL algorithms by optimizing two targets: value function $J(Q)$ and policy function $J(\pi)$. The model architecture is shown in Table 5.1. FC(), Conv(), and ConvT() represent the fully connected, convolutional, and transposed convolutional networks, respectively. The arguments of FC() and Conv() / ConvT() are [node] and [channels, kernel size, stride]. The training process of the whole framework is shown in Figure 5.6. VAE is pre-trained before being used in RL; the two encoders for object and goal share the same weights. During the experiment, we can understand what the agent sees by visualizing the reconstruction image from the decoder, which is quite useful for debugging during the training process.

Table 5.1: Model Architecture

Model	**Architecture**
Encoder	Conv([[32,4,2],[64,4,2],[128,4,2],[256,4,2]]) FC([256,6])
Decoder	FC([256,1024]) ConvT([[32,5,2],[32,5,2],[16,6,2],[1,6,2]])
Actor	FC([128,256,64,2])
Critic	FC([128,256,64,1])

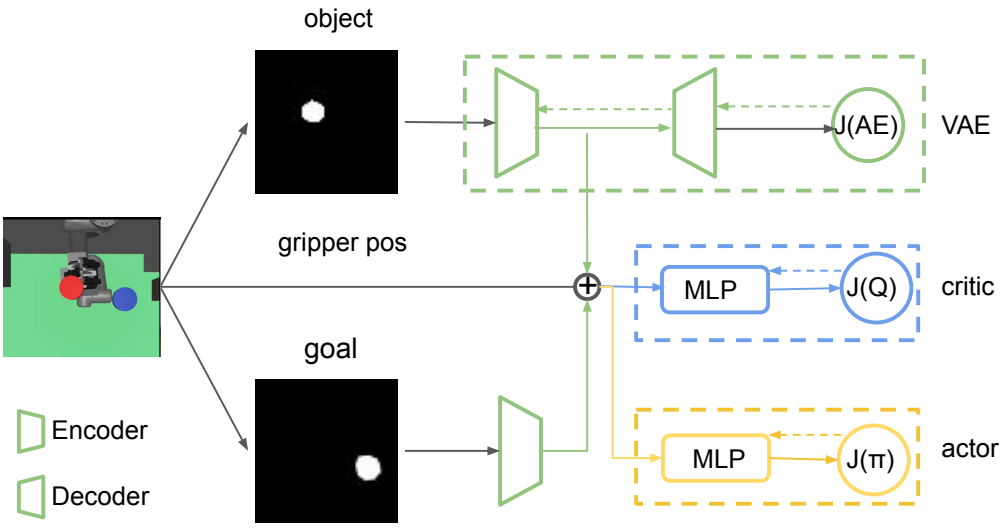


Figure 5.6: Network structure of our RL framework, in which dash arrows represent back propagation. VAE and RL are trained separately, encoders from VAE do not update during the RL training process. The critic and actor part are optimized by value target $J(Q)$ and expected future return $J(\pi)$.

Algorithm 2: RL with Vision-Proprioception Model

Given: Pre-trained Encoder $e_\theta(x)$;
Policy $\pi_\phi(a_t|p_r,z_o,z_g)$;
Value Function $Q_\psi(p_r,z_o,z_g,a)$;
Replay Buffer D : $\{\tau^{(1\sim N)}\}$;

for $n \leftarrow 1$ **to** N *episodes* **do**
  Reset environment;
  Sample and store p_g, x_g in τ^n;
  for $t \leftarrow 1$ **to** H *episodes* **do**
    Get state $S_t = \{p_r, p_o, x_o\}$;
    Encode $z_o = e_\theta(x_o)$, $z_g = e_\theta(x_g)$;
    Select action $a_t \sim \pi_\phi(a_t|p_r,z_o,z_g)$;
    Execute a_t in the simulation;
    Get next state $S_t^{'} = \{p_r^{'}, p_o^{'}, x_o^{'}\}$, reward r_t;
    Store $(S_t, a_t, r_t, S_t^{'})$ in τ^n;
  Sample transitions $(S, a, r, S^{'}) \sim D$;
  Relabel p_g, x_g to p_g^η, x_g^η with method in [4];
  Get new encoding $z_g = e_\theta(x_g^\eta)$ and compute new reward r_n from function 5.10;
  Minimize $J(Q)$ and $J(\pi)$ in Figure 5.6 using $(S, a, r_n, S^{'})$

5.2.3 State Space and Reward Specification

We use the following symbols in Algorithm 2: p_r, p_o, p_g are the ground truth positions of the robot pusher, center of the object, and goal, respectively. Both p_o and p_g are only used to compute the step reward during the training process in simulation, but not during tests or in the real robot experiment. x_o, x_g are pixel observations of the object and goal, z_o, z_g are corresponding encodings. Different reward functions can lead to diverse training results. Nair et al. [143] train the robot in the real world and compute rewards in the latent space of the pixel observation. In the simulation, computing rewards from the ground truth data is also possible, as all ground truth information is readily available. In our work, we consider three different kinds of reward functions. The first is dense reward in latent space:

$$r(z_o, z_g) = -\|z_o - z_g\| \tag{5.10}$$

and the second is sparse reward computed by the ground truth state:

$$r(p_o, p_g) = \begin{cases} -1, & \text{if} \|p_o - p_g\| > threshold \\ 0, & \text{if} \|p_o - p_g\| \leqslant threshold \end{cases} \tag{5.11}$$

The third is dense reward with ground truth state:

$$r(p_o, p_g) = -\|p_o - p_g\| \tag{5.12}$$

The results are compared in the next section.

5.2.4 Sim2Real

We use Robogym [96] as the framework and build our simulation environment according to our UR5 platform. We apply 18 differently shaped objects during the training process (Figure 5.3) and randomize their physical parameters, including mass, sliding, and rotation friction coefficient of the object from a reasonable range. Details of the physical parameters are shown in Table 5.2. We randomize 20 different combinations of physical parameters for each of the objects. At the beginning of each episode, one combination of the physical parameter is chosen and remains unchanged during the episode (50 action steps in the training process). During training and testing, we find an obvious gap between simulation and the real world in all objects' rotational motion: given the same pushing action, the object shows more rotation in the real world.

Table 5.2: Dynamic Parameters and Their Ranges in Simulation

Parameters	**Range**
Size	[4, 15] cm in length, fixed height=2cm
Mass	[0.05, 0.3] kg
Sliding Friction Coefficient	[0.1, 1]
Rotation Friction Coefficient	[0.001, 0.01]
Damping Coefficient	[0.01, 0.015]

5.3 Experiments

We test our algorithm by training an agent in simulation first, then evaluate the training results by applying the model directly to a real robot platform. As is shown in Figure 5.5, the red rectangle represents the object to be pushed, and blue is the goal position. In simulation [134], the non-collision goal object can be rendered conveniently. However, in real experiments, for each episode, we first put the object at the point we want, take an image with the bottom camera and record it as the target image. Underactive human interference, the robot can switch the pushing side accordingly and keep pushing the object to the target consistently. We use the same online controller [115] to translate Cartesian motions of the pusher into joint-space robot commands as in our previous pushing research [27].

5.3.1 Simulation Results

We evaluate our method against two prior model-free state-of-the-art algorithms and do ablation studies to determine how critical each method component is. In Figure 5.8, we compare the learning performances with the pushing success rate. From top to bottom: model trained with different VAE latent space dimensions (first row), reward functions (second row), and input modalities (third row). **Oracle** in the third figure is the training results of a state-based agent on one single object (cylinder, $d = 4cm$). **RIG** [91] takes only the image as input. We set random explorations at the first 200 episodes for each training process, leading to an initial success rate of around 20% for each learning curve.

One episode is considered a success if the final center point distance is within the threshold (5cm) we set in Equation 5.11. Orientation error is not considered in the reward function. To our best knowledge, Reinforcement Learning with Imagined Goals (RIG) is the state-of-the-art algorithm for the visual pushing task [90, 91]. We choose RIG as the baseline method. Besides, we also give the results with direct access to state information, including the robot's end-effector position and the object's pose (Oracle). However, because the interaction dynamics of differently shaped objects differ a lot from each other, one state-based policy can neither learn to push all 18 candidates used in our experiment nor train on one specific object and then generalize to another object. Therefore, the **Oracle** learning curve is the learning result of pushing **one specific cylinder**. For the other experiments, **one random object** is selected from all the candidates at the beginning of each episode.

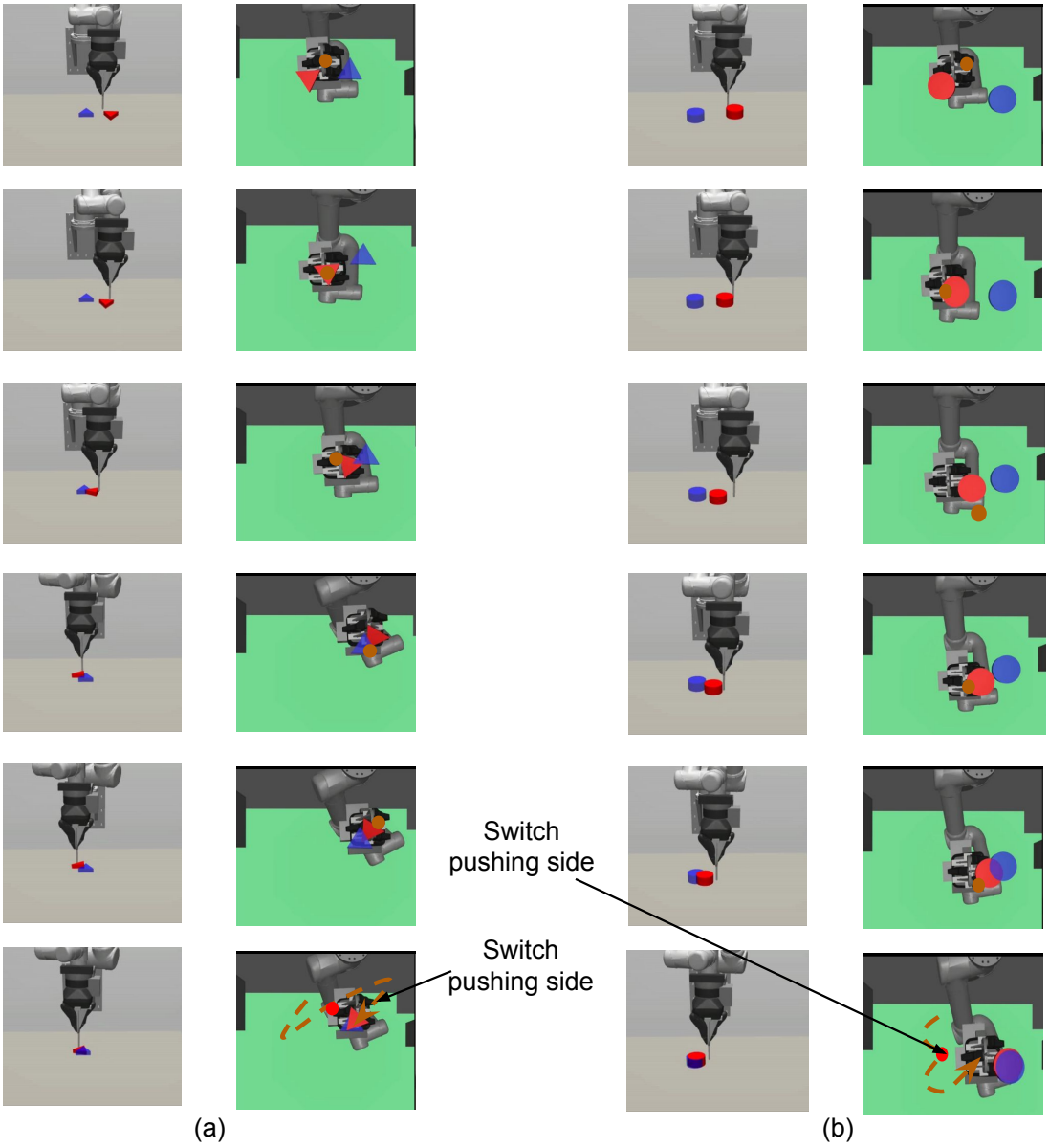


Figure 5.7: The figure shows two pushing processes: (a) triangle and (b) cylinder. Both the initial and target position is generated randomly at the beginning of each episode. We show both the front view (first column) to give an overall robot-object scene and the bottom view (second column) to show what the robot sees. The transparency of the table is set to 0.3 in the simulation. The brown point and the dashed line represent the pusher position and trajectory, respectively. In both (a) and (b), we can see that the robot learns to firstly choose a proper initial pushing direction and switch the pushing side when the object deviates from the target. The final orientation error in b) is explained in Section 5.3.1 .

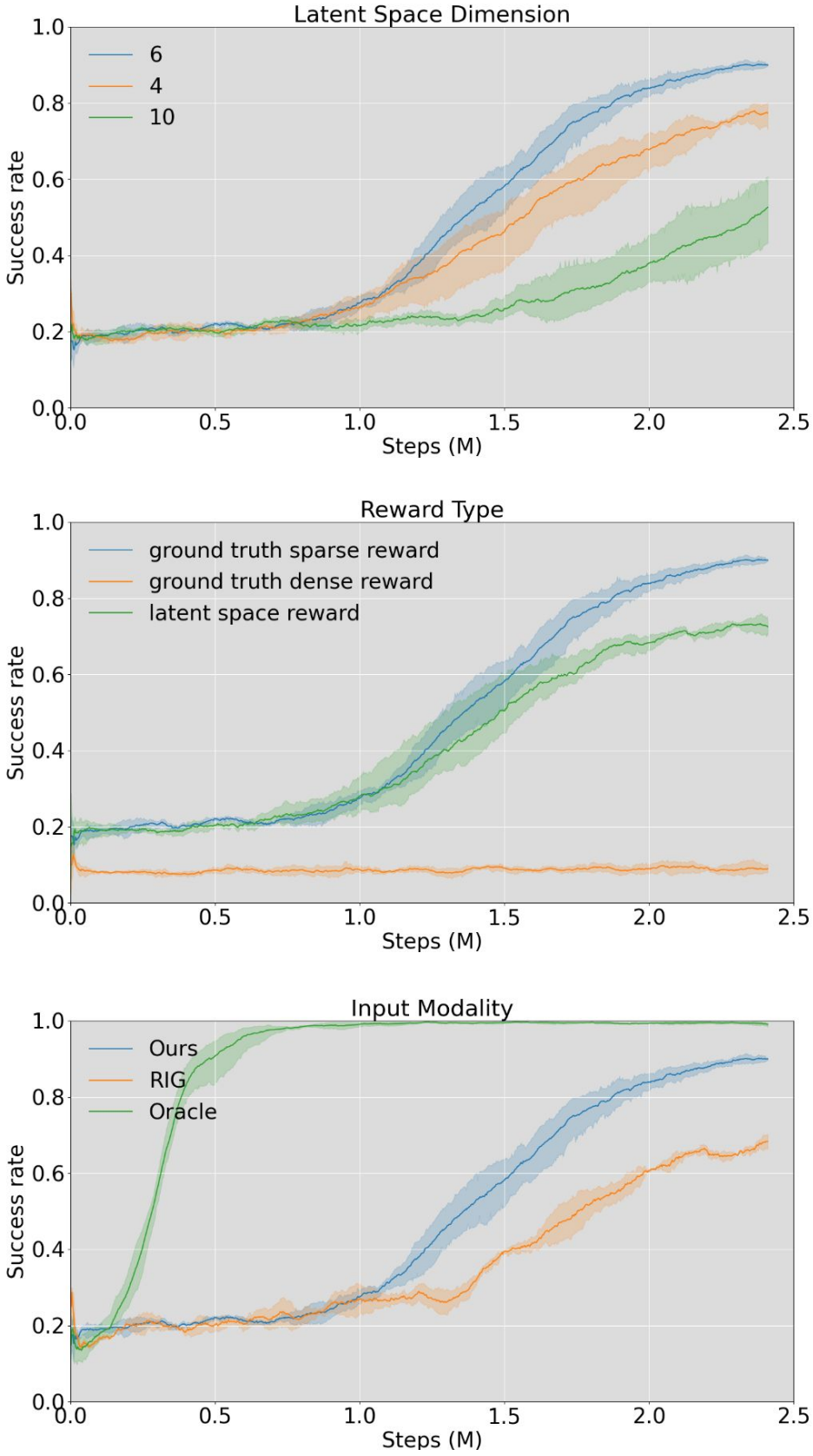


Figure 5.8: Learning curves in simulation.

During experiments, we find that two components have the most significant influence on the training performance: 1) the dimension of VAE latent space and 2) the reward function type. We first compare learning results using VAE models with different latent space dimensions (Figure 5.8 first figure). During all training processes of the VAE, as the training epoch keeps rising, the model goes from under-fitting to over-fitting. Most of the over-fitting happens between the range of $(60, 90)$ epochs. For the models with different latent spaces, we choose the saved ones, which occur just before the over-fitting happens. Through analysis, we find that the best latent dimension is 6. This may be because it is a suitable dimension to remember the shape and pose of the object, and at the same time, its representation is not too complicated for the agent to learn an effective policy.

One of the differences between our work and [91] is that we train our model in simulation and use it in on a real robot. This reduces training time considerably and facilitates access to the environment's ground truth state, while in the former work, the policy is directly trained in the real world. We apply three different reward functions (Section 5.2.3) in the training time and get the following results: 1) sparse reward from ground truth state (Equation 5.11) leads to the best pushing performance (90% success rate), 2) dense reward in latent space (Equation 5.10) can also guide the agent to a working policy, 3) dense reward from ground truth state is invalid (Figure 5.8).

5.3.2 Real Robot Verification

This part evaluates whether our model can be transferred to the real world and manipulate unseen objects (with similar shape and size) without any fine-tuning. All the models in our method, including the VAE and policy network, are trained in simulation. The robot setup and debugging interface are shown in Figure 5.9. We visualize both the original object mask (1c) and its reconstruction (1a) in real-time to check whether the information in latent space is correct or not. The object is put at an initial pose (marked with a blue box in (e)), then the bottom camera takes an image and records it as the target image, which is the blue mask in (b). The robot keeps pushing the object to the target. In 1b) a comprehensive view is given: the blue and red rectangles represent the target and real-time object pose, respectively, the yellow cross represents the pusher position. 1c) and 1a) are the real-time object masks after the color filter and the mask reconstruction from the decoder. We find it quite useful to visualize the two masks during debugging for the experiment. 1d) shows what the robot sees from the bottom camera. From (1) to (3), we can also see that the robot is switching the pushing side as in the simulation. In (3), the first pushing target is reached, and we give an active interference in (4). Till (6), the robot adjusts the pusher consistently and finishes the second pushing process successfully.

During experiments, we find that the interaction dynamics in the real world are different from that in simulation, especially on the rotation motion of the object. The objects turn quickly around the vertical axis in the real world under the robot's pushing actions. Even though we randomize the physical parameters in simulation from a wide range, the gap cannot be eliminated. We assume this is because the simulator simplifies the contact model to save computation resources. The position distribution frequency

from 500 episode trajectories is shown in Figure 5.10. We set the same robot working space and goal randomization space for simulation and the real world. However, because the objects rotate easily in the real world, the robot needs more adjusting motions to push the object to goal positions than in simulation. Most of these unexpected adjusting motions happen around the workspace center, making the object and pusher trajectory distribution more intensive in the center part. To analyze the transfer performance of our model, we measure the distance between the goal position of the object and the final position (without orientation), also the corresponding time consumption. We test 3 objects from the training dataset and two novel objects (similar shape but different size) in the experiment. Table 5.3 shows the pushing performance comparison on average final position error and the corresponding time consumption between simulation and the real world. Objects from the last two rows are novel objects (outside the training set but in our pre-designed size range. More adjusting pushing actions also mean more time consumption for each push. As the threshold we set in the reward function 5.11 is 5 cm, the mean distance between goal and final position is within 5 cm in both simulation and the real world.

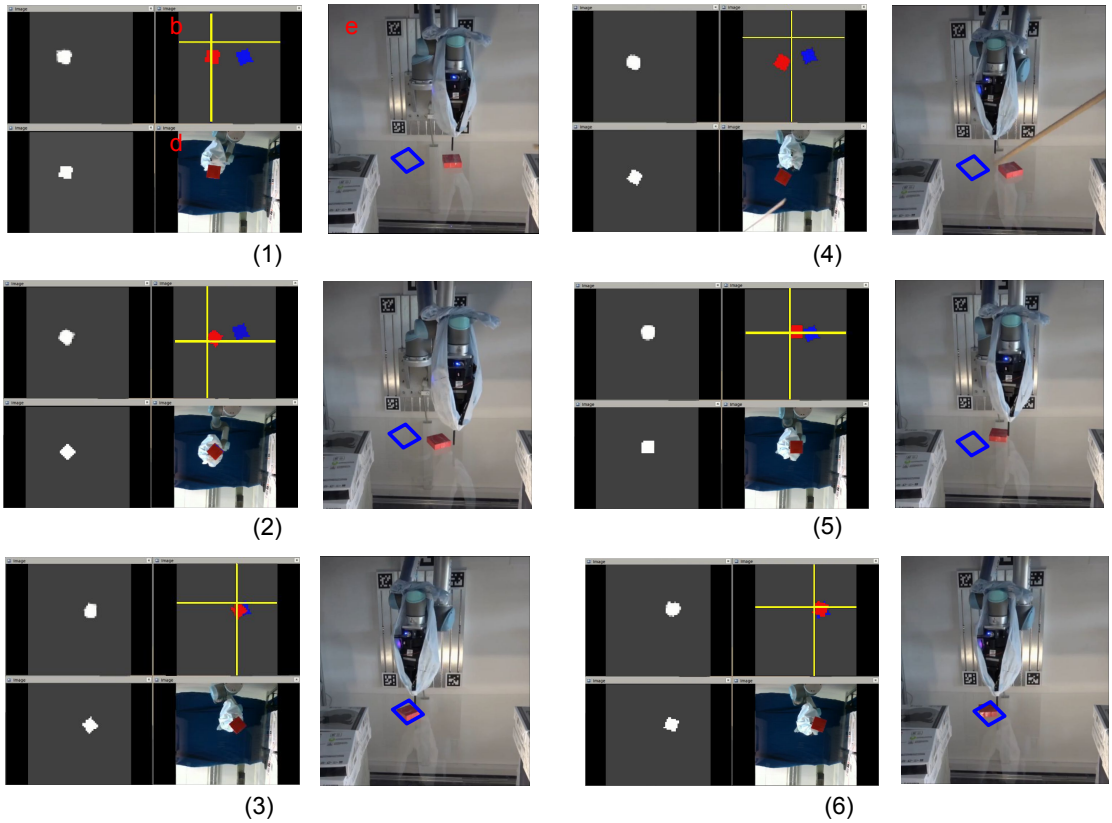


Figure 5.9: A cuboid pushing process under human interference on a real robot platform. The first pushing process is shown from (1) to (3), the interference happens in (4).

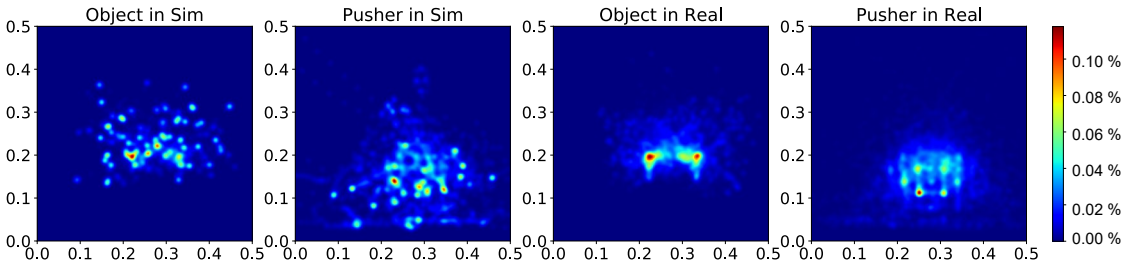


Figure 5.10: Distribution of the object and pusher's position during tests in both simulation and real experiments. The colorbar represents the occurrence frequency.

Table 5.3: Comparison of Pushing Results

	In Simulation	In Real World
Triangle ($l = 8$ cm)	3.2 (cm) / 5.5 (s)	3.5 (cm) / 22.9 (s)
Cylinder ($d = 6$ cm)	3.5 (cm) / 4.6 (s)	3.7 (cm) / 18.6 (s)
Square1 ($l = 8$ cm)	2.9 (cm) / 4.3 (s)	3.1 (cm) / 19.4 (s)
Square2 ($l = 10$ cm)	3.6 (cm) / 5.6 (s)	3.9 (cm) / 21.5 (s)
Rectangle ($l_1 = 6, l_2 = 8$ cm)	4.2 (cm) / 6.2 (s)	4.5 (cm) / 23.3 (s)

5.4 Summary

In this chapter, we present a self-supervised pixel-based method that can encode visual inputs into latent space and fuse with a robot's proprioception into one model to solve the task of object pushing and achieve a competitive advantage over a state-based method. The latter method can only be trained on a single object without generalizing it to other objects. Our model is trained in a simulation environment and can be applied on the real robot platform without any fine-tuning. Real experiment results show that the model is of high robustness to unseen objects.

The core idea of our method is to force the agent only to pay attention to valuable information in the image and fuse the encoding with information from other perceptions in the environment, making use of all task-relevant inputs from multiple channels. We believe our method can be taken as an inspiration to extract useful information from different modalities and fuse them for end-to-end decision-making problems, improving learning efficiency and performance in real robot RL tasks.

One limitation of the current method is that the agent cannot judge intelligently whether the information is helpful to the task or not. Learning and inferring task-relevant information from sequential observations could solve more complicated tasks and make our algorithm more generalizable.

Chapter 6

Attention Augmented Reinforcement Learning for Object Pushing

In the previous chapters, two different methods were proposed for object planar pushing. The model-based method from chapter 4 builds a fusion motion prediction model first and plans the action of the robot end-effector with RMPPI. To endow the controller with the ability to switch pushing sides intelligently and get rid of the dependence on Apriltag, we propose the model-free visual-pushing method in chapter 5, taking the filtered object masks obtained from the bottom camera as input. However, the requirement of mounting the camera below the transparent table makes the experiment setup complex, and the lighting in the room also influences the performance of the experiment. To simplify the experiment setup, get rid of the transparent table and the bottom camera, a new planar pushing method taking only a front view is introduced in this chapter.

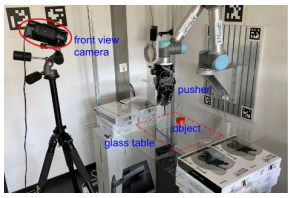
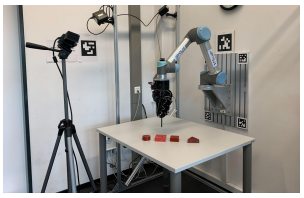
	State Based Pushing	Visual Pushing (mask)	Visual Pushing (raw image)
Setup			
Input	State (object pose)	Object mask (bottom camera)	Raw image (front camera)
Algorithm	RMPPI (model based)	SAC (model free)	RL + imitation learning (model free)
Weakness	AprilTag dependence & can't switch pushing side	Complicated setup	**Hard to train**

Figure 6.1: Comparison of different pushing methods proposed in the thesis.

Section 6.1 introduces a self-supervised attention mechanism to predict the attention map, which corresponds to the motion part (maybe the robot arm, or the manipulating object, or both) among all anchor-target pairs in the training dataset (detailed in Section 6.1.3). The anchor-target pair is easy to collect during the training process of RL in simulation. Section 6.2 proposes a learning framework that integrates RL and the

pre-trained attention mechanism. Through comparison experiments, the attention augmented reinforcement learning is proven effective and leads to better performance than the original framework without attention.

6.1 Self-supervised Attention Mechanism

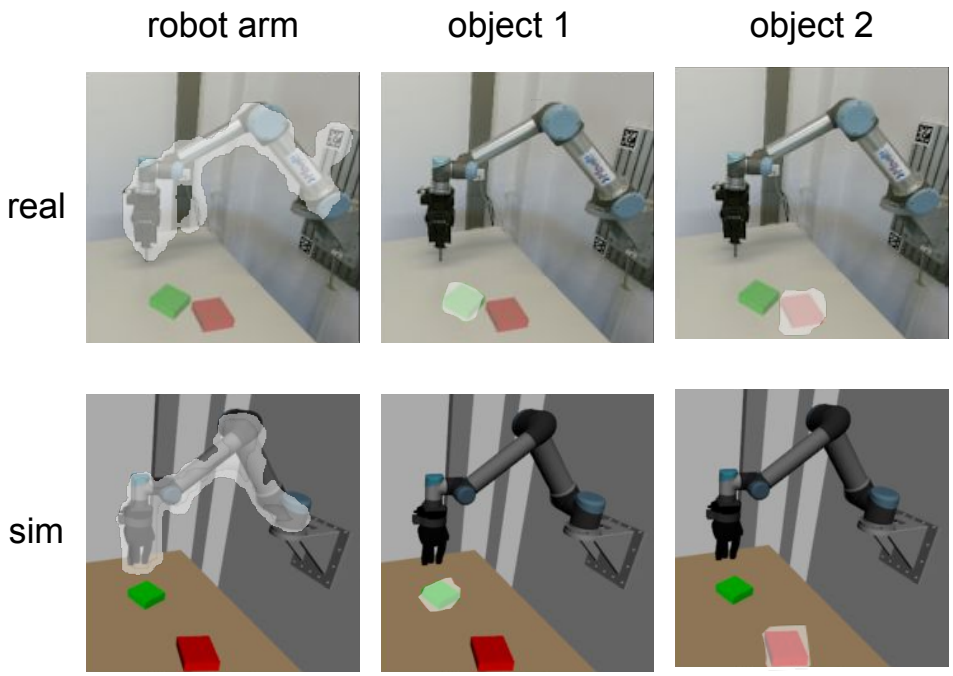


Figure 6.2: Given different anchor-target pairs as training dataset, our model can learn to pay attention to different motion parts (robot and different objects separately) from one same image. The figure shows the training results of our UR5 platform in real word and simulator.

Attention map is defined in [130] as a scalar matrix representing the relative importance of layer activations at different 2D spatial locations to the target task. In robot tasks, most task-related parts in the vision are the motion parts. Taking the pushing task in Figure 6.2 as an example, the moving parts in the image are the robot arm, the object to push (green), and the target pose (red object). These components are the task-related parts in the image which may attract a human's attention while solving the task. Paying more attention to some specific parts while neglecting unrelated background increase our working efficiency to a large degree. Recent end-to-end learning framework [67, 69] learn control strategy by mapping high-dimensional visual inputs to robot actions directly. This requires the model to learn the visual representation and corresponding control policy at the same time, which always leads to poor strategy and low learning efficiency [69]. Inspiring works [84, 88, 119] propose to combine attention mechanism with reinforcement learning to increase agent's performance and interpretability. However, all of these work only add extra attention mechanisms into the model, increasing the complexity of the model, but the RL and control strategy is still not separated apart. In

this work, we address the problem by decoupling the two parts. The output attention map can be used for downstream control tasks by training a self-supervised attention model and highlighting the task-related parts in the image.

Humans tend to focus on different parts of an image for different manipulation purposes. By adding different anchor-target image pairs into the dataset (detailed in Section 6.1.3), our model can also learn to pay attention to different components from the same image. As is shown in Figure 6.2, we train the model with different anchor-target pairs, and the model successfully learns to highlight different parts.

6.1.1 Attention Mechanism Structure

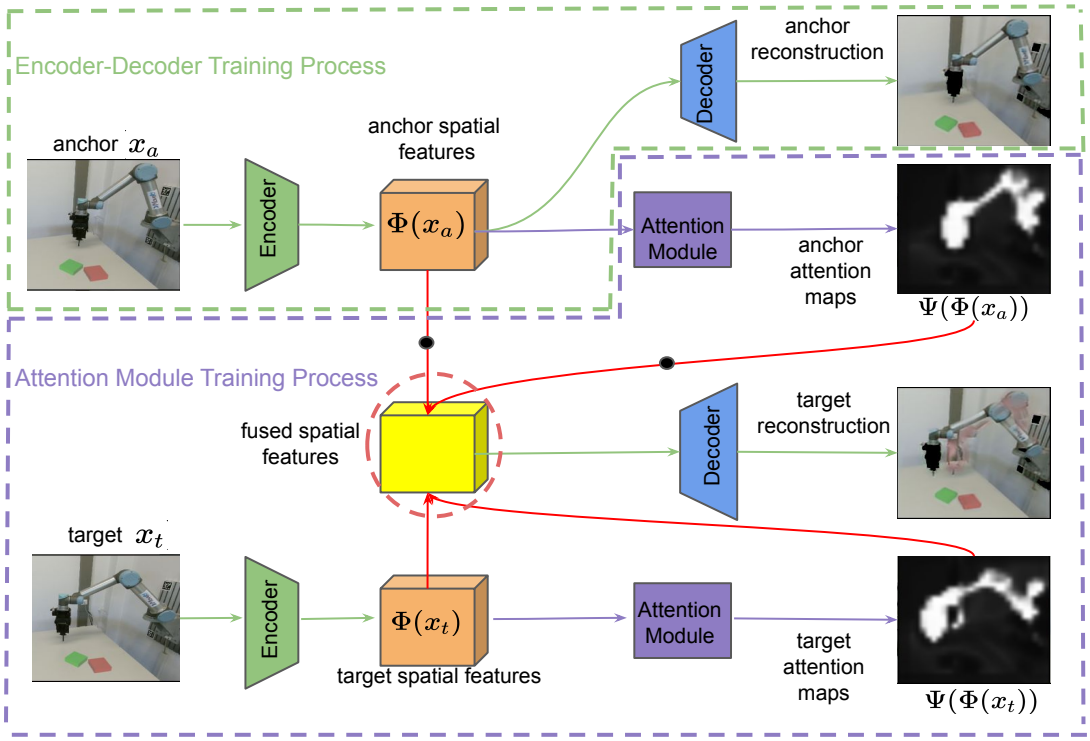


Figure 6.3: Illustration of the framework. The green and purple frames in the dashed line represent the Encoder-Decoder and Attention module training process, respectively. The spatial features $\Phi(x)$ is taken as the intermediate variable, used both for the training of the Encoder-Decoder module and for spatial feature fusion to reconstruct the target image in the Attention module training process (spatial features fusion is illustrated in Figure 6.4). The black point in the figure represents the gradient stop. By learning to transform an anchor image x_a into another target image x_t, our model is forced to find and highlight the motion part among all the image pairs in the dataset.

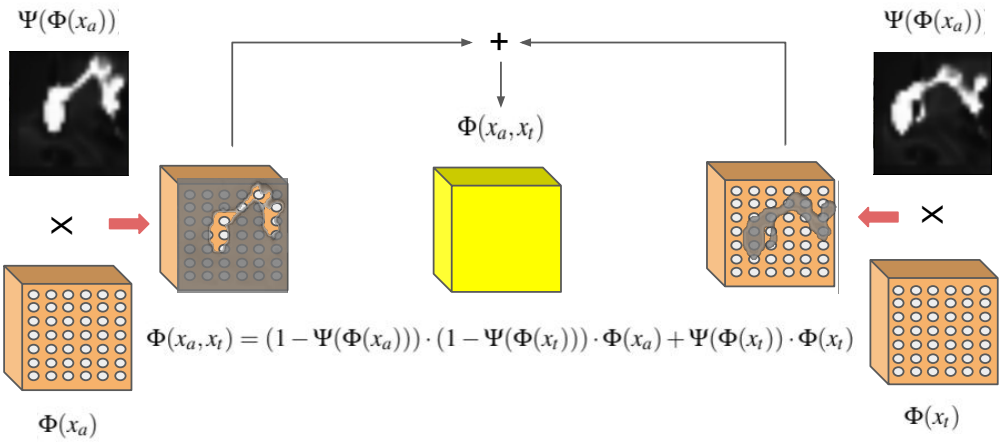


Figure 6.4: Spatial feature fusion process (red dash circle in Figure 6.3) of the attention transporter.

As is illustrated in Figure 6.3, given an input RGB image $x \in \mathbb{R}^{H\times W\times 3}$, our objective is to predict the attention map $\Psi(\Phi(x)) \in \mathbb{R}^{H^{''}\times W^{''}}$, which corresponds to the motion part among all anchor-target frames pairs (x_a, x_t) in the training dataset. Before training, we collect our own dataset in a way that all of the anchor-target frame pairs (x_a, x_t) differ only in the part which we hope to highlight. It could be the whole robot, part of the robot or even one of the manipulation target in the workspace like in Figure 6.2. The whole training process can be divided into the Encoder-Decoder training part and Attention Module training part: (i) the Encoder (ConvNet block in green frame from Figure 6.5) module turns the image x into spatial feature $\Phi(x) \in \mathbb{R}^{H^{'}\times W^{'}\times 64}$; then the Decoder (blue frame in Figure 6.5) maps from the spatial features $\Phi(x)$ back to an reconstruction image $x^{'}$. The weights of the two networks are trained by minimizing the pixel-wise $L2$ reconstruction loss $\|x^{'} - x\|$; (ii) the anchor and target spatial features $\Phi(x_a), \Phi(x_t)$ go into the Attention Module (detailed in Section 6.1.2) and comes out as anchor and target attention maps $\Psi(\Phi(x_a)), \Psi(\Phi(x_t))$. Then the fused spatial features are generated by the spatial feature fusion module (Figure 6.4) through removing the highlight part from the anchor spatial features $\Phi(x_a)$ and compensating with corresponding part of the target spatial features $\Phi(x_t)$ as proposed in [64]:

$$\Phi(x_a, x_t) = (1 - \Psi(\Phi(x_a))) \cdot (1 - \Psi(\Phi(x_t))) \cdot \Phi(x_a) + \Psi(\Phi(x_t)) \cdot \Phi(x_t) \tag{6.1}$$

In summary, the spatial features from the anchor image $\Phi(x_a)$ at the highlight part $\Psi(\Phi(x_t))$ are replaced with the spatial features from the target image $\Psi(\Phi(x_t)) \cdot \Phi(x_t)$; at the same time, those highlight spatial features $\Psi(\Phi(x_a))$ from the anchor image are set to zero. Then the Decoder maps the fused feature $\Phi(x_a, x_t)$ back to the target reconstruction image $x^{'}_t$. By minimizing the $L2$ loss $\|x^{'}_t - x_t\|$ in pixel wise, we enforce the Attention Module to learn to focus on the motion part.

6.1.2 Trainable Attention Module

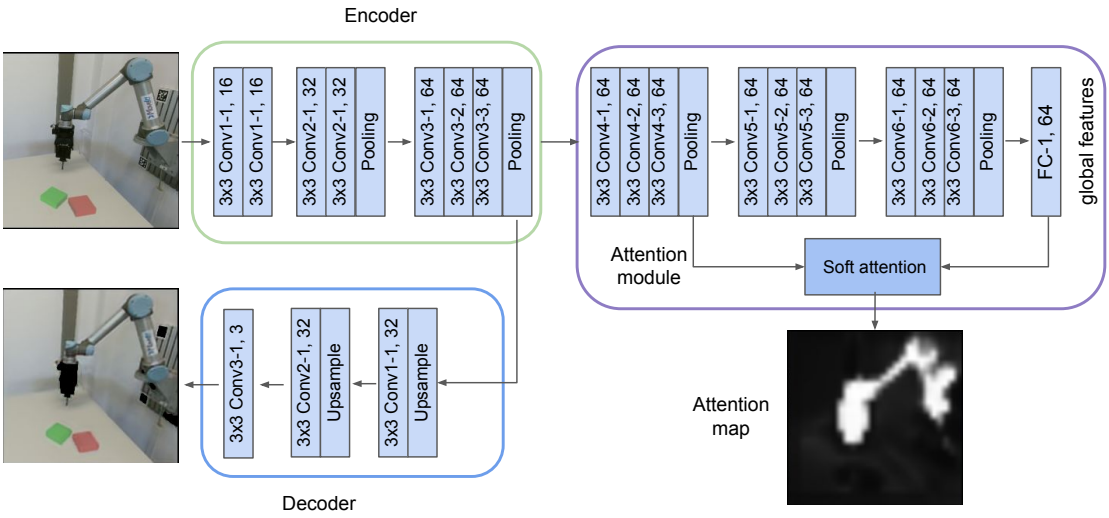


Figure 6.5: Network architecture of the module used in our model. We use the VGG as the backbone but choose lower channel numbers in all Conv blocks to reduce the computation. The Soft attention block takes the feature vectors output by the Conv4 as inputs and gives the attention map as output. The Encoder and Decoder modules are trained first through reconstruction loss and keep weights unchanged during the training of the Attention module.

As illustrated in Figure 6.5, the Attention Module (in purple frame) takes the spatial features $\Phi(x)$ from the Encoder as input, first extracting a set of feature vectors with Conv4 block

$$L = \{l_1, l_2, ..., l_n\} = Conv4(\Phi(x)) \in \mathbb{R}^{H''' \times W''' \times 64} \tag{6.2}$$

in which each l_n is the output vector at spatial location i of $n = H''' \times W'''$ total spatial locations. The global feature $g \in \mathbb{R}^{1\times 64}$ is the encoding vector of the entire image, following after a series of Conv blocks and at last a fully connected layer FC-1. Then we construct a compatibility score function C with trainable variable u as proposed in [57], which takes two vectors of equal dimension and gives a scalar compatibility scores as output:

$$(L, g) = \{c_1, c_2, ..., c_n\} \tag{6.3}$$

in which:

$$c_i = \langle u, l_i + g \rangle, \quad i \in \{1...n\} \tag{6.4}$$

Then the compatibility scores are normalised by the softmax operation:

$$s_i = \frac{exp(c_i)}{\sum_{i=1}^{n} exp(c_i)} \quad i \in \{1...n\} \tag{6.5}$$

As the values in attention map matrix $\Psi(\Phi(x_t))$ from Equation 6.1 need to be ranged in $[0, 1]$ (representing from zero attention to full attention at the corresponding location), we rescale the compatibility scores by:

$$a_i = exp\big(\delta * (s_i - s_{max})\big) \quad i \in \{1...n\} \tag{6.6}$$

in which

$$s_{max} = \max\{s_1, s_2, ..., s_n\} \quad i \in \{1...n\} \tag{6.7}$$

δ is a hyperparameter during the training process, we find $\delta = 1000$ gives pretty good performance. Finally, the attention map can be denoted as:

$$\Psi(\Phi(x)) = \{a_1, a_2, ..., a_n\}, \quad a_i \in [0, 1] \tag{6.8}$$

6.1.3 Dataset Collection

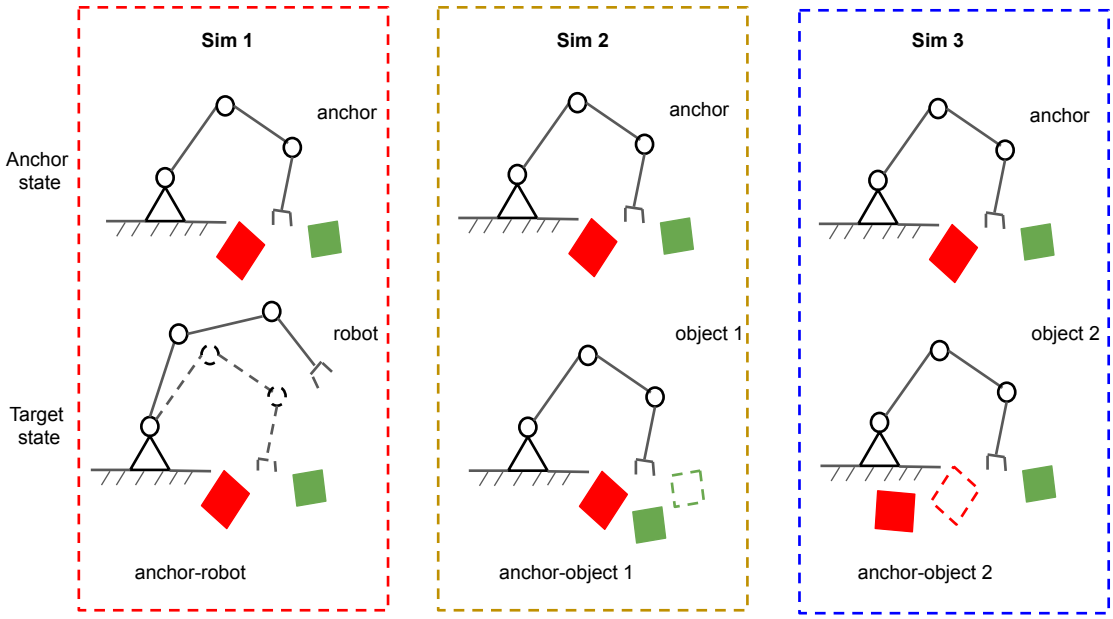


Figure 6.6: Illustration of the anchor-target pair generation process in robot scenario. We run three simulators in parallel to generate different anchor-target pairs (anchor-robot, anchor-object 1, anchor-object 2). The process can be divided into two steps: 1) initialize the robot and object state randomly, take as the anchor state, and save as anchor image. 2) Individually change the robot, object 1 and object 2 states in 3 simulators, save as target image. The dashed line in the target state represents the original pose of the moving target. The dataset with three groups of anchor-target pairs is generated by repeating the above generation process 3000 times.

We first collect the dataset, then train the model with the proposed method. We use Intersection over Union (IoU) to evaluate the performance of our algorithm. It is slightly different between the robot scenarios (UR5 and Fetch robot) and the DeepMind Control Suit. For the robot scenarios, to train the model to highlight different components (robot arm, object 1, and object 2) in the image, we separated the dataset into three groups: anchor-robot, anchor-object 1, and anchor-object 2. The dataset generation process is shown in Figure 6.6. Take the real UR5 robot as an example (Figure 6.7); the anchor image is generated by randomizing the initial state of the robot and two objects. Through moving only the robot to a new pose, the anchor-robot image pair is obtained. Similarly, we keep the robot still and change the object pose only to get the anchor-object pair.

We collect three image pairs each time by running three simulators in parallel. The procedure is repeated 3000 times to collect a dataset with 3000 image pairs for each anchor-target group. When we collect images on the real robot platform, we use virtual objects.

It is easier to collect datasets in the DeepMind Control Suit. We let the agent do random explorations in the environment, at the same time, collect the pixel observations each time-step, and reset the environment every five time-steps. We collect 1000 images for each environment which can give us 9.99×10^5 different image pairs. There is no manual labeling during the dataset collecting.

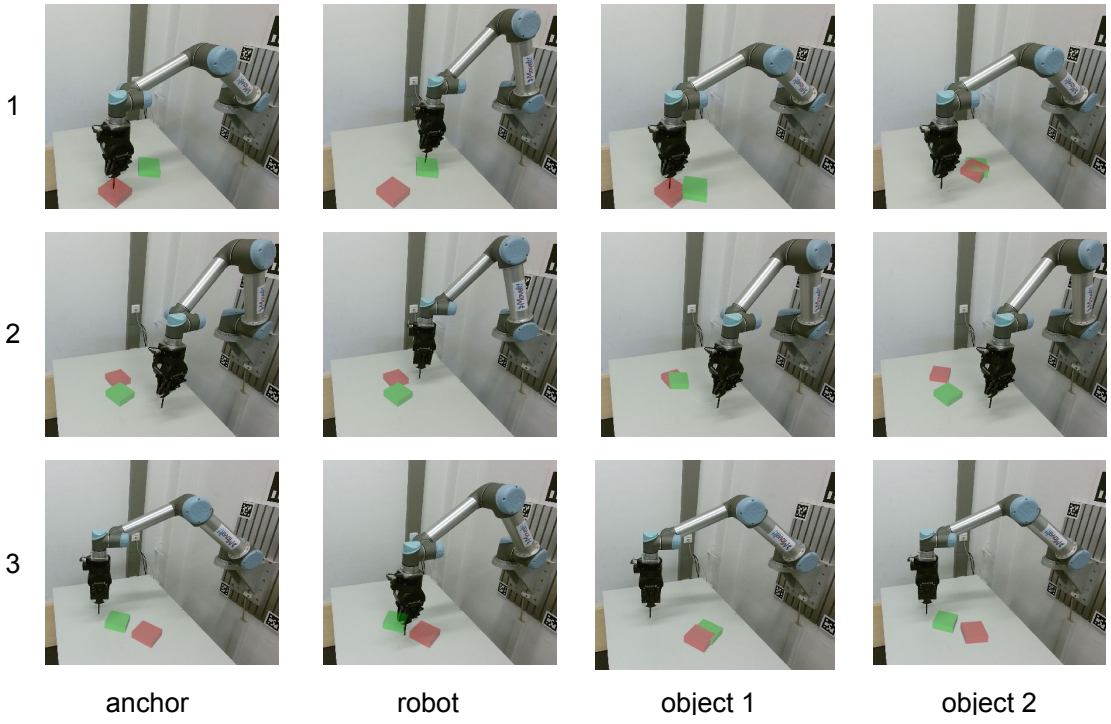


Figure 6.7: Example of three groups of anchor-target pairs from real UR5 dataset. In each group, the first column is the anchor image, forming different anchor-target pairs with images from "robot", "object 1", and "object 2" columns separately.

6.1.4 Training Result Analysis

We use IoU as the metric to evaluate the performance of our model. We choose 50 images randomly from each group of the dataset and mask the motion part in the images manually. As all the pixel values a_i from the attention map matrix are ranged in $[0,1]$, we set a threshold $\sigma \in [0,1]$ to classify the pixel values with:

$$a_i = \begin{cases} 0 & if \quad a_i < \sigma \\ 1 & if \quad a_i >= \sigma \end{cases} \tag{6.9}$$

Then we compute the IoU of the ground truth masks and the threshold cut of the attention map $\Psi(\Phi(x))$. During experiments, we find that $\sigma = 0.8$ can lead to the best

IoU for the robot scenarios while $\sigma = 0.9$ is better for DeepMind Control Suit. The result is shown in Table 6.1 and Table 6.2. In the table, "a-r", "a-o1", and "a-o2" are short for "anchor-robot", "anchor-object 1", and "anchor-object 2". Both the attention map and the corresponding threshold cut are shown in Figure 6.8 and Figure 6.9. For the robot scenarios, we only give the result of the anchor-robot group as the training results of the anchor-object group are quite similar to that of the DeepMind Control Suit.

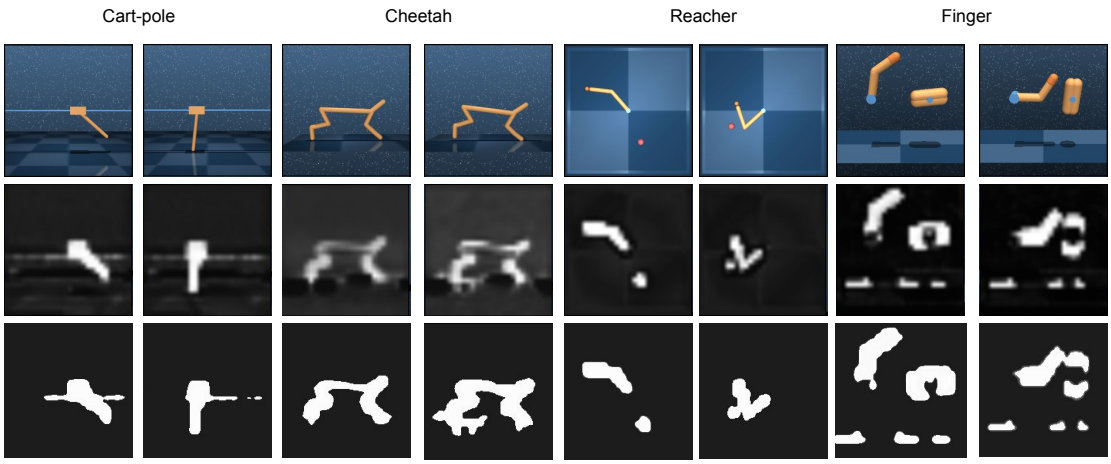


Figure 6.8: Attention maps and the corresponding threshold cut when $\sigma = 0.8$. As we can see, the model learns to find the motion part successfully from the background. In the Cart-pole, Cheetah, and Finger samples, errors are caused by some disturbance components like the rail of the cart, the reflection from the ground, and the shadow on the ground (which is also moving together with the robot). While the attention maps from Reacher samples are quite clean.

Table 6.1: IoU of Robot Environment Attention Maps on Different Thresholds

σ	Fetch	sim, a-r	sim, a-o1	sim, a-o2	real, a-r	real, a-o1	real, a-o2
0.7	0.65	0.83	0.82	0.82	0.81	0.80	0.81
0.8	0.67	**0.86**	**0.83**	**0.85**	**0.84**	**0.86**	0.82
0.9	**0.71**	0.82	0.79	0.82	0.83	0.81	**0.83**

Table 6.2: IoU of Deepmind Control Suit Attention Maps on Different Thresholds

σ	Cart-pole	Cheetah	Reacher	Finger
0.7	0.66	0.64	0.70	0.57
0.8	0.71	0.68	**0.75**	0.59
0.9	**0.75**	**0.72**	0.70	**0.68**

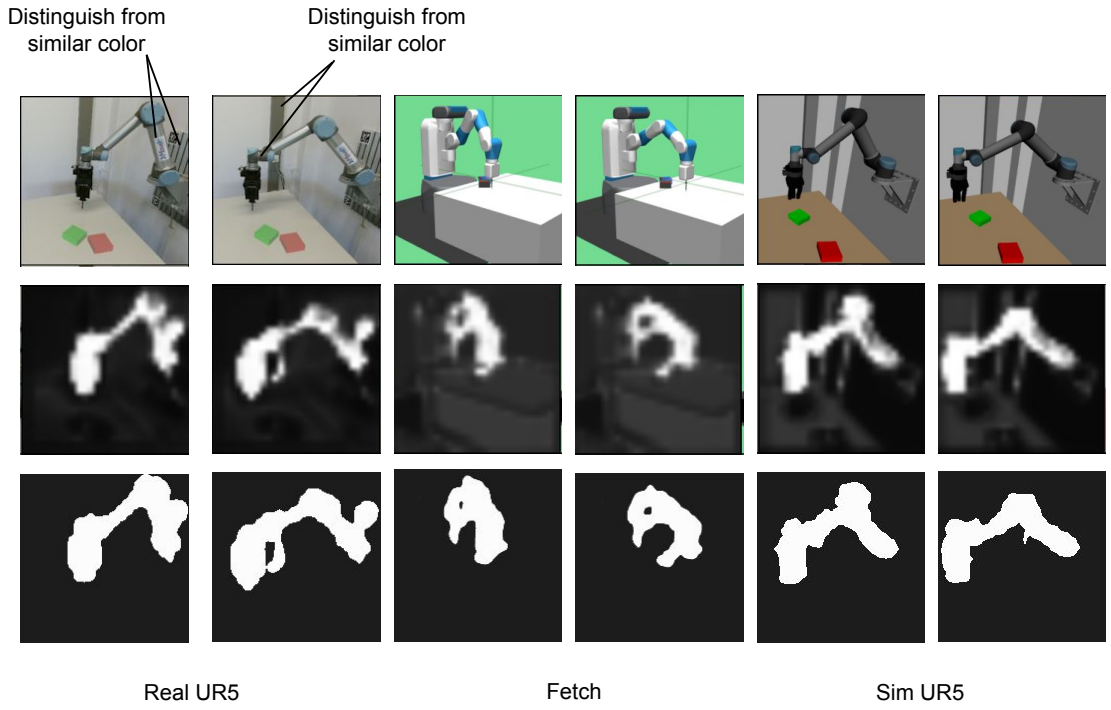


Figure 6.9: We can observe some interesting details from the original images and their corresponding attention maps in the real UR5 samples: 1) the model learns to distinguish the arm from the mounting base even though they are similar in the color; 2) the joint part of the robot arm also shares very close color (gray) to the vertical pillar beside the wall, our model also learns to distinguish them from each other. Both mean that color is not the only classification criteria of the model; it does learn some deeper features such as shape and appearance.

6.2 Attention Augmented Reinforcement Learning Framework

Taking the image from the front view camera as the only input makes the training process even harder than our previous proposed vision-proprioception model for the following reasons:

- The input to the model is the high-dimensional RGB image from the front camera now. In the previous model, we used a one-channel object mask.
- In the new model, the network has to learn the relative position between the objects and the robot end-effector from the image. However, the position of the end-effector is obtained directly from forward-kinematics on the joint-angles.
- The network in the new model has to simultaneously learn the visual representations (perception problem) and the control policy (control problem). In comparison, the representation learning of the visual part is pre-trained in the previous vision-proprioception model.

To deal with the problems mentioned above, we propose the attention augmented reinforcement learning algorithm. The attention mechanism (Section 6.1) is integrated into the RL framework to enforce the policy net to focus on the task-relevant part (object and robot arm) in the image, ignoring the changing background during robot motion.

6.2.1 Pre-training of the Attention Mechanism

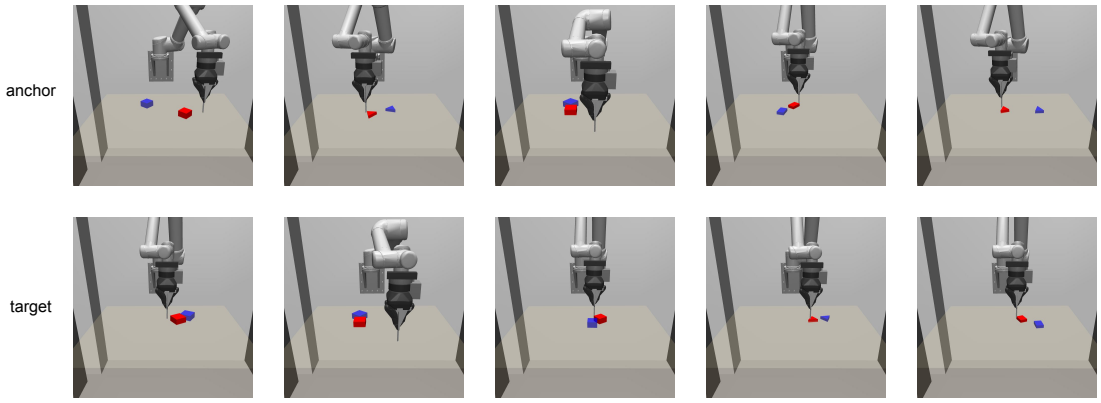


Figure 6.10: Samples from dataset of UR5 in simulator, both robot arm and objects are different from each other in one anchor-target pair.

As is discussed in Section 6.1, the attention module can learn to focus on different components in the same scenario. In this pushing task, task-relevant information includes both the object and the robot arm's pose. Therefore, in the dataset, the target image should differ from the anchor image on both elements. Some samples of image pairs are shown in Figure 6.10. After training, the attention module learns to highlight the three components (manipulating the object, the target, and the robot arm) in the image. The training result is shown in Figure 6.11.

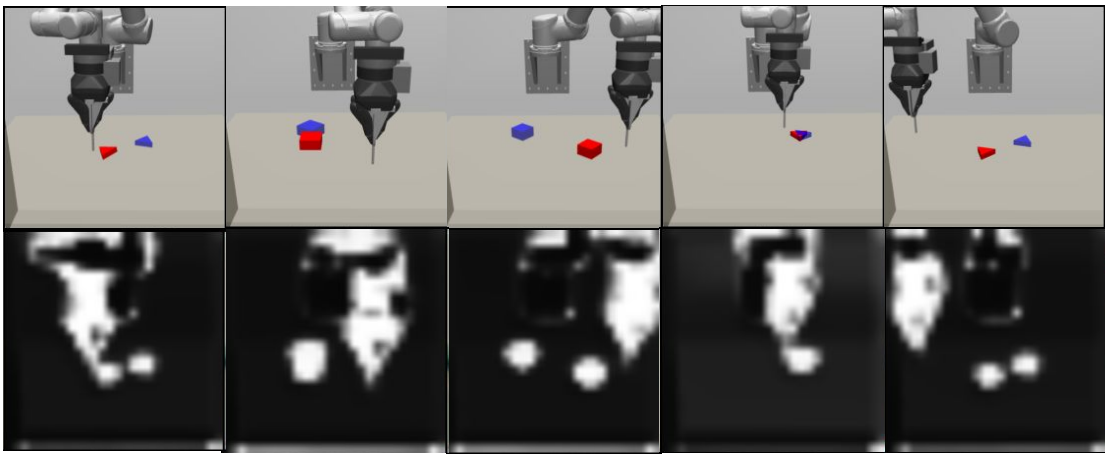


Figure 6.11: Attention maps of our UR5 platform in simulation.

6.2.2 Trust Region Policy Optimization

TRPO is used as the student policy in the proposed reinforcement learning framework 6.12. Different from the reinforcement learning algorithm used in previous chapters (DDPG in chapter 3 and SAC in chapter 4, two state-of-the-art off-policy algorithms). As the goal information of each episode is not provided explicitly by the environment, the goal relabelling of HER can not be performed easily with the RGB images as the only input.

TRPO updates policies by taking the largest step possible to improve performance under a special constraint that prevents the new policy from deviating too much from the old policies. The KL-Divergence is used as the constraint to measure the distance between the two policy distributions. The normal policy gradient keeps new and old policies close in parameter space, but even one bad step that causes slight differences in parameter space may lead to performance collapse. TRPO is proposed to avoid this collapse and improve performance monotonically. This can be presented with the equation as:

$$\theta_{k+1} = \arg\max_{\theta} L(\theta_k, \theta) \quad s.t. \quad \bar{D}_{KL}(\theta \parallel \theta_k) \leq \delta \tag{6.10}$$

where π_θ denotes a policy with trainable parameters θ, $L(\theta_k, \theta)$ is the surrogate advantage which measures the relative performance of the policy π_θ to the old policy π_{θ_k}:

$$L(\theta_k, \theta) = \mathbb{E}_{s,a\sim\pi_{\theta_k}}\left[\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s,a)\right] \tag{6.11}$$

and the average KL-divergence between the old policy and new policy across the visited states can be presented as:

$$\bar{D}_{KL}(\theta \parallel \theta_k) = \mathbb{E}_{s\sim\pi_{\theta_k}}\left[D_{KL}(\pi_\theta(\cdot|s) \parallel \pi_{\theta_k}(\cdot|s))\right] \tag{6.12}$$

TRPO makes the approximation to get an easy update format through Taylor expand the objective and constraint around θ_k:

$$\begin{aligned} L(\theta_k, \theta) &\approx g^T(\theta - \theta_k) \\ \bar{D}_{KL}(\theta \parallel \theta_k) &\approx \frac{1}{2}(\theta - \theta_k)^T H(\theta - \theta_k) \end{aligned} \tag{6.13}$$

The approximate problem can be solved through Lagrangian duality:

$$\theta_{k+1} = \theta_k + \sqrt{\frac{2\delta}{g^T H^{-1} g}} H^{-1} g \tag{6.14}$$

TRPO slightly modify the formula to satisfy the KL constraint as follows:

$$\theta_{k+1} = \theta_k + \alpha^j \sqrt{\frac{2\delta}{g^T H^{-1} g}} H^{-1} g \tag{6.15}$$

where $\alpha \in (0,1)$ is the backtracking coefficient and j is the smallest nonnegative integer such that $\pi_{\theta_{k+1}}$ satisfies the KL constraint.

6.2.3 Attention Augmented Reinforcement Learning Framework

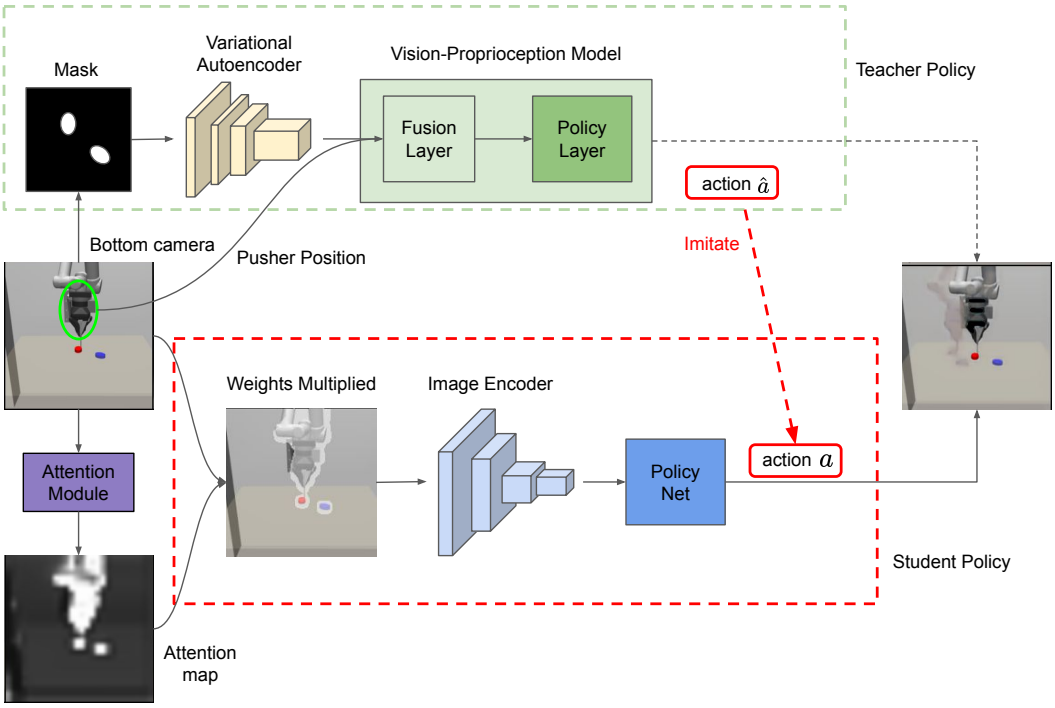


Figure 6.12: Overview of the attention augmented reinforcement learning framework.

The representation of the high-dimensional image and the policy network are trained at the same time. TRPO with convolutional networks as image encoder behaves poorly on the pushing task often gets stuck at locally optimum and learns very slowly (as shown in Figure 6.14). Two methods are proposed to accelerate the learning process:

- The attention module proposed in Section 6.1.2 is integrated into the RL framework to force the agent to focus only on the valuable information from the input; we assume this can help with the image representation learning.
- The trained teacher policy guides the student policy to an optimal solution, accelerating the learning process.

An overview of our framework is illustrated in Figure 6.12. Inspired by "privileged learning strategy" [71], we first train a teacher policy that has access to the robot pusher position and the objects' filtered mask from the bottom camera (the same model proposed in the previous chapter). This teacher policy is then used as the guide for the student policy, which does not rely on the information provided by the bottom camera.

The teacher policy computes a fusion embedding with the latent variables from VAE and the pusher position with the fusion layer. The policy layer in the vision-proprioception model takes the fusion embedding and outputs action $\hat{a}$. The training details of the teacher policy are also described in the previous chapter. After the teacher policy is trained, it is used to supervise a student policy, which takes only the visual input from the front camera. The optimization is performed through imitating the teacher

policy and maximum the expected reward at the same time. The imitation loss is defined as the mean squared error between the student policy action a and the action from the teacher policy $\hat{a}$. The attention maps from the attention module are multiplied as weights to the input image. The whole framework is illustrated in Figure 6.12.

6.2.4 State Space and Reward Specification

As is shown in Figure 6.12, the pushing task can be described the same as previous: pushing the object (red) from a random initial pose to a given target pose (blue), which is also a goal-conditioned RL problem. The difference is that the pushing model proposed in Chapter 4 requires the goal information provided explicitly (through the filterer object mask) by the environment. However, in this framework (Section 6.2.3), the goal information (presented as transparent blue object) is included in the visual observation, and the agent needs to learn the perception implicitly. A finite-horizon discounted MDP is constructed by (S, A, p, r), where state $s \in \mathbb{R}^{H \times W \times 3}$ is the visual observation (an RGB frame of height H and width W) from the front camera; $a = [a_x, a_y]$ is the continuous action, representing the pusher's 2-D velocity in the motion plane.

The reward is composed of 2 parts, namely the imitation reward and object-target distance reward:

$$r = -0.8 * \|\hat{a} - a\| - 1.5 * \|p_o - p_g\| \tag{6.16}$$

in which p_o and p_g are the object center positions of the pushing object and the goal.

6.3 Experiments

Comparison experiments are performed to verify the improvement of the attention mechanism and the teacher policy for the pushing task. The experiment is divided into three groups:

- The policy takes image observation x as input, trained without attention mechanism or teaching policy. The reward is only the negative distance between the current and goal positions: $-\|p_o - p_g\|$.
- The policy takes image observation x multiplied with attention map as input, which can be denoted as $x \cdot \Psi(\Phi(x))$. The reward function is also $-\|p_o - p_g\|$ without the guidance of teacher policy.
- The policy takes image observation multiplied with attention map as input, trained with the guidance of the pre-trained teacher policy. The reward function is specified in Formula 6.16.

The learning curve is shown in Figure 6.14: both the pre-trained teacher policy and the attention mechanism improve the pushing performance significantly. Taking the high-dimensional RGB image as input, TRPO can hardly learn a proper pushing policy without guidance. The high variance is because the policy always gets stuck in a

local optimum. Under the correct guidance (teacher policy), the student policy is more likely to jump out of the local optimum.

Through the comparison of the learning curve between the original image input (green line) and the input weighted by the attention map (orange line), the improvement of the attention module for visual reinforcement learning is also prominent, we assume this is because the pre-trained weights of each spatial location for the image input can help with the image representation learning.

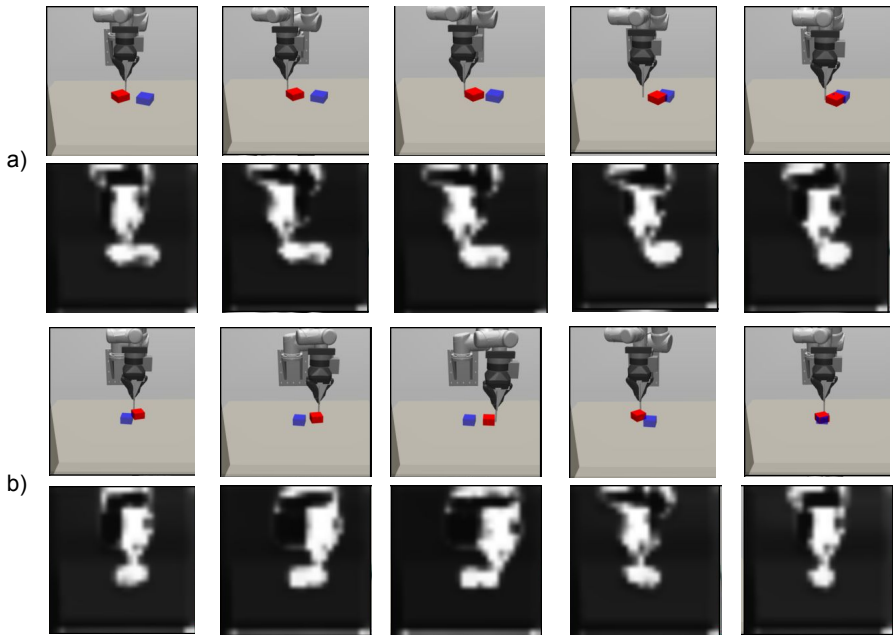


Figure 6.13: Examples of two pushing process. The first and second row show the image observation and the corresponding attention map respectively.

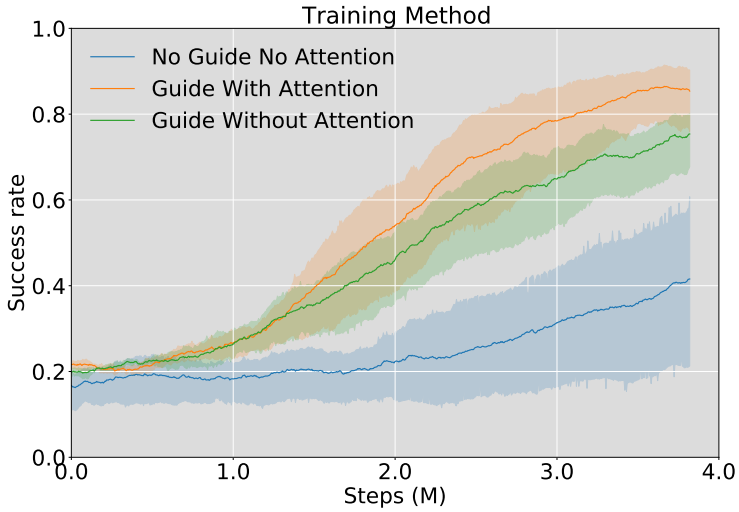


Figure 6.14: Learning performance comparison of different reinforcement learning frameworks.

6.4 Summary

This chapter presents a new top-down self-supervised model for finding the motion part from the noisy background. The output attention map is then used in the object pushing task by properly weighting all the spatial locations of the image input (highlighting the task-related part of the high-dimensional visual input), reducing the difficulty of the representation learning of the deep network. The training of the attention mechanism does not need any manual labeling, which is a promising method for other customized robotic tasks. Then we propose the attention augmented reinforcement learning framework, integrating the attention module and the pre-trained teacher policy. Through comparison experiments in simulation, the effectiveness of the algorithm is verified.

Chapter 7

Multimodal Reinforcement Learning for Multifingered Dexterous Grasping

In the previous chapters, three different RL-based methods were proposed for the object planar pushing task. This chapter continues to focus on the RL-based real robot control. A multimodal (including tactile, joint angles, and joint torques) RL framework is proposed to fuse the data from different sensors and apply motion control with the fusion representation. The model is evaluated with multifingered dexterous grasping task, generalizing over different shaped objects. Similar to the methods used in the previous three chapters, all the training processes are performed in simulation, and the trained model is directly applied on our Shadow Hand + UR10 robot platform (Figure 7.1).

Figure 7.1: UR10+Shadow hand platform.

7.1 Multimodal Representation Model for Grasping

How to design a robot controller which can combine modalities with different characteristics is always a challenging problem. Humans tend to solve a manipulation task through multiple sensing feedbacks, such as tactile, proprioception, vision, and even audition. The diverse set of modality data makes the feature representation and sensor fusion very challenging. Choosing proper modalities for a specific task is critical to the performance of the controller. Different combinations of modality representations are tested using the model proposed in Section 7.1.3.

7.1.1 Principal Component Analysis for Action Dimension Reduction

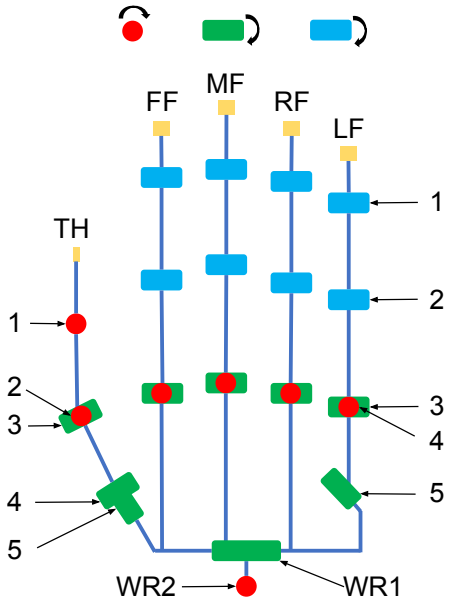


Figure 7.2: Joints mechanics of the Shadow hand. WR1, WR2, LF, RF, MF, FF, and TH refer to wrist joint1, wrist joint2, little finger, ring finger, middle finger, first finger, and thumb. Joint1 and joint2 marked as blue in each finger are coupled (these two joints are controlled by one motor).

The human hand is a high-dimensional, complex, and versatile end-effector system. Usually, a substantial computational resource in robotics applications is necessary to control a system like the human hand. However, according to the research from [121]: human beings control their hand in a subspace of much lower dimension than the original hand's DoF. Shadow Hand has 20 active actuators, among which 18 motors are used for 22 finger joints and the other two motors for two wrist joints (24 DoFs in total, illustrated in Figure 7.2).

An RL agent with 20 active actuators has a huge exploration space to search for the optimum policy. Training robot hand with RL in high-dimension action space means a huge exploration space, which usually causes low sample efficiency and local optimum

and generates weird hand poses under no constraints during training. PCA is used for grasping synergies definition [121] and action space reduction in this task. A Shadow Hand motion dataset is collected to calculate the eigenvectors and eigenvalues representing the correlated directions in joint space. We obtain 3000 motion samples by controlling the simulation hand by wearing the Cyberglove. Afterward, PCA is performed to reduce the action dimension, mapping from joint space to a subspace. The proper subspace dimension N and the corresponding performance are discussed in Section 7.2.1.

7.1.2 Proximal Policy Optimization

PPO tries to solve the classic reinforcement learning problem: how to make the biggest possible improvement step on a policy using the current data, without stepping so far, which may cause performance collapse? TRPO [124] (used in the previous) chapter deals with this problem with a complex second-order method, while PPO is a first-order method to keep new policies close to the old ones. From the implement perspective, PPO is much easier than TRPO but always comes with better performance than TRPO. PPO is an abbreviation of a family of algorithms, among which two are commonly used: PPO-Penalty and PPO-Clip. The former penalizes the KL divergence in the objective function, and the other designs a specialized clipping in the objective function to prevent the new policy from getting far away from the old policy by one step. PPO-Clip is used in the following framework (section 7.1.3).

Let π_θ denote a policy with trainable parameters θ. PPO-clip updates parameters with:

$$\theta_{k+1} = \arg\max_{\theta} \mathbb{E}_{s,a\sim\pi_{\theta_k}}\left[L(s,a,\theta_k,\theta)\right] \tag{7.1}$$

in which L is given by:

$$L(s,a,\theta_k,\theta) = \min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}A^{\pi_{\theta_k}}(s,a), clip\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, 1-\varepsilon, 1+\varepsilon\right)A^{\pi_{\theta_k}}(s,a)\right) \tag{7.2}$$

Usually in the implementation, taking multiple steps of Stochastic Gradient Descent (SGD) to maximize the objective. ε is the hyperparameter represents the abstract distance how far the new policy is allowed to be updated from the old policy by one step. The upper formula can be simplified as a version which can be easily implemented in code:

$$L(s,a,\theta_k,\theta) = \min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}A^{\pi_{\theta_k}}(s,a), g(\varepsilon, A^{\pi_{\theta_k}}(s,a))\right) \tag{7.3}$$

where

$$g(\varepsilon, A^{\theta_k}(s,a)) = \begin{cases}(1+\varepsilon)A & A \geq 0\\ (1-\varepsilon)A & A \leq 0\end{cases} \tag{7.4}$$

Suppose the advantage function $A^{\theta_k}(s,a)$ is positive, in this case, the objective function is simplified to:

$$L(s,a,\theta_k,\theta) = \min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, (1+\varepsilon)\right)A^{\pi_{\theta_k}}(s,a) \tag{7.5}$$

Suppose the advantage is negative, in this case, it is simplified to:

$$L(s,a,\theta_k,\theta) = \min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)},(1-\varepsilon)\right)A^{\pi_{\theta_k}}(s,a) \tag{7.6}$$

7.1.3 Reinforcement Learning Framework

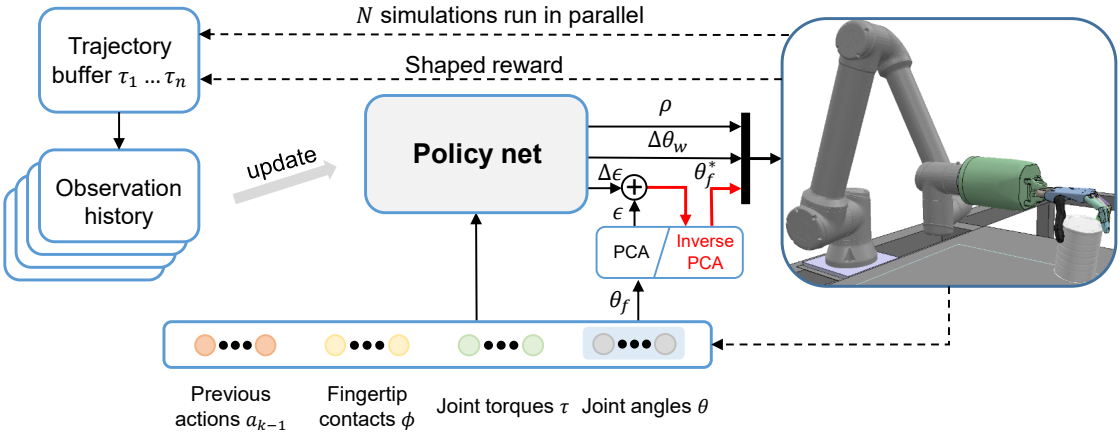


Figure 7.3: An overview of our multimodal reinforcement learning structure. Four different types of input information are captured from the environment and concatenated as one vector at each timestep, representing the agent's current state. This state vector is then stacked with several history states as the input and goes into the policy net. Three actions are generated from the policy net, controlling the lifting decision, wrist rotation, and finger joints separately. The observation history buffer keeps track of several previous state transitions. All trials are collected into the trajectory buffer as training data to update the policy.

A complete grasp could be defined from four aspects 1) the grasping point describing the tool center position, 2) the approach vector in which the robot hand approaches the grasping point, 3) the wrist orientation, and 4) an initial finger configuration [18]. To generate a grasp for Shadow hand, we need to control the finger and wrist joints during the RL agent training process. Instead of generating a fixed grasp candidate each time, we propose a two-stage dynamic grasping method where the robot continuously adjusts its motions until a lift-up decision. In the first stage, the hand tries to make a closing motion from the initial setup following a human-like hand closing trajectory until contact is detected between any of the five fingertips and the object. Afterward, the hand closing motion is stopped, and the robot goes into the second stage. This stage is a closed-loop control process, during which the hand obtains a set of observations from proprioception, binary tactile values of the fingertips, and finger joint torques. No visual perception or object model is provided in the simulation environment. The robot adjusts all the joint angles continuously to grasp the object better until it lifts the object. The overview of our multimodal reinforcement learning is illustrated in Figure 7.3.

The whole multifingered grasping process is modeled as a finite-horizon discounted MDP. At each timestep t, the agent perceives an observation $o_t \in O$ from the environment, executes an action $a_t \in A$ and obtains a reward $r_t \in R$. The agent executes an action

according to a stochastic policy $\pi(a_t|O_t)$, a distribution over actions conditioned on several recent observations. The goal is to find a policy π that maximizes the expected sum of discounted rewards over a finite trajectory T. The action value function is defined as:

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\tau(\pi)} \left[\sum_{t=0}^{T} \gamma^t r_t \right] \tag{7.7}$$

where $\gamma \in (0,1)$ is the discount factor, τ is the trajectory distribution under the policy π.

We apply PPO [125], an on-policy gradient algorithm to solve the specified policy optimization problem. Default network parameters are used in our work.

Observations Inspired by how humans use multimodal sensing to grasp objects, we also introduce tactile sensing, torques, and joint angles in the observation space of the Shadow hand agent. The agent's observations should match a real robot as much as possible to transfer the training model directly from the simulation to a real platform without any further training. As accurate contact force values and joint torque values are notoriously hard to get in simulation environments and these continuous values are difficult to map to a real robot, we use the binary contact information of the fingertip denoted as $\phi \in \{0,1\}$ and level-based joint torque denoted as $\tau \in \{0,...,5\}$ in the model to minimize the gap between simulation and real scene. Detailed mapping from raw values to the abstracted values is described in Section 7.2.2. The whole observation at $t = t_k$ is defined as $o_k = \langle a_{k-1}, \phi, \tau, \theta \rangle$, where a_{k-1} is the previous target action, τ are the joint torques, $\theta = \{\theta_f, \theta_w\}$ are finger joint and wrist joint angles of the hand. We find that the historical observations of several previous timesteps are meaningful during the training process. This may be because these initial observations can characterize the surface shape of the object, which helps with the exploring process of the agent. Finally, the input observation values at timestep t can be denoted as a vector $O_t = \langle o_{t-h}, ..., o_{t-1}, o_t \rangle$.

Actions The action generated from the policy comprises two continuous parts. 1) principal component value increments $\Delta\varepsilon$, representing the increments of the first n principal components ($n \leq 10$ in our model). 2) wrist joint angle increments $\Delta\theta_w$, and a discrete part 3) lifting decision ρ, which decides whether or not to lift the hand for a pick-up attempt. The combined action output from the policy is represented as $a_t = \{\Delta\varepsilon, \Delta\theta_w, \rho\}$.

We do action planning and optimization in the subspace in our method after applying PCA dimension reduction to the 22 finger joints. As is illustrated in Figure 7.3, the output action $\Delta\varepsilon$ from policy net is an increment based on the current principal components $\varepsilon = P(\theta_f)$, in which P is the function mapping from joint space to the subspace of planning. Target joint angles of the current timestep can be denoted as:

$$\theta_f^{target} = P^{-1}(\Delta\varepsilon + P(\theta_f)) \tag{7.8}$$

The initial palm pose may not always be a proper grasping pose when the objects move because of external interference, so we also consider wrist motion. The robot can adjust its wrist joint angles to a better palm pose like humans do while grasping objects. The Shadow hand has two wrist joints; $\Delta\theta_w$ is a 2-dimensional vector representing the rotation angles around the two wrist axes.

The robot's decision to lift its arm for a pick-up or not is also determined by policy output ρ. The robot arm keeps the original pose if $\rho = 0$ else lifts to a fixed height above the object. During lifting all hand joints stay invariant and the episode terminates after the lifting attempt. The log action probability can be denoted as:

$$\begin{aligned} \log \pi(a_t|O_t) = \log \pi(\rho|O_t) + \\ (1-\rho)\big[(\log \pi(\Delta\varepsilon|O_t) + \log \pi(\Delta\theta_w|O_t)\big] \end{aligned} \tag{7.9}$$

To ensure state-space exploration, the PPO learning algorithm used for our agent internally represents the policy as a set of Gaussian functions, whose mean μ and variance σ are updated during learning. The stochastic action output from the policy net $F(O_t)$ at timestep t is therefore a tuple of two Gaussian samples (namely the change of principal component values for the fingers, $\Delta\varepsilon_w \sim N(\mu_\varepsilon(O_t), \sigma_\varepsilon(O_t))$, and the update of hand wrist angles $\Delta\theta_w \sim N(\mu_\theta(O_t), \sigma_\theta(O_t)))$ and one discrete value taken from a Bernoulli distribution, $\rho \sim Bern(sigmoid(\beta_\rho(O_t)))$.

Reward Designing a reward function for a specific task to apply RL in robotic research is always necessary. In our multifingered grasping task, a training episode is terminated after the lifting attempt, and then a binary reward $r_b \in \{0,1\}$ represents whether the object picked up the object successfully is returned. The concrete reward function is defined as:

$$R = \begin{cases} r_b, & t = t_{final} \\ 0.03 r_c & t \in [1, t_{final} - 1] \end{cases} \tag{7.10}$$

The hand closing reward (r_c) is to guide the agent towards closing the hand. We assume that the combination of positive increments in several key joints:

$$J^* = \{FF3, MF3, RF3, LF3, TH5\} \tag{7.11}$$

will lead to a close-up hand motion. Therefore we denote a mask matrix $M = [m_1, m_2 ... m_n]$, $m_i \in \{0,1\}$ representing joint J_i in J^* or not.

$$r_c = \sum_{i \in joints} (\theta_f^{target} - \theta_f) \cdot M \tag{7.12}$$

Curriculum Learning The grasped object's initial horizontal position and rotation angle are randomized in a variable range related to the learning process p, namely curriculum learning. The aim is to choose environmental parameters that are neither too challenging nor too trivial for the agents [71]. During the experiment, we found that the initial pose of the object is essential for the hand to generate an effective grasp. In the beginning, we chose one fixed pose for every object to grasp in our simulation environment. As the training process goes on, we increase the task difficulty by adding a randomized disturbance to the initial pose. The new position pos_{start} and orientation rot_{start} of the object at the beginning of each episode change according to the grasping success rate r_s, original position pos_o, and rot_o.

$$\begin{cases} pos_{start} = pos_o + \delta_{pos} \quad \delta_{pos} \in [-\delta_p, +\delta_p] \\ rot_{start} = rot_o + \delta_{rot} \quad \delta_{rot} \in [-\delta_o, +\delta_o] \end{cases} \tag{7.13}$$

in which $\delta_p = 1.2(r_s + 0.2)$ and $\delta_o = 0.035 r_s$ are the variation range of orientation and position, respectively. δ_{pos} and δ_{rot} are both sampled from uniform distribution: $U(-\delta, \delta)$.

Figure 7.4: All the objects used for training in simulation, objects in the first row, second row and third row are selected from YCB objects [22], ShapeNet [24], and EGAD objects [87].

7.2 Experiments

This section evaluates the model's performance with different modalities and network structures in terms of grasping success rate. The agent is trained in simulation and transferred to a real robot platform (Shadow + UR10 in Figure 7.1). As there is always an apparent sim-to-real gap, a sensor mapping from simulation data to real robot data is necessary (Section 7.2.2). 24 different shaped objects (Figure 7.4) are used in the simulation environment to train and test the agent.

7.2.1 Performance Analysis of Different Modalities and Structures

The performance of the algorithm with different parameters is evaluated from 3 perspectives, which are input modalities (combination of fingertip tactile, joint torque, joint angle), network structures (MLP or GRU), and the dimension of subspace from PCA, (ranging from 3 to 10). We use the below pattern for naming the models, M-X where $X \in \{1,2,3\}$ means the different number of input modalities. PCA-N where $N \in \{3,5,8,10\}$ represents the dimension of subspace. All models are trained with three timesteps of history observations as input. The experiment results in Figure 7.5 show the grasp success rate of the above three experiments, respectively. Each curve is plotted using five individual runs. All models are trained for 1500 episodes.

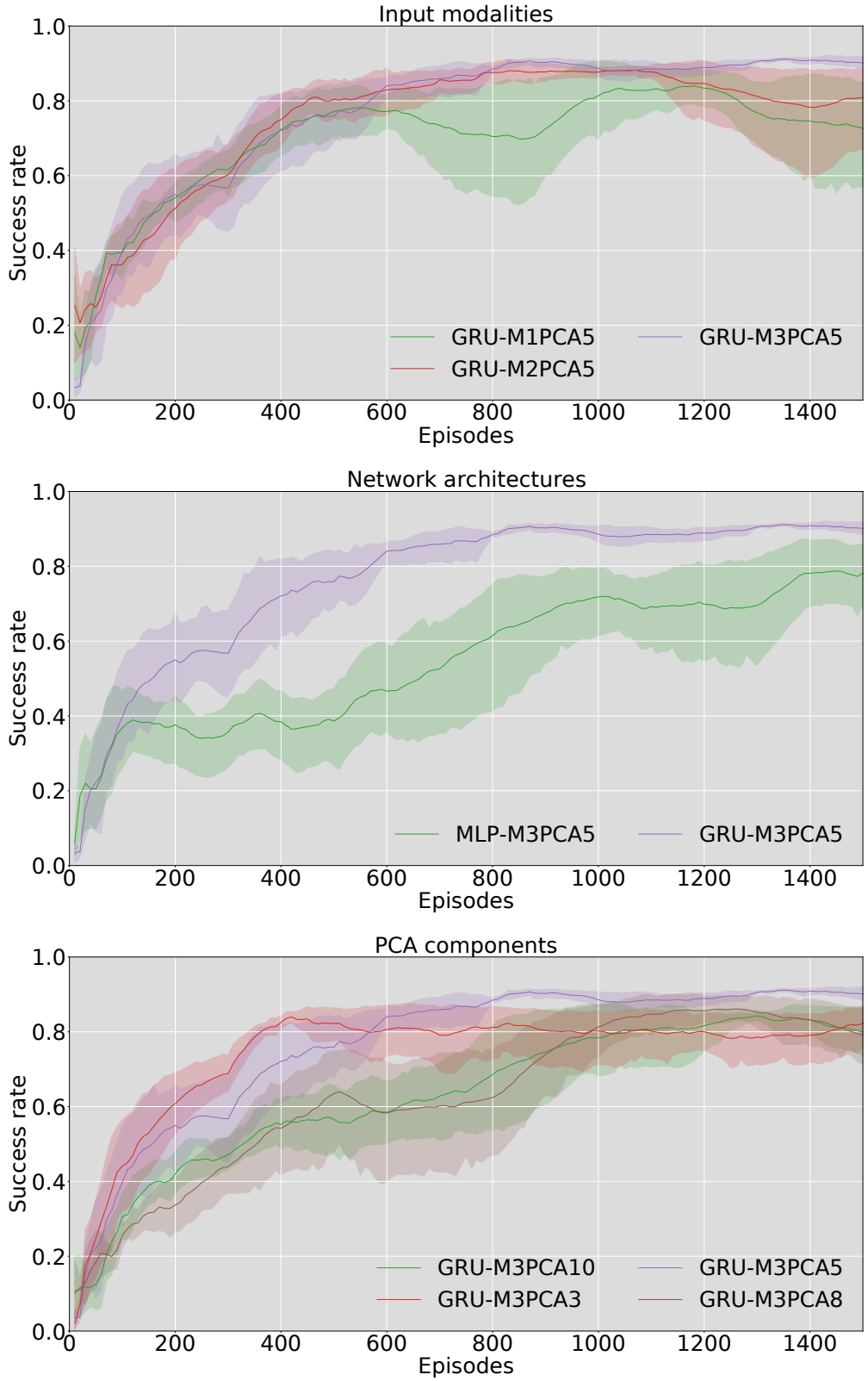


Figure 7.5: Network evaluation with different parameters. (top) Different input modalities. (middle) Different network architectures. (bottom) Different dimensional subspace after PCA.

The first factor we test is the input modality (Figure 7.5 top). M1, 2, 3 represents the input modality of joint angles; the fusion of joint angles and fingertip sensing; the fusion of joint angles, fingertip sensing, and joint torques. The learning performance of different input modalities is quite similar from the start. However, as the task difficulty is increasing gradually (see Section 7.1.3), the learning curve of the single modality (M1) shows volatility from the 600th episode. M2 begins to drop from around the 1200th episode, while M3 shows the best stability, verifying that the multimodal inputs can improve the model robustness.

The second experiment (Figure 7.5 middle) shows the comparison between models with different network structures. The model with recurrent architecture (GRU) outperforms the MLP model, and this can prove that the memory mechanism of GRU can learn a better representation from the interaction sequence.

In the third experiment (Figure 7.5 bottom), we test different dimensional action spaces. Because the finger motions are planned in the subspace of PCA, the latent space with a higher dimension can express more dexterous finger motions. Meanwhile, the action space for the agent to explore is also bigger, which usually increases the learning difficulty. Therefore, a properly dimensional reduction is important for grasping performance. From the figure, we can conclude that the best dimension for the subspace is five (the purple curve), three dimensional PCA (red curve) can learn faster but converges at a lower success rate. Higher-dimensional PCA (eight and ten) cannot improve the performance further.

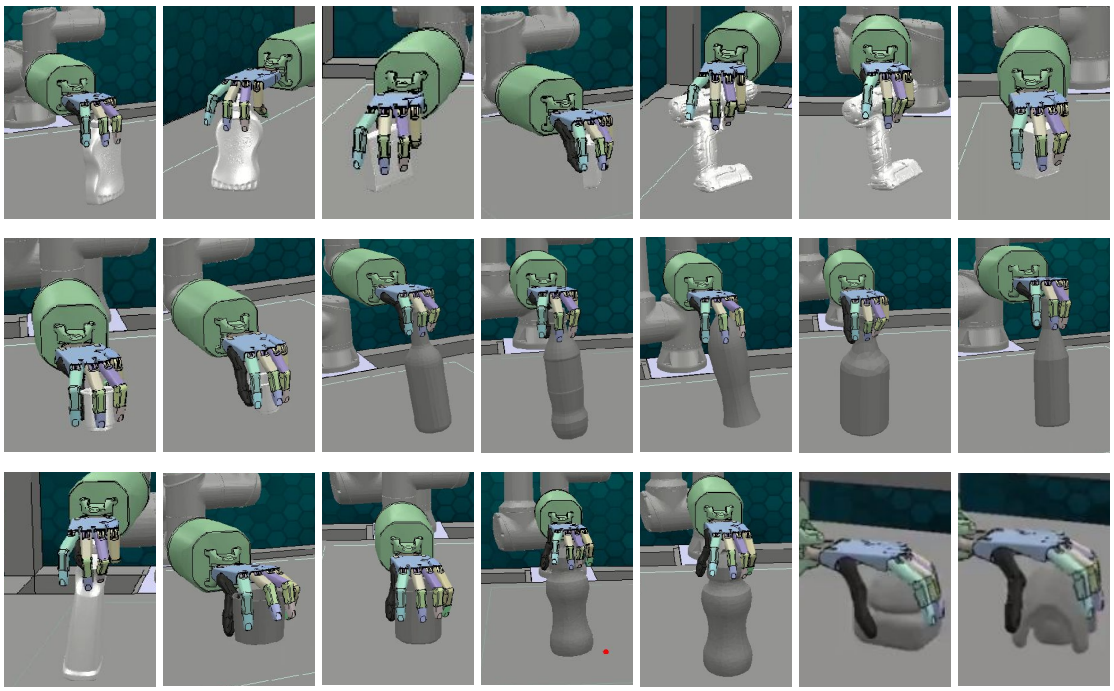


Figure 7.6: Grasping configurations in simulation.

7.2.2 Sensor Mapping

The sensor mapping system from real sensor data to the data reading in simulation is introduced in this section. For the tactile sensing on the fingertip, the binary data $0, 1$ is used to describe the contact status. The fingertip force sensor on the real Shadow Hand is a continuous pressure reading. The calibration experiment is performed to get the lower and upper raw data reading range by pressing each force sensor manually. The lower range of the tactile is defined by keeping the hand still, reading the sensor raw value ten times, and calculating an average value. During the grasping experiment, a tactile observation is considered 1 if the sensor reading is higher than 0.3% of the sensor range, otherwise 0. In this way, the continuous sensor reading is mapped into binary data, the same as the data format used in the simulator.

For the sensing of joint torques, the real Shadow Hand provides only the tendon force reading. The tendon reading is recorded when the hand is empty at the beginning of each grasp attempt and set as the initial tendon value $\tau_{j,0}$ for each joint. We use empirical thresholds $\in \{-200, -100, 0, 100, 200\}$ for the real robot and $\{-20, -10, 0, 10, 20\}$ for the simulation robot to map the reading $\tau_j - \tau_{j,0}$ to the interval $[0, 5]$, which is the default value range of the RL agent during training.

7.2.3 Real Robot Verification

15 different objects are tested on the real robot platform, including 9 training objects (number from 1-9 in Figure 7.7) and 6 unseen objects (number from 10-15 in Figure 7.8). The previous work from our group PointNetGPD [76] is used to generate the initial grasp pose. The robot setup is illustrated in Figure 7.1. A Kinect depth camera is used to receive object point cloud, which is taken as the input for the PointNetGPD. AprilTags on the table surface are used to locate the camera pose. At the beginning of each experiment, the grasping target is put at a fixed position on the table. Then the initial grasp pose is generated by PointNetGPD, after which the hand motion from the above three policies is performed. For each target, three different policies are tested for ten grasp attempts. The results are shown in Table 7.2 and 7.1. In baseline1, we set a torque limit for each joint and control the hand in position mode. All active joints are controlled to track the given trajectory until reaching the target positions or the tendon force limit. In baseline2, besides joint position and joint torque-sensing, we add one more modality: tactile sensing. The first finger, middle finger, and ring finger will stop closing if the tactile sensor on the fingertip is triggered, which helps prevent from over pushing the object beyond a proper grasping position. The model with the best grasping performance in simulation: GRU-M3-PCA5, and a comparison model GRU-M2-PCA are applied directly to the real robot.

As is shown in Table 7.1 and 7.2, baseline2 is slightly better than baseline1. Our algorithm still outperforms both the baselines, indicating that the agent learns to adjust hand motions according to the multimodal feedback as we proposed.

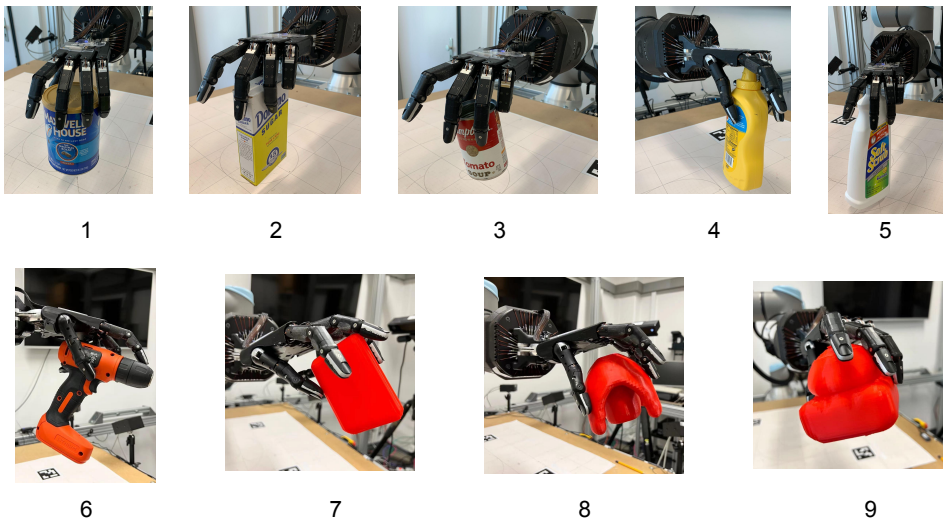


Figure 7.7: Grasping examples on training objects.

Table 7.1: Robot experiment result on training objects (from 1-9)

Object ID	1	2	3	4	5	6	7	8	9
Baseline1	80%	70%	40%	40%	20%	20%	50%	50%	80%
Baseline2	70%	80%	40%	40%	30%	40%	60%	80%	70%
GRU-M2-PCA5	60%	90%	**80%**	**60%**	**50%**	**80%**	60%	**70%**	90%
GRU-M3-PCA5	**100%**	**100%**	**80%**	50%	40%	**80%**	**70%**	**70%**	**100%**

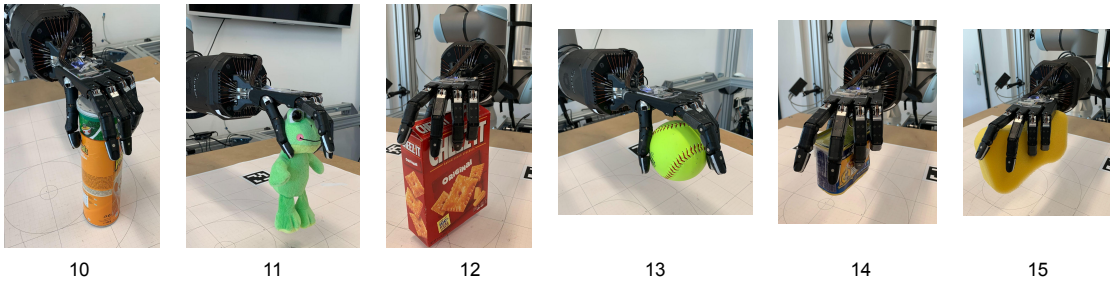


Figure 7.8: Grasping examples on novel objects.

Table 7.2: Robot experiment result on novel objects (from 10-15)

Object ID	10	11	12	13	14	15
Baseline1	60%	60%	**100%**	50%	70%	**100%**
Baseline2	70%	60%	**100%**	60%	80%	**100%**
GRU-M2-PCA5	**100%**	90%	**100%**	90%	**80%**	**100%**
GRU-M3-PCA5	**100%**	**100%**	**100%**	**100%**	70%	**100%**

7.3 Summary

In this chapter, we propose a multimodal RL algorithm to do multifingered dexterous grasping. First, a hand pose dataset is collected by humans grasping different objects while wearing the data glove. Afterward, we train a hand synergy using PCA, reducing the agent's action space dimension and generating human-like finger motions. Through experiments in simulation, we test different combinations of modalities as policy input, policy network structures (MLP and GRU), and the proper dimension of action space. The results show that the multimodal model with GRU as a policy network achieves the best grasping success rate. Finally, we successfully apply the model trained in simulation on the real robot through sensor mapping.

Chapter 8

Conclusions and Future Work

8.1 Summary

This thesis studies the application of RL in the scenario of object pushing and dexterous grasping, which are everyday tasks. The method or algorithm proposed in a later chapter always aims to solve the limitation or extend the generalization of the previous method. In conclusion, the research questions from Section 1.3 can be answered as follows:

- **Pushing Object Based on Dynamic Model**: A fusion of data-driven and analytical models for object pushing, which can perform online self-adapting, is proposed, the training data for the model is collected in the simulation. Domain adaption is adopted to bridge the sim-to-real gap. Recurrent Model Predictive Path Integral is developed to predict object motion for a few future steps according to recent interaction histories.

- **Visual Pushing with RL**: For an intelligent pushing side switch, RL is used to make the decision. Visual input is shown necessary for the generalization to manipulating a different object. Object mask is obtained through color filtering, then encoded into low-dimensional latent variables as the state for RL agent.

- **Attention Augmented RL framework**: Given the front camera as the only sensor, the understanding of the image, including the robot arm, the pushing object, and the goal pose, is more complicated. A self-supervised attention mechanism that can highlight the task-related region of an image is developed and integrated into the RL framework and proven to be effective in improving learning performance.

- **Multimodal RL for Dexterous Grasping**: A copy of our UR10-Shadow Hand platform is built in simulation to train grasping policy. The action planning is performed in a subspace of PCA. To verify the assumption that multimodal sensor data can improve the grasping success rate, comparison experiments of taking different modalities as input are performed to get the best multimodal combination.

The core contribution of this thesis is the deep study of RL around the problem of object pushing and grasping. The input of different modalities are tried through all chapters

(ground truth pose state in chapter 4, object mask and robot proprioception in chapter 5, raw images in chapter 6, multimodal sensory data including torques, joint angles, and tactile data in chapter 7). Accordingly, different RL algorithms are proposed to train the agent in simulation. Models are verified on real robot experiments.

8.2 Challenges

Despite the promising results presented in the previous chapters, some defects still exist in the methods.

One limitation of the dynamic motion model proposed in chapter 4 is that while pushing, the robot cannot effectively switch sides to control the object more precisely. However, benefiting from one of the advantages of model-based control, the learned model can be adapted to new tasks. The solution may be adding an upper-level controller in RMPPI. Another limitation is the generalization ability to off-centered objects during real experiments, which the model can learn by adding more off-centered objects in the simulation.

The vision-proprioception model proposed in chapter 5 has strict requirements for the experiment setup, limiting the application to other manipulation tasks. During real experiments, the pushing performance is influenced by lighting conditions such as illumination angle, the table's reflection, and light intensity because all the factors can affect the object mask filtering through a transparent table.

In the attention augmented RL framework, the attention mechanism needs to be pre-trained before the training start of the RL agent, and one specific dataset needs to be collected to train the attention module. Besides, further work is necessary to transfer the model trained in simulation to the real world.

One of the weaknesses of the multimodal grasping policy is the lack of reopening action. When trying to grasp a novel object, human beings can reopen their hands and adjust to a better grasping pose if the initial trying fails until reaching a comfortable grasping pose. Another limitation is that there is no visual information in the observation, which is a valuable input modality for robot manipulation.

8.3 Future Research

Based on the current results, some future researches can be planned. One direction is to transfer the attention augmented pushing model to the real world, in which some visual domain randomization methods can be used. A technical problem in this task is to render a virtual pushing target pose (a no-collision transparent object in simulation) in real-time on the camera observation during real experiments, which is easier to do in a simulator.

To transfer the attention augmented RL model to the real world is not trivial. Possible solutions include: adding the visual domain randomization into the training process in simulation, which may increase the training difficulty, or using CycleGAN [111] to transfer the real images into the simulation domain. Some initial work has already been in progress: as is shown in Figure 8.1, Contrastive Unpaired Translation (CUT) [97] is

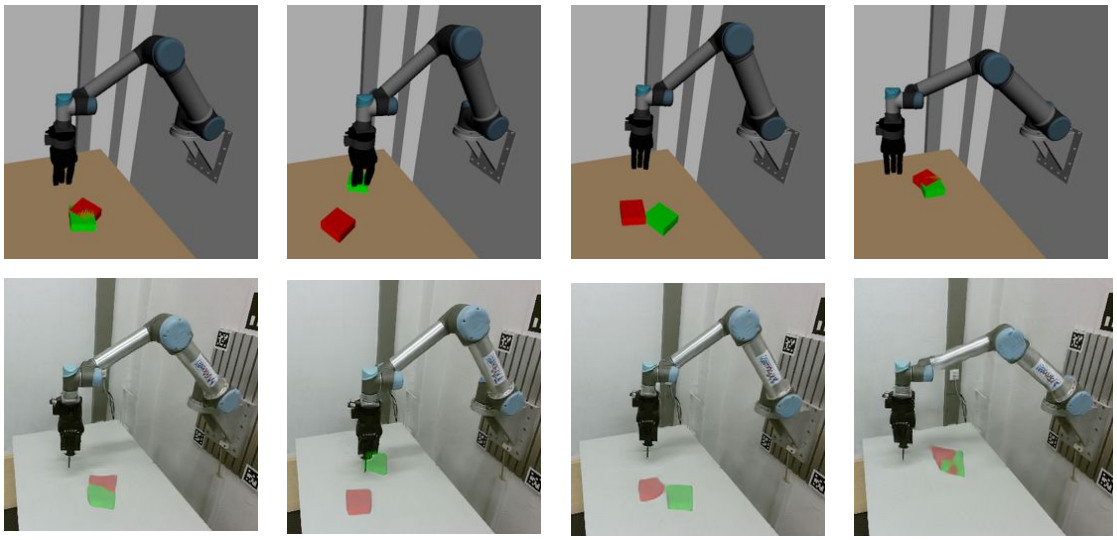


Figure 8.1: Examples of using CUT to transfer images from simulation domain (first row) to real world domain (second row).

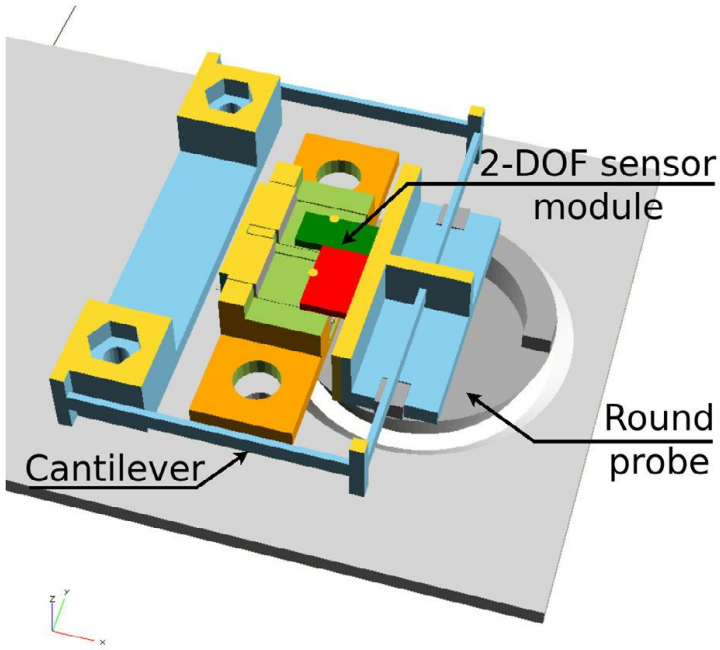


Figure 8.2: 2 DOF sliding friction force sensor for pushing experiments [47].

applied to transfer images from simulation domain to real domain. The images from the first line are taken as inputs to the translation model, and the second line shows the corresponding outputs. Current results show that some important details of the pushing targets (red and green boxes in the images), like the rotation and shape, are not correctly translated, which further research needs to improve.

Another direction to improve the pushing experiment is by adding more sensing

modalities. As is proved by the experiments in chapter 7, taking advantage of multimodal information from proper modalities in reinforcement learning can improve the final learning performance. Inspired by this finding, future research is to apply multimodal reinforcement learning in object planar pushing. The friction between the tabletop and the pushing target during motion is an essential factor that can represent the current interaction state and influence the optimal pushing action of the next step. One 2-DOF 3D printed sensor is designed [47] to measure friction forces during the pushing experiments, which can be easily mounted into different pushing targets.

Appendix A

List of Abbreviations

ADR Automatic Domain Randomization

CNN Convolutional Neural Network

DDPG Deep Deterministic Policy Gradient

DoF Degree of Freedom

DRL Deep Reinforcement Learning

GAN Generative Adversarial Network

GPR Gaussian Process Regression

GRU Gated Recurrent Units

HER Hindsight Experience Replay

IoU Intersection over Union

LSTM Long Short-Term Memory

MDP Markov Decision Process

MPC Model Predictive Control

MPPI Model Predictive Path Integral

MLP Multi-Layer Perception

NLP Natural Language Processing

PCA Principal Component Analysis

PPO Proximal Policy Optimization

RIG Reinforcement Learning with Imagined Goals

RL Reinforcement Learning

RMPPI Model Predictive Path Integral

RNN Recurrent Neural Network

ROI Region of Interest

SAC Soft Actor-Critic

TRPO Trust Region Policy Optimization

VAE Variational Autoencoder

Appendix B

Acknowledgements

Life is an endless practice. Every encounter has a different theme and brings us different feelings. Four years of doctoral career taught me the unity of knowledge and action, made me stay curious about the unknown, and hold awe to all the living things.

Thanks to my respectable supervisor Prof. Dr. Jianwei Zhang, I got the chance to pursue my doctorate at the University of Hamburg. Both Prof. Dr. Jianwei and his wife Xumei gave me kindly support and helped me overcome difficult times. I would also thank Prof. Dr. Janick Edinger for his effort to review this thesis.

As a member of the TAMS family, I feel at home surrounded by all the lovely colleagues. Dr. Norman provides insightful ideas for my research topic, guides me through tricky troubles during my experiments, and teaches me extended knowledge. Hongzhuo can always show up when I need professional technical support. I enjoyed our discussions and learned a lot from the collaboration. I also met Yunlei, who has professional industrial experience in robot development. We have a lot in common and often share insights. Michael is always the most lively and popular one among Chinese colleagues. Thanks to his friendly encouragement and brilliant idea, we had our first cooperation work published. Philipp, who acts as my technique idol, brings incredible results once in a while to us. At these moments, I always realize that geniuses do exist. Thanks to Dr. Andreas, my machine and our working station are always healthy and robust. I would also express my warmest thanks to our secretaries Lu, Wiebke, and Yihong for their selfless assistance. I am also grateful to have Dengyu Xiao, Shuang, Ge Gao, Jianzhi, Di Zhang, Dr. Chao Zeng, Yuyang, Wenkai, Chaobin, and also Niklas, Dr. Florens, German, Daniel, Marc, Yannick, Bernd as my companion, whose cheers and laughter embellish my life. Working with the TAMS family shall be one of the most fantastic experiences I will never forget.

Along the way, my parents Donsheng Cong and Zhirong Jiang, give me their best love and encouragement. I can always pluck up my courage to face setbacks and disappointments when I think of them. I am also lucky to meet my love Lixiang in the last year of my doctoral study. She makes me happier than I ever thought I could be, and I will spend the rest of my life trying to make her feel the same way.

The China Scholarship Council (CSC) is gratefully acknowledged for funding my study in Hamburg.

Bibliography

[1] Anurag Ajay, Jiajun Wu, Nima Fazeli, Maria Bauza, Leslie P Kaelbling, Joshua B Tenenbaum, and Alberto Rodriguez. Augmenting physical simulators with stochastic neural networks: Case study of planar pushing and bouncing. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3066–3073, 2018.

[2] Ankesh Anand, Evan Racah, Sherjil Ozair, Yoshua Bengio, Marc-Alexandre Côté, and R Devon Hjelm. Unsupervised state representation learning in atari. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, pages 8766–8779, 2019.

[3] Peter Anderson, Xiaodong He, Chris Buehler, Damien Teney, Mark Johnson, Stephen Gould, and Lei Zhang. Bottom-up and top-down attention for image captioning and visual question answering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6077–6086, 2018.

[4] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 5048–5058, 2017.

[5] OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Józefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, Jonas Schneider, Szymon Sidor, Josh Tobin, Peter Welinder, Lilian Weng, and Wojciech Zaremba. Solving Rubik's Cube with a Robot Hand. *arXiv:1910.07113v1*, 2019.

[6] OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Józefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, Jonas Schneider, Szymon Sidor, Josh Tobin, Peter Welinder, Lilian Weng, and Wojciech Zaremba. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research (IJRR)*, 39(1):3–20, 2020.

[7] Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C Lawrence Zitnick, and Devi Parikh. Vqa: Visual question answering. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 2425–2433, 2015.

[8] Rika Antonova, Silvia Cruciani, Christian Smith, and Danica Kragic. Reinforcement learning for pivoting task. *arXiv preprint arXiv:1703.00472*, 2017.

[9] Ermano Arruda, Michael J Mathew, Marek Kopicki, Michael Mistry, Morteza Azad, and Jeremy L Wyatt. Uncertainty averse pushing with model predictive path integral control. In *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, pages 497–502, 2017.

[10] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38, 2017.

[11] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *Proceedings of the International Conference on Learning Representations (ICLR), 2015*.

[12] Yujia Bao, Shiyu Chang, Mo Yu, and Regina Barzilay. Deriving machine attention from human rationales. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1903–1913, 2018.

[13] Maria Bauza, Ferran Alet, Yen-Chen Lin, Tomás Lozano-Pérez, Leslie P. Kaelbling, Phillip Isola, and Alberto Rodriguez. OmniPush: accurate, diverse, real-world dataset of pushing dynamics with RGB-D video. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4265–4272, 2019.

[14] Maria Bauzá, Francois Robert Hogan, and Alberto Rodriguez. A data-efficient approach to precise and controlled pushing. In *Proceedings of the Conference on Robot Learning (CoRL)*, pages 336–345, 2018.

[15] Maria Bauza and Alberto Rodriguez. A probabilistic data-driven model for planar pushing. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3008–3015, 2017.

[16] Irwan Bello, Barret Zoph, Ashish Vaswani, Jonathon Shlens, and Quoc V Le. Attention augmented convolutional networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 3286–3295, 2019.

[17] Christopher Berner, Greg Brockman, Brooke Chan, Christy Cheung, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

[18] Jeannette Bohg, Antonio Morales, Tamim Asfour, and Danica Kragic. Data-driven grasp synthesis—a survey. *IEEE Transactions on Robotics*, 30(2):289–309, 2013.

[19] Samarth Brahmbhatt, Ankur Handa, James Hays, and Dieter Fox. Contactgrasp: Functional multi-finger grasp synthesis from contact. In *Proceedings of the*

IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 2386–2393, 2019.

[20] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI gym. *arXiv:1606.01540*, 2016.

[21] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 1877–1901, 2020.

[22] Berk Calli, Arjun Singh, James Bruce, Aaron Walsman, Kurt Konolige, Siddhartha Srinivasa, Pieter Abbeel, and Aaron M Dollar. Yale-CMU-Berkeley dataset for robotic manipulation research. *The International Journal of Robotics Research (IJRR)*, 36(3):261–268, 2017.

[23] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *European Conference on Computer Vision (ECCV)*, pages 213–229, 2020.

[24] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. ShapeNet: An Information-Rich 3D Model Repository. Technical Report arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015.

[25] Christopher B Choy, JunYoung Gwak, Silvio Savarese, and Manmohan Chandraker. Universal correspondence network. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 29, pages 2414–2422, 2016.

[26] Matei Ciocarlie, Corey Goldfeder, and Peter Allen. Dimensionality reduction for hand-independent dexterous robotic grasping. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3270–3275, 2007.

[27] Lin Cong, Michael Görner, Philipp Ruppel, Hongzhuo Liang, Norman Hendrich, and Jianwei Zhang. Self-adapting recurrent models for object pushing from learning in simulation. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5304–5310, 2020.

[28] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. `http://pybullet.org`, 2016.

[29] N Faraji Davar, Teofilo de Campos, David Windridge, Josef Kittler, and William Christmas. Domain adaptation in the context of sport video action recognition. In *Domain Adaptation Workshop, in conjunction with NIPS*. University of Surrey, 2011.

[30] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[31] F. Ficuciello, A. Migliozzi, G. Laudante, P. Falco, and B. Siciliano. Vision-based grasp learning of an anthropomorphic hand-arm system in a synergy-based control framework. *Science Robotics*, 4(26), 2019.

[32] Fanny Ficuciello, Damiano Zaccara, and Bruno Siciliano. Synergy-based policy improvement with path integrals for anthropomorphic hands. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1940–1945. IEEE, 2016.

[33] Chelsea Finn, Xin Yu Tan, Yan Duan, Trevor Darrell, Sergey Levine, and Pieter Abbeel. Deep spatial autoencoders for visuomotor learning. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 512–519. IEEE, 2016.

[34] Peter R Florence, Lucas Manuelli, and Russ Tedrake. Dense object nets: Learning dense visual object descriptors by and for robotic manipulation. In *Proceedings of the Conference on Robot Leaning (CoRL)*, pages 373–385, 2018.

[35] Simone Frintrop. *VOCUS: A visual attention system for object detection and goal-directed search*. Springer, 2006.

[36] Simone Frintrop and Patric Jensfelt. Attentional landmarks and active gaze control for visual slam. *IEEE Transactions on Robotics*, 24(5):1054–1065, 2008.

[37] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning (ICML)*, pages 1587–1596, 2018.

[38] Jim Gao. Machine learning applications for data center optimization. In *Google White Paper*, 2014.

[39] Florian Golemo, Adrien Ali Taiga, Aaron Courville, and Pierre-Yves Oudeyer. Sim-to-real transfer with neural-augmented robot simulation. In *Proceedings of the Conference on Robot Learning (CoRL)*, pages 817–828, 2018.

[40] Suresh Goyal, Andy Ruina, and Jim Papadopoulos. Planar sliding with dry friction part 1. limit surface and moment function. *Wear*, 143(2):307–330, 1991.

[41] Abhishek Gupta, Coline Devin, YuXuan Liu, Pieter Abbeel, and Sergey Levine. Learning invariant feature spaces to transfer skills with reinforcement learning. *arXiv preprint arXiv:1703.02949*, 2017.

[42] David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 31, pages 2450–2462, 2018.

[43] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning (ICML)*, pages 1861–1870. PMLR, 2018.

[44] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9729–9738, 2020.

[45] Eric Heiden, David Millard, Erwin Coumans, Yizhou Sheng, and Gaurav S Sukhatme. Neuralsim: Augmenting differentiable simulators with neural networks. *arXiv preprint arXiv:2011.04217*, 2020.

[46] Olivier Henaff. Data-efficient image recognition with contrastive predictive coding. In *International Conference on Machine Learning (ICML)*, pages 4182–4192. PMLR, 2020.

[47] Norman Hendrich, Florens Wasserfall, and Jianwei Zhang. 3D printed low-cost force-torque sensors. *IEEE Access*, 8:140569–140585, 2020.

[48] Daniel Ho, Kanishka Rao, Zhuo Xu, Eric Jang, Mohi Khansari, and Yunfei Bai. Retinagan: An object-aware approach to sim-to-real transfer. *arXiv preprint arXiv:2011.03148*, 2020.

[49] Ronald A Howard. *Dynamic Programming and Markov Processes*. John Wiley, 1960.

[50] Han Hu, Zheng Zhang, Zhenda Xie, and Stephen Lin. Local relation networks for image recognition. pages 3464–3473, 2019.

[51] Junyan Hu, Hanlin Niu, Joaquin Carrasco, Barry Lennox, and Farshad Arvin. Voronoi-based multi-robot autonomous exploration in unknown environments via deep reinforcement learning. *IEEE Transactions on Vehicular Technology*, 69(12):14413–14423, 2020.

[52] Yuanming Hu, Luke Anderson, Tzu-Mao Li, Qi Sun, Nathan Carr, Jonathan Ragan-Kelley, and Frédo Durand. Difftaichi: Differentiable programming for physical simulation. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020.

[53] Drew A Hudson and Christopher D Manning. Compositional attention networks for machine reasoning. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.

[54] Jemin Hwangbo, Joonho Lee, Alexey Dosovitskiy, Dario Bellicoso, Vassilios Tsounis, Vladlen Koltun, and Marco Hutter. Learning agile and dynamic motor skills for legged robots. *Science Robotics*, 4(26), 2019.

[55] Nick Jakobi. Evolutionary robotics and the radical envelope-of-noise hypothesis. *Adaptive behavior*, 6(2):325–368, 1997.

[56] Stephen James, Paul Wohlhart, Mrinal Kalakrishnan, Dmitry Kalashnikov, Alex Irpan, Julian Ibarz, Sergey Levine, Raia Hadsell, and Konstantinos Bousmalis. Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12627–12637, 2019.

[57] Saumya Jetley, Nicholas A. Lord, Namhoon Lee, and Philip H. S. Torr. Learn to pay attention. In *Proceedings of the International Conference on Learning Representations (ICLR), 2018*.

[58] Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H. Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, Afroz Mohiuddin, Ryan Sepassi, George Tucker, and Henryk Michalewski. Model Based Reinforcement Learning for Atari. In *Proceedings of the International Conference on Learning Representations (ICLR), 2020*.

[59] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, et al. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *arXiv preprint arXiv:1806.10293*, 2018.

[60] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations (ICLR), 2015*.

[61] Diederik P. Kingma and Max Welling. Auto-Encoding Variational Bayes. In Yoshua Bengio and Yann LeCun, editors, *Proceedings of the International Conference on Learning Representations (ICLR), 2014*.

[62] Alina Kloss, Stefan Schaal, and Jeannette Bohg. Combining learned and analytical models for predicting action effects from sensory data. *The International Journal of Robotics Research (IJRR)*, 2020.

[63] Marek Kopicki, Sebastian Zurek, Rustam Stolkin, Thomas Moerwald, and Jeremy L Wyatt. Learning modular and transferable forward models of the motions of push manipulated objects. *Autonomous Robots*, 41(5):1061–1082, 2017.

[64] Tejas D Kulkarni, Ankush Gupta, Catalin Ionescu, Sebastian Borgeaud, Malcolm Reynolds, Andrew Zisserman, and Volodymyr Mnih. Unsupervised learning of object keypoints for perception and control. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, pages 10724—-10734, 2019.

[65] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, 1951.

[66] Visak Kumar, Tucker Hermans, Dieter Fox, Stan Birchfield, and Jonathan Tremblay. Contextual reinforcement learning of visuo-tactile multi-fingered grasping policies. In *NeurIPS Workshop on Robot Learning: Control and Interaction in the Real World*, 2019.

[67] Michael Laskin, Aravind Srinivas, and Pieter Abbeel. CURL: Contrastive unsupervised representations for reinforcement learning. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*, volume 119, pages 5639–5650, 2020.

[68] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[69] Alex X. Lee, Anusha Nagabandi, Pieter Abbeel, and Sergey Levine. Stochastic latent actor-critic: Deep reinforcement learning with a latent variable model. In *Neural Information Processing Systems (NeurIPS)*, pages 741–752, 2020.

[70] Jeongseok Lee, Michael X Grey, Sehoon Ha, Tobias Kunz, Sumit Jain, Yuting Ye, Siddhartha S Srinivasa, Mike Stilman, and C Karen Liu. Dart: Dynamic animation and robotics toolkit. *Journal of Open Source Software*, 3(22):500, 2018.

[71] Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning quadrupedal locomotion over challenging terrain. *Science robotics*, 5(47), 2020.

[72] Jiwei Li, Will Monroe, Alan Ritter, Dan Jurafsky, Michel Galley, and Jianfeng Gao. Deep reinforcement learning for dialogue generation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, (EMNLP), 2016*, pages 1192–1202.

[73] Yanghao Li, Naiyan Wang, Jianping Shi, Jiaying Liu, and Xiaodi Hou. Revisiting batch normalization for practical domain adaptation. In *Proceedings of the International Conference on Learning Representations (ICLR), 2017*.

[74] Yunzhu Li, Jiajun Wu, Russ Tedrake, Joshua B. Tenenbaum, and Antonio Torralba. Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids. In *Proceedings of the International Conference on Learning Representations (ICLR), 2019*.

[75] Zhongyu Li, Xuxin Cheng, Xue Bin Peng, Pieter Abbeel, Sergey Levine, Glen Berseth, and Koushil Sreenath. Reinforcement learning for robust parameterized locomotion control of bipedal robots. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–7, 2021.

[76] Hongzhuo Liang, Xiaojian Ma, Shuang Li, Michael Görner, Song Tang, Bin Fang, Fuchun Sun, and Jianwei Zhang. PointNetGPD: Detecting grasp configurations from point sets. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3629–3635, 2019.

[77] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *Proceedings of the International Conference on Learning Representations (ICLR), 2016*.

[78] Paulo Lissa, Conor Deane, Michael Schukat, Federico Seri, Marcus Keane, and Enda Barrett. Deep reinforcement learning for home energy management system control. *Energy and AI*, 3:100043, 2021.

[79] Weibo Liu, Zidong Wang, Xiaohui Liu, Nianyin Zeng, Yurong Liu, and Fuad E Alsaadi. A survey of deep neural network architectures and their applications. *Neurocomputing*, 234:11–26, 2017.

[80] Kendall Lowrey, Aravind Rajeswaran, Sham M. Kakade, Emanuel Todorov, and Igor Mordatch. Plan online, learn offline: Efficient learning and exploration via model-based control. In *Proceedings of the International Conference on Learning Representations (ICLR), 2019*.

[81] Jiasen Lu, Jianwei Yang, Dhruv Batra, and Devi Parikh. Hierarchical question-image co-attention for visual question answering. *Advances in Neural Information Processing Systems (NeurIPS)*, 29:289–297, 2016.

[82] Kevin M Lynch, Hitoshi Maekawa, and Kazuo Tanie. Manipulation and active sensing by pushing using tactile feedback. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 416–421, 1992.

[83] Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, and Ankur Handa. Isaac Gym: High Performance GPU-Based Physics Simulation For Robot Learning. *arXiv preprint arXiv:2108.10470*, 2021.

[84] Anthony Manchin, Ehsan Abbasnejad, and Anton van den Hengel. Reinforcement learning with attention that works: A self-supervised approach. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 223–230, 2019.

[85] Matthew T Mason. Mechanics and planning of manipulator pushing operations. *The International Journal of Robotics Research (IJRR)*, 5(3):53–71, 1986.

[86] Hamza Merzić, Miroslav Bogdanović, Daniel Kappler, Ludovic Righetti, and Jeannette Bohg. Leveraging contact forces for learning to grasp. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3615–3621, 2019.

[87] Douglas Morrison, Peter Corke, and Jürgen Leitner. Egad! an evolved grasping analysis dataset for diversity and reproducibility in robotic manipulation. *IEEE Robotics and Automation Letters*, 5(3):4368–4375, 2020.

[88] Alexander Mott, Daniel Zoran, Mike Chrzanowski, Daan Wierstra, and Danilo Jimenez Rezende. Towards interpretable reinforcement learning using attention augmented agents. In *Advances in Neural Information Processing Systems (NeurIPS), 2019*, pages 12329–12338.

[89] Jayraj Mulani, Sachin Heda, Kalpan Tumdi, Jitali Patel, Hitesh Chhinkaniwala, and Jigna Patel. Deep reinforcement learning based personalized health recommendations. In *Deep Learning Techniques for Biomedical and Health Informatics*, pages 231–255. 2020.

[90] Ashvin Nair, Shikhar Bahl, Alexander Khazatsky, Vitchyr Pong, Glen Berseth, and Sergey Levine. Contextual imagined goals for self-supervised robotic learning. In *Proceedings of the Conference on Robot Learning (CoRL)*, pages 530–539. PMLR, 2020.

[91] Ashvin V Nair, Vitchyr Pong, Murtaza Dalal, Shikhar Bahl, Steven Lin, and Sergey Levine. Visual reinforcement learning with imagined goals. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems (NeurIPS)*, volume 31, pages 9191–9200, 2018.

[92] Ozan Oktay, Jo Schlemper, Loic Le Folgoc, Matthew Lee, Mattias Heinrich, Kazunari Misawa, Kensaku Mori, Steven McDonagh, Nils Y Hammerla, Bernhard Kainz, et al. Attention u-net: Learning where to look for the pancreas. *arXiv preprint arXiv:1804.03999*, 2018.

[93] Edwin Olson. AprilTag: A robust and flexible visual fiducial system. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3400–3407, 2011.

[94] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

[95] OpenAI. [Blog] OpenAI Spinning Up. `https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html`, 2020.

[96] OpenAI. Robogym. `https://github.com/openai/robogym`, 2020.

[97] Taesung Park, Alexei A. Efros, Richard Zhang, and Jun-Yan Zhu. Contrastive learning for conditional image synthesis. In *European Conference on Computer Vision (ECCV)*, 2020.

[98] Vishal M Patel, Raghuraman Gopalan, Ruonan Li, and Rama Chellappa. Visual domain adaptation: A survey of recent advances. *IEEE signal processing magazine*, 32(3):53–69, 2015.

[99] Romain Paulus, Caiming Xiong, and Richard Socher. A deep reinforced model for abstractive summarization. In *Proceedings of the International Conference on Learning Representations (ICLR), 2018*.

[100] Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel van de Panne. Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Trans. Graph.*, 37(4):143:1–143:14, 2018.

[101] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–8, 2018.

[102] Xue Bin Peng, Erwin Coumans, Tingnan Zhang, Tsang-Wei Edward Lee, Jie Tan, and Sergey Levine. Learning agile robotic locomotion skills by imitating animals. In *Robotics: Science and Systems XVI, 2020*.

[103] Xue Bin Peng, Angjoo Kanazawa, Jitendra Malik, Pieter Abbeel, and Sergey Levine. Sfv: Reinforcement learning of physical skills from videos. *ACM Trans. Graph.*, 37(6), November 2018.

[104] Lerrel Pinto, Marcin Andrychowicz, Peter Welinder, Wojciech Zaremba, and Pieter Abbeel. Asymmetric actor critic for image-based robot learning. In *Robotics: Science and Systems XIV, 2018*.

[105] Matthias Plappert, Marcin Andrychowicz, Alex Ray, Bob McGrew, Bowen Baker, Glenn Powell, Jonas Schneider, Josh Tobin, Maciek Chociej, Peter Welinder, et al. Multi-goal reinforcement learning: Challenging robotics environments and request for research. *arXiv:1802.09464*, 2018.

[106] Mariya Popova, Olexandr Isayev, and Alexander Tropsha. Deep reinforcement learning for de novo drug design. *Science advances*, 4(7):eaap7885, 2018.

[107] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[108] Antonin Raffin, Ashley Hill, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, and Noah Dormann. Stable baselines3. `https://github.com/DLR-RM/stable-baselines3`, 2019.

[109] Tanzila Rahman, Shih-Han Chou, Leonid Sigal, and Giuseppe Carenini. An improved attention for visual question answering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pages 1653–1662, 2021.

[110] Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. In *Robotics: Science and Systems XIV, 2018*.

[111] Kanishka Rao, Chris Harris, Alex Irpan, Sergey Levine, Julian Ibarz, and Mohi Khansari. RL-CycleGAN: Reinforcement learning aware simulation-to-real. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11157–11166, 2020.

[112] Yongming Rao, Jiwen Lu, and Jie Zhou. Attention-aware deep reinforcement learning for video face recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3931–3940, 2017.

[113] Carl Edward Rasmussen. *Gaussian Processes in Machine Learning*. Springer, 2003.

[114] Maximo A. Roa, Max J. Argus, Daniel Leidner, Christoph Borst, and Gerd Hirzinger. Power grasp planning for anthropomorphic robot hands. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 563–569, 2012.

[115] Philipp Ruppel, Norman Hendrich, Sebastian Starke, and Jianwei Zhang. Cost functions to specify full-body motion and multi-goal manipulation tasks. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3152–3159, 2018.

[116] Andrei A Rusu, Matej Večerík, Thomas Rothörl, Nicolas Heess, Razvan Pascanu, and Raia Hadsell. Sim-to-real robot learning from pixels with progressive nets. In *Proceedings of the Conference on Robot Learning (CoRL)*, pages 262–270, 2017.

[117] Fereshteh Sadeghi, Alexander Toshev, Eric Jang, and Sergey Levine. Sim2real viewpoint invariant visual servoing by recurrent control. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4691–4699, 2018.

[118] Kate Saenko, Brian Kulis, Mario Fritz, and Trevor Darrell. Adapting visual category models to new domains. In *European Conference on Computer Vision (ECCV)*, pages 213–226, 2010.

[119] Sasha Salter, Dushyant Rao, Markus Wulfmeier, Raia Hadsell, and Ingmar Posner. Attention-Privileged Reinforcement Learning. In *Proceedings of the Conference on Robot Learning (CoRL)*, 2020.

[120] Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter Battaglia. Learning to simulate complex physics with graph networks. In *International Conference on Machine Learning (ICML)*, pages 8459–8468, 2020.

[121] Marco Santello, Martha Flanders, and John F Soechting. Postural hand synergies for tool use. *Journal of neuroscience*, 18(23):10105–10115, 1998.

[122] Ashutosh Saxena, Justin Driemeyer, and Andrew Y Ng. Robotic grasping of novel objects using vision. *The International Journal of Robotics Research (IJRR)*, 27(2):157–173, 2008.

[123] Tanner Schmidt, Richard Newcombe, and Dieter Fox. Self-supervised visual descriptor learning for dense correspondence. *IEEE Robotics and Automation Letters*, 2(2):420–427, 2016.

[124] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International Conference on Machine Learning (ICML)*, pages 1889–1897, 2015.

[125] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[126] Andrew W Senior, Richard Evans, John Jumper, James Kirkpatrick, Laurent Sifre, Tim Green, Chongli Qin, Augustin Žídek, Alexander WR Nelson, and Alex Bridgland. Improved protein structure prediction using potentials from deep learning. *Nature*, 577(7792):706–710, 2020.

[127] Pierre Sermanet, Corey Lynch, Yevgen Chebotar, Jasmine Hsu, Eric Jang, Stefan Schaal, Sergey Levine, and Google Brain. Time-contrastive networks: Self-supervised learning from video. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 1134–1141, 2018.

[128] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[129] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

[130] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2014.

[131] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. The MIT Press, 2018.

[132] Jie Tan, Tingnan Zhang, Erwin Coumans, Atil Iscen, Yunfei Bai, Danijar Hafner, Steven Bohez, and Vincent Vanhoucke. Sim-to-real: Learning agile locomotion for quadruped robots. In *Robotics: Science and Systems XIV*, 2018.

[133] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 23–30, 2017.

[134] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5026–5033, 2012.

[135] Eric Tzeng, Judy Hoffman, Ning Zhang, Kate Saenko, and Trevor Darrell. Deep domain confusion: Maximizing for domain invariance. *arXiv preprint arXiv:1412.3474*, 2014.

[136] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, pages 5998–6008, 2017.

[137] Pengfei Wang, Yu Fan, Long Xia, Wayne Xin Zhao, ShaoZhang Niu, and Jimmy Huang. Kerl: A knowledge-guided reinforcement learning model for sequential recommendation. In *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 209–218, 2020.

[138] Grady Williams, Paul Drews, Brian Goldfain, James M Rehg, and Evangelos A Theodorou. Aggressive Driving with Model Predictive Path Integral Control. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 1433–1440, 2016.

[139] Grady Williams, Nolan Wagener, Brian Goldfain, Paul Drews, James M Rehg, Byron Boots, and Evangelos A Theodorou. Information theoretic MPC for model-based reinforcement learning. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 1714–1721, 2017.

[140] Bohan Wu, Iretiayo Akinola, Jacob Varley, and Peter Allen. Mat: Multi-fingered adaptive tactile grasping via deep reinforcement learning. In *Proceedings of the Conference on Robot Learning (CoRL)*, 2019.

[141] Huijuan Xu and Kate Saenko. Ask, attend and answer: Exploring question-guided spatial attention for visual question answering. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 451–466, 2016.

[142] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International Conference on Machine Learning (ICML)*, pages 2048–2057, 2015.

[143] Zhuo Xu, Wenhao Yu, Alexander Herzog, Wenlong Lu, Chuyuan Fu, Masayoshi Tomizuka, Yunfei Bai, C Karen Liu, and Daniel Ho. Cocoi: Contact-aware online context inference for generalizable non-planar pushing. *arXiv preprint arXiv:2011.11270*, 2020.

[144] Xinchen Yan, Mohi Khansari, Jasmine Hsu, Yuanzheng Gong, Yunfei Bai, Sören Pirk, and Honglak Lee. Data-efficient learning for sim-to-real robotic grasping using deep point cloud prediction networks. *arXiv preprint arXiv:1906.08989*, 2019.

[145] Jun Yang, Rong Yan, and Alexander G Hauptmann. Cross-domain video concept detection using adaptive SVMs. In *Proceedings of the ACM international conference on Multimedia*, pages 188–197, 2007.

[146] Denis Yarats, Amy Zhang, Ilya Kostrikov, Brandon Amos, Joelle Pineau, and Rob Fergus. Improving sample efficiency in model-free reinforcement learning from images. In *Proceedings of the Conference on Artificial Intelligence (AAAI), 2021*, pages 10674–10681, 2021.

[147] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 3320–3328, 2014.

[148] Quanzeng You, Hailin Jin, Zhaowen Wang, Chen Fang, and Jiebo Luo. Image captioning with semantic attention. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4651–4659, 2016.

[149] Kuan-Ting Yu, Maria Bauza, Nima Fazeli, and Alberto Rodriguez. More than a Million Ways to Be Pushed. A High-Fidelity Experimental Dataset of Planar Pushing. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 30–37, 2016.

[150] Wenhao Yu, Jie Tan, C. Karen Liu, and Greg Turk. Preparing for the unknown: Learning a universal policy with online system identification. In *Robotics: Science and Systems XIII*, 2017.

[151] Zhou Yu, Jun Yu, Yuhao Cui, Dacheng Tao, and Qi Tian. Deep modular co-attention networks for visual question answering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6281–6290, 2019.

[152] Andrii Zadaianchuk, Maximilian Seitzer, and Georg Martius. Self-supervised visual reinforcement learning with object-centric representations. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

[153] Chi Zhang, Chetan Gupta, Ahmed Farahat, Kosta Ristovski, and Dipanjan Ghosh. Equipment health indicator learning using deep reinforcement learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 488–504, 2018.

[154] Jingwei Zhang, Lei Tai, Peng Yun, Yufeng Xiong, Ming Liu, Joschka Boedecker, and Wolfram Burgard. VR-Goggles for robots: Real-to-sim domain adaptation for visual control. *IEEE Robotics and Automation Letters*, 4(2):1148–1155, 2019.

[155] Xiangyu Zhao, Changsheng Gu, Haoshenglun Zhang, Xiaobing Liu, Xiwang Yang, and Jiliang Tang. Deep reinforcement learning for online advertising in recommender systems. *arXiv preprint arXiv:1909.03602*, 2019.

[156] Guanjie Zheng, Fuzheng Zhang, Zihan Zheng, Yang Xiang, Nicholas Jing Yuan, Xing Xie, and Zhenhui Li. DRN: A deep reinforcement learning framework for news recommendation. In *Proceedings of the 2018 World Wide Web Conference*, pages 167–176, 2018.

[157] Jiaji Zhou, Robert Paolini, J Andrew Bagnell, and Matthew T Mason. A convex polynomial force-motion model for planar sliding: Identification and application. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA))*, pages 372–377, 2016.

[158] Zhenpeng Zhou, Xiaocheng Li, and Richard N Zare. Optimizing chemical reactions with deep reinforcement learning. *ACS central science*, 3(12):1337–1344, 2017.

[159] Fengda Zhu, Linchao Zhu, and Yi Yang. Sim-real joint reinforcement transfer for 3D indoor navigation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11388–11397, 2019.

Erklärung der Urheberschaft

HHiermit versichere ich an Eides statt, dass ich die vorliegende Dissertation im Studiengang Informatik selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel - insbesondere keine im Quellenverzeichnis nicht benannten Internet-Quellen – benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht. Ich versichere weiterhin, dass ich die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht habe und die eingereichte schriftliche Fassung der auf dem elektronischen Speichermedium entspricht.

Ort, Datum Unterschrift

Erklärung zur Veröffentlichung

Ich stimme der Einstellung der Doctoral thesis in die Bibliothek des Fachbereichs Informatik zu.

Ort, Datum Unterschrift