Robotic Object Manipulation via Hierarchical and Affordance Learning

September 2023

Xintong Yang

in partial fulfilment of the requirements for the degree of

Doctor of Philosophy (Engineering)

School of Engineering

Cardiff University, United Kingdom

Abstract

With the rise of computation power and machine learning techniques, a shift of research interest is happening to roboticists. Against this background, this thesis seeks to develop or enhance learning-based grasping and manipulation systems.

This thesis first proposes a method, named A^2, to improve the sample efficiency of end-to-end deep reinforcement learning algorithms for long-horizon, multi-step and sparse reward manipulation. The named A^2 comes from the fact that it uses $\underline{\text{A}}$bstract demonstrations to guide the learning process and $\underline{\text{A}}$daptively adjusts exploration according to online performances. Experiments in a series of multi-step gridworld tasks and manipulation tasks demonstrate significant performance gains over baselines.

Then, this thesis develops a hierarchical reinforcement learning approach towards solving the long-horizon manipulation tasks. Specifically, the proposed *universal option framework* integrates the knowledge-sharing advantage of goal-conditioned reinforcement learning into hierarchical reinforcement learning. An analysis of the parallel training non-stationarity problem is also conducted, and the A^2 method is employed to address the issue. Experiments in a series of continuous multi-step, multi-outcome block stacking tasks demonstrate significant performance gains as well as reductions of memory and repeated computation over baselines.

Finally, this thesis studies the interplay between grasp generation and manipulation motion generation, arguing that selecting a good grasp before manipulation is essential for contact-rich manipulation tasks. A theory of *general affordances* based on the reinforcement learning paradigm is developed and used to represent the relationship between grasp generation and manipulation performances. This leads to the *general affordance-aware ma-*

nipulation framework, which selects task-agnostic grasps for downstream manipulation based on the predicted manipulation performances. Experiments on a series of contact-rich hook separation tasks prove the effectiveness of the proposed framework and showcase significant performance gains by filtering away unsatisfactory grasps.

Acknowledgement

I would like to express my acknowledgement to those who have loved, supported and helped me in the four years of my PhD career.

I deeply thank my supervisor Dr. Ze Ji for accepting me as one of his first PhD students and giving me enormous support in the development of engineering, academics, teaching and research resources. My thanks also go to my co-supervisors Dr. Wu Jing and Dr. Yukun Lai who have given me valuable advice on research, programming and academic writing. I express my thanks to Dr Ting Zhang who introduced me to my PhD supervisor.

I came to the UK by myself in September 2019, leaving the place where I had lived for 25 years. I could not have possibly reached even the beginning of my PhD without the love and support of my parents. My deepest thanks go to my loving father and mother, who have sacrificed themselves, put their trust in me and let me pursue my own future.

During the most stressful and lonely days of my life, at times when I could not even notice the pressure and pain in my heart, it was my dear wife who stayed with me, guided me, protected me, helped me and encouraged me. My heart belongs to my dear wife, who has walked by me and witnessed my successes and failures with her own eyes.

I am grateful for the companionship and help provided by my colleagues in the Cardiff RoPal group. It has been good for me to work alongside them.

I also deeply appreciate the love, help and encouragement that I received from my dear friends in Cardiff. I am grateful for the practical and spiritual gifts they have generously given to me.

Finally, I express my thanks to the China Scholarship Council which financially supports my tuition fee and living costs during my study at Cardiff University.

Contents

List of Figures

List of Tables

List of Acronyms

DL	Deep learning
RL	Reinforcement learning
DRL	Deep reinforcement learning
DNN	Deep neural network
MLP	Multi-layer perceptron
DQN	Deep Q network
DDPG	Deep deterministic policy optimisation
SAC	Soft actor critic
HRL	Hierarchical reinforcement learning
UOF	Universal option framework
GRL	Goal-conditioned reinforcement learning
GAM	General affordance-aware manipulation
GPU	Graphics processing unit
TAG	Task-agnostic grasp
TOG	Task-oriented grasp
CEM	Cross entropy method
Dof	Degree of freedom
PID	Proportional–integral–derivative
IL	Imitation learning
RRT	Rapid-exploring random tree
PRM	Probabilistic roadmap
AI	Artificial intelligence
TAMP	Task and motion planning
GUI	Graphic user interface

URDF	Universal Robot Description Format
ROS	robot operating system
MDP	Markov decision process
MC	Monte Carlo
TD	Temporal difference
AC	Actor critic
MaxEnt	Maximum entropy
TD3	Twin delayed deep deterministic policy gradient
HER	Hindsight experience replay
PMG	Pybullet Multigoal
AD	Abstract demonstrations
ADAE	Abstract demonstrations & adaptive exploration
SMDP	Semi Markov decision process
IOL	Intra-option learning
DRAL	Deep robotic affordance learning
MAGF	Manipulation affordance-based grasp filter
GAP	General action-consequence prediction
CM	Cartesian movement
HM	Hemisphere movement
SLM	Straight-up lifting motion

List of Notations

Index (global meanings unless redefined in the relevant texts):

t	index of discrete timestep
n	index of samples drawn from a dataset of reply buffer
i, j	index of optimisation iteration

Operation:

$\sum_{t=0}^{T}[f(x_t)]$	summation of $f(x_t)$ from $t = 0$ to $t = T$
$\sum_{x}[f(x)]$	summation of $f(x)$ over the set of x
$\mathbb{E}_p[X]$	expectation of X over the distribution p
$\mathbb{E}_{x \sim p(x)}[f(x)]$	expectation of $f(x)$ estimated by samples of x drawn from $p(x)$
$\max_x f(x)$	maximise f over the set of x
$\arg\max_x f(x)$	retrive the x that maximises f
$\nabla f(\boldsymbol{x})$	compute the gradient of $\boldsymbol{x}$ w.r.t. f when $\boldsymbol{x}$ is the only optimisable parameter
$\nabla_{\boldsymbol{x}} f(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$	compute the gradient of $\boldsymbol{x}$ w.r.t. f when multiple optimisable parameter occur
$\int_{\mathcal{X}}[f(x)]$	integral of f over $x \in \mathcal{X}$
$x \sim p(x)$	stachastically sample x from distribution $p(x)$
$exp(x)$	exponentiation of x
$\odot$	element-wise (Hadamard) product
$\|\|$	vector concatenation

Appear first in section 3.1:

$\mathcal{S}$	set of all states
$\mathcal{A}$	set of all actions
p	dynamic transition distribution
p_0	initial state distribution
$r, r_t, r_n, r(s_t, a_t)$	rewards (function)
γ	discount factor
$\mathbb{R}$	space of real number
s, s_t, s_n	states
s', s_{t+1}, s'_n	next states
a, a_t, a_n	actions
G, G_t	return (cumulated rewards)
T	the length of an episodic trajectory
$\pi(s), \pi(a\|s)$	policy as deterministic mapping, stochastic distribution
$v^\pi, v^\pi(s)$	state value function w.r.t π
$q^\pi, q^\pi(s, a)$	q value function w.r.t π
π_*, v_*, q_*	optimal policy and values
S_t, A_t, R_t	sampled state, action and reward
$\mathbb{E}_\pi$	expectation over the policy distribution
$J(\pi)$	cost function for the policy
$\hat{Q}, \hat{q}$	the target q value
α	learning rate or update ratio
δ_t	temporal-different (TD) error
$G_t^{(n)}$	n-step return from time t
$d(s)$	the function that weights the importance of value error
$\boldsymbol{w}, \boldsymbol{\theta}$	set of optimisable parameters

$\tilde{q}_{\boldsymbol{w}}, \tilde{v}_{\boldsymbol{w}}$	the approximate values w.r.t. parameter $\boldsymbol{w}$
$\pi_{\boldsymbol{\theta}}$	the policy distribution w.r.t. parameter $\boldsymbol{\theta}$
$d^{\pi}(s)$	the state distribution induced by the policy
$adv^{\pi}(s,a)$	the advantage function
$Pr(s_0 \to s, k, \pi)$	the probability of transitioning to s from s_0 in k steps following the policy π
$\beta_{\boldsymbol{\theta}}(a\|s)$	a behavioural policy for exploration under the off-policy setting
ρ_t	the importance sampling ratio

Appear first in section 3.2:

$\mathcal{D}$	replay buffer
ξ, ξ_n	a transition, or the n-th transition from a mini-batch of size N
$J(\boldsymbol{w}), J(\boldsymbol{\theta})$	cost function for the parameter
$\mathbb{E}_{\xi_n \sim \mathcal{D}}$	expectation over the distribution of transitions sampled from the replay buffer
α_w, α_θ	learning rate or update ratio
$\boldsymbol{w}^-, \boldsymbol{\theta}^-$	(delayed) copy of the parameters
$\mathcal{H}_\pi$	entropy of π
$\bar{\mathcal{H}}$	target entropy
$\alpha_{\mathcal{H}}$	the temperature parameter for soft-actor critic update
D_{KL}	Kullback-Leibler distance
$\mathbb{E}_{a \sim \pi_{\boldsymbol{\theta}}}$	expectation over the online policy distribution
$Z_{\boldsymbol{w}}$	partition function
$\mathcal{N}(a,b)$	Gaussian distribution of mean a and variance b

Appear first in chapter 4:

g	goal
$\mathcal{G}$	set of goals
$m(s)$	a representation mapping from state space to goal space
$\mathbf{x}$	Cartesian poses
g^+, g^-	desired goal and undesired goals
$\mathcal{G}^+, \mathcal{G}^-$	set of desired goals and undesired goals
δ_d	distance threshold
$\bar{g}^+$	substitutional desired goal
$\mathcal{U}\{a, b\}$	uniform distribution over the integers between a and b included
$\mathcal{U}\{x \vert x \in \mathcal{X}\}$	unifrom distribution over the set of x
$\{\tau_j\}$	sequence of trajectories
s^+_{end}	the target state of a task
$p\{s^+_{end} \vert s_0, \tau_0, \tau_1, ...\}$	the probability of ending at the target state following $\{\tau_j\}$ from s_0
$\{x^*_n\}$	a sequence of integers indicated the order of subtasks
η	demonstration proportion
$\boldsymbol{S}$	vector of task performance metrics for a number subtasks
$\overline{S}, \overline{\boldsymbol{S}}$	Polyak average of the task performances
τ_S	Polyak average update ratio
$\epsilon_{dqn}, \boldsymbol{\epsilon}_{dqn}$	parameter of the ϵ-greedy exploration strategy
$\epsilon_{ddpg}, \sigma_{ddpg}$	parameter of the $\epsilon - Gaussion$ exploration strategy
$\boldsymbol{\sigma}_{sac}$	deviations of the SAC policy distribution

Appear first in chapter 5:

o	an option
$\mathcal{O}$	set of options
π^o	the policy that option o executes
$\mathcal{I}^o$	set of states where o can be activated
β^o, b^o	stochastic and deterministic termination condition
$\tilde{u}(s, o)$	state-option value upon arrival
π^L, π^H	low-level and high-level polcy
o_g	universal option
π_g^L, π_g^H	universal low-level and high-level policy
$\mathcal{I}_g^L$	set of states where o_g can be activated
β_g, b_g	stochastic and deterministic goal-augmented termination condition
$\mathcal{A}^L, \mathcal{A}^H$	set of low-level and high-level actions
$\mathcal{G}^L, \mathcal{G}^H$	set of low-level and high-level goals
a^L, a^H	low-level and high-level action
g^L, g^H	low-level and high-level goal
$m^L(s), m^H(s)$	representation mappings from state space to low-level and high-level goal spaces
$\mathbf{x}, \mathbf{v}, \mathbf{w}$	pose, linear velocity, angular velocity vectors
$\mathbb{1}\left[c(\cdot)\right]$	an indicator function that gives 1 when the condition $c(\cdot)$ is satisfied and 0 otherwise
ξ_n^H	the n-th high-level transition of a mini-batch of size N
$b_{n,g_{a^H}^{L+}}$	the termination condition for the desired low-level goal selected by action a^H in the n-th high-level transition

Appear first in chapter 6:

$I_a, I_a(s), I_a^p$	the (dynamical) intents of an action a, in state s
$\mathcal{S}_a^+$	set of states that are desired to be the consequential states of the action a
$\mathcal{I}, \mathcal{I}_a^p$	set of (dynamical) intents
$\mathcal{AF}_\mathcal{I}, \mathcal{AF}_\mathcal{I}^p$	the affordances (set of state-action pairs) related to the (dynamical) intents
$p^{\mathcal{AF}_\mathcal{I}}(s,a)$	the probability of a state-action pair being within the affordance set
$y, y(s,a)$	the measurement of the consequence of taking an action at a state
$\mathcal{Y}$	set of measurement values
$I_{a,\pi}^y, I_{a,\pi}^y(s)$	general intent
$y^{\pi+}(s,a)$	the measurement of the consequence of taking an action at a state and following the policy π thereafter
$\mathcal{Y}_{a,\pi}^+$	set of measurement values of taking an action and following the policy π thereafter
$\hat{\pi}$	a baseline policy
$\mathcal{AF}_{\mathcal{I},\pi}^y$	the general affordances (set of state-action pairs) related to the general intents for a policy π based on the action consequences measured by y
$\Phi, \Phi(a^H)$	a mapping from high-level actions to low-level states
$\tilde{y}_{i,\boldsymbol{\vartheta}}^{\pi^L}$	the GAP model, the estimated action consequences for π^L based on the i-th measurement approximated by a set of weights $\boldsymbol{\vartheta}$

Publications

[1] Yang, X., et al., 2021. Hierarchical reinforcement learning with universal policies for multistep robotic manipulation. IEEE Transactions on Neural Networks and Learning Systems.

[2] Yang, X., et al., 2021, September. An open-source multi-goal reinforcement learning environment for robotic manipulation with pybullet. In Annual Conference Towards Autonomous Robotic Systems (pp. 14-24). Springer, Cham.

[3] Yang, X., et al., 2022, September. Abstract demonstrations and adaptive exploration for efficient and stable multi-step sparse reward reinforcement learning. In 2022 27th International Conference on Automation and Computing (ICAC). IEEE.

[4] Yang, X., et al., 2023. Recent Advances of Deep Robotic Affordance Learning (DRAL): A short review. IEEE Transactions on Cognitive and Developmental Systems (TCDS).

[5] (Under review) Yang, X., et al., 2023. GAM: General Affordance-based Manipulation for Contact-rich Object Disentangling Tasks.

Chapter 1

Introduction

1.1 Background

Humans are capable of manipulating objects to serve their day-to-day needs, such as the tasks shown in Figure 1.1. On one hand, humans possess the adaptivity, flexibility and dexterity of finger and hand motion control mechanisms shown by playing a joystick or knitting a sweater. On the other hand, in higher-level cognitive processes, humans also have the perception, understanding and reasoning abilities for the planning processes for tasks such as object rearranging and disentangling. Reproducing these manipulation abilities on robots is one of the long-term aims of the robotic community (Mason, 2018; Billard and Kragic, 2019). As suggested by many researchers, classic and modern learning-based approaches have achieved stable grasping and precise manipulation in short-horizon tasks with rich feedback signals (Mason, 2018; Billard and Kragic, 2019; Du *et al.*, 2021).

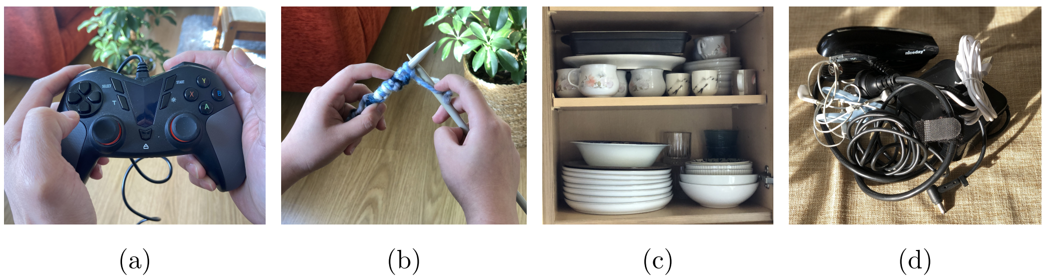

(a) (b) (c) (d)

Figure 1.1: Human manipulation examples. (a): Playing a joystick. (b): Knitting. (c) Rearranging tableware. (d) Cable disentangling.

However, one of the bottlenecks for robot manipulation is the low sample efficiency problem in the face of long-horizon manipulation, sparse task feedback and complex interplays between subtasks (Mason, 2018; Billard and Kragic, 2019; Kroemer *et al.*, 2021; Liu *et al.*, 2021; Newbury *et al.*, 2022). Such tasks have important applications in the real world. For example, building a house requires a long trajectory of arm, hand and finger motions in continuous spaces, yet what is normally given as task guidance only includes descriptive subgoals/subtasks. In addition, the dependencies among subtasks/steps tend to exacerbate the exploration or searching difficulty. One cannot build the walls without first building up a base. Moreover, planning a pose to grasp the object needs to consider what the object will be used for. Assembling and disassembling a piece of furniture requires different grasping poses and different organisation of different motion skills. Such problems occur in many scenarios, such as automated construction, assembly/disassembly tasks in the industry, cooking, object rearrangement, and furniture or toy assembly tasks in home environments, etc.

Evidently, existing grasping and manipulation algorithms fail in such long-horizon and multi-step tasks due to low sampling or searching efficiency (Mason, 2018; Billard and Kragic, 2019; Kroemer *et al.*, 2021; Liu *et al.*, 2021; Newbury *et al.*, 2022). On one hand, non-learning solutions that require a dynamic model, which is difficult to design, are much less suitable for solving the type of long-horizon manipulation tasks of our concerns (Caldera *et al.*, 2018; Fang *et al.*, 2019a; Ravichandar *et al.*, 2020). On the other hand, learning-based approaches suffer from low sample efficiency and thus struggle to learn and reason about the long-term relationships among dependent subtasks (Nair *et al.*, 2018; Zhu *et al.*, 2022; Shah *et al.*, 2022).

1.2 Aim and Ojbectives

Motivated by this research gap, the overall aim of this thesis is to improve the learning efficiency and performances of such long-horizon and multi-step manipulation tasks. More specifically, the following challenges regarding such tasks will need to be addressed:

- **Long task horizon**: This implies that the task is so complex that it could be decomposed into a number of subtasks that require different skills. For example, stacking a number of blocks into different orders or assembling/disassembling a piece of furniture.
- **Subtask dependency**: This implies that the success of some subtasks depends on the success of other subtasks. For example, placing a block requires first grasping it, or separating an entangled object has different performances when the object is grasped at different locations.
- **Delayed and sparse task feedback**: This implies that the feedback signals required to induce the desired motion skills or plans are sparse and tend to appear at the very late stage of a task. For example, only when the last piece of an assembly task is placed correctly can positive task completion feedback be given, because there are countless combinations of possible trajectories that can achieve the task and it is difficult to design a dense feedback function that induces the optimal behaviour.

Therefore, the following objectives are to be pursued in this thesis:

1. Investigate the performances of end-to-end reinforcement learning (RL) in long-horizon and multi-step manipulation tasks with sparse and delayed task feedback.
2. Develop simulation software for safer and faster manipulation learning.
3. Develop algorithms to accelerate end-to-end RL methods for such manipulation tasks.

4. Investigate the performances of learning multiple outcomes in long-horizon manipulation tasks via parallelly trained hierarchical reinforcement learning (HRL).
5. Develop an HRL framework with only one policy to learn multiple goals at each decision level to reduce memory consumption and data collection costs.
6. Identify the root cause of the non-stationarity problem that occurred in the parallel training process.
7. Propose a solution to stabilise the non-stationary parallel training process.
8. Review the progress of affordance-based solutions for such long-horizon manipulation tasks.
9. Investigate the performance of modern task-agnostic grasp (TAG) generation methods in terms of the downstream manipulation performances.
10. Develop an affordance-based manipulation framework where grasp actions can be selected based on the desired downstream manipulation performances.

1.3 Contributions

In accordance with the three research objectives, this section summarises the contributions claimed in this thesis.

By pursuing objectives 1 - 3, this thesis makes the following contributions:

- Develop an open-source simulation software for long-horizon and multi-step manipulation tasks.
- Develop A^2, which uses abstract demonstrations and adaptive exploration to accelerate RL algorithms in long-horizon, multi-step and sparse

reward manipulation motion generation tasks. Abstract demonstrations leverage human priors to decompose a manipulation task and provide the correct sequences of subtasks/steps to guide the exploration directions of the RL algorithm. Adaptive exploration reduces exploratory behaviours when the learning algorithm is certain about the solution to the target subtask so that it proceeds faster to the later stage of the task and reduces the variance of the final performance.

- Implement and mathematically demonstrate that A^2 can be integrated with three popular DRL algorithms (DQN, DDPG, and SAC).
- Demonstrate the effectiveness of the A^2 method on a series of multi-step simulation tasks including discrete grid world and continuous object manipulation.

By pursuing objectives 4 - 7, this thesis makes the following contributions:

- Develop the universal option framework (UOF) that integrates the knowledge integration ability of goal-conditioned reinforcement learning (GRL) into a classic HRL framework. The UOF possesses only one policy at each decision level to learn multiple subtasks or tasks.
- Adapt a classic HRL learning algorithm (Sutton *et al.*, 1998) for goal-conditioned high-level policies.
- Mathematically analyse the root cause of the non-stationarity issue that happens to the parallel training processes of HRL algorithms and propose that A^2 can stabilise the parallel training process.
- Demonstrate the parallel learning improvements over previous methods in a range of long-horizon, multi-step and sparse reward block stacking manipulation tasks.

- Demonstrate the memory and computation reduction achieved by the proposed UOF and parallel training acceleration techniques.

By pursuing objectives 8 - 10, this thesis makes the following contributions:

- Review and summarise the state-of-the-art of robotic affordance learning according to an RL-based affordance learning framework.
- Extend an RL-based affordance theory to include the prediction of arbitrary action consequences, called *general affordances*.
- Based on the general affordance concept, develop a practical manipulation framework that selects task-agnostic grasps according to predicted manipulation performances.
- Design and implement the training processes of the general affordance-aware manipulation (GAM) framework in a series of hook-disentangling tasks in simulation.
- Demonstrate the substantial improvements on a series of hook disentangling tasks over existing methods with the use of an affordance-based grasp filter.

1.4 Outline of the thesis

The rest of this thesis is comprised of 6 chapters, whose contents are briefly introduced as follows.

Chapter 2 presents a thorough review of the problem definitions of grasping and manipulation with their classic and modern learning-based methods, covering the topics of grasp generation, motion planning, hierarchical manipulation systems, robotic manipulation simulators, and sim-to-real policy transfer.

Chapter 3 introduces the mathematical foundation of reinforcement learning (RL) algorithms, including temporal difference learning, the policy gradient theorem and three important deep reinforcement learning (DRL) algorithms.

Chapter 4 is devoted to addressing objectives 1 to 3, focusing on accelerating the end-to-end DRL approach to long-horizon and multi-step manipulation tasks.

Chapter 5 is devoted to objectives 4 to 7, seeking to develop a more efficient hierarchical reinforcement learning (HRL) framework for long-horizon, multi-step and multi-outcome manipulation.

Chapter 6 is devoted to objectives 8 to 10, seeking to improve the grasp selection process for better downstream manipulation using the concept of affordance.

Chapter 7 concludes this thesis, summarising the contributions and limitations, and proposing future research directions.

Chapter 2

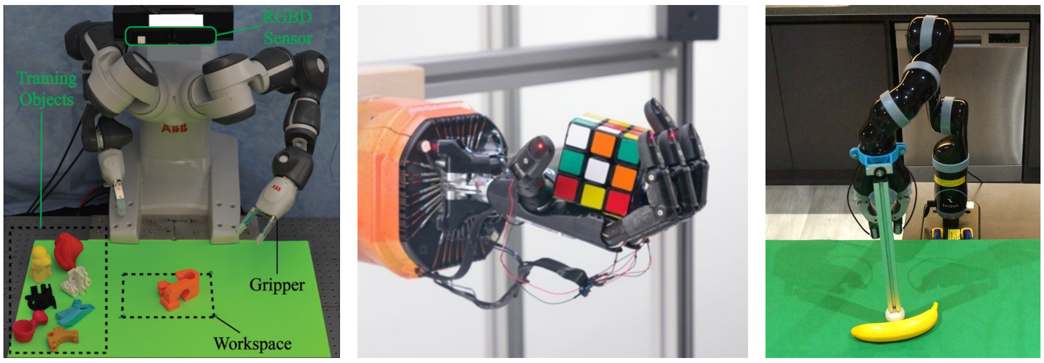
Literature review

2.1 Introduction

The rise of robotic manipulation research started under the background of the ongoing industrial revolution at the beginning of the 20th century when the automation of the production process was demanded to liberate human labour and increase cost-effectiveness (Nitzan and Rosen, 1976; Siciliano and Valavanis, 1998). Manipulation is a very broad term and includes many different problems, each of which has a very distinct objective and specific computational concerns. One of the general definitions of manipulation is:

An agent's control of its environment through selective contacts.

This definition is very broad. However, roboticists is mostly interested in reproducing manipulation skills performed by human hands. They may be classified into pick-and-place manipulation, in-hand manipulation, and non-prehensile manipulation (see Figure 2.1) (Mason, 2018).



(a) (b) (c)

Figure 2.1: Three types of robot manipulation tasks. (a): A planar pick-and-place system (Mahler *et al.*, 2017). (b): In-hand manipulation system for the Rubik's cube (Akkaya *et al.*, 2019). (c): Pushing, a non-prehensile manipulation skill (Li *et al.*, 2018)

The main distinction among the three kinds of manipulation tasks lies in the forms and dynamics of contacts established between the gripper and the target object. **Non-prehensile manipulation** refers to tasks in which the robot manipulates the object without satisfying a force or form-closure constraint (pushing, tilting, for example). This means that the object is not constrained strictly to the movement of the hand (fingers). **In-hand manipulation** refers to the study of manipulating an object whose contact locations relative to the fingers are allowed to change, but the object has to remain within the hand or between the fingers. This can be illustrated by the Rubik's cube manipulation task. The object is always held within the hand but the contact locations will be changed by the finger forces according to task requirements. When the object is allowed to rest on the palm, the manipulation becomes non-prehensile. **Pick-and-place manipulation** refers to the manipulation of a firmly grasped object, such as bin-picking, throwing, insertion, etc. In this case, the object is not expected to move relative to the contact locations established with the fingers. In other words, the contact points are not supposed to be changed after a grasp is established. Even though they may still be changed due to external disturbances or re-grasping.

The focus of this thesis is on **pick-and-place manipulation** tasks, which is itself a broad research field. In particular, this thesis focuses on multi-step and long-horizon manipulation tasks (chapters 4 and 5) and object disentangling manipulation tasks (chapter 6). To understand the specific problems studied in this thesis, the foundation and common practice of pick-and-place manipulation systems will be reviewed in this chapter.

What are the key questions?

Pick-and-place manipulation, as arguably the most fundamental and mastered ability of humans, has been proved extremely difficult and complicated to understand, model and reproduce on robots. A robotic manipulation system typically involves a few subsystems, whose relationships are shown in Figure 2.2. The perception module transforms raw observations into useful information and keeps track of the state of the robot arm and the objects (Corke and Khatib, 2011; Premebida *et al.*, 2018). The task planner concerns the representation and planning of different grasping and manipulation tasks (Garrett *et al.*, 2021). The grasp planner computes the grasping poses (Bicchi and Kumar, 2000; Caldera *et al.*, 2018). The motion planner produces arm motion trajectories (Latombe, 2012; Siciliano *et al.*, 2008). Lastly, the motion controller moves the robot arm from one configuration to another (Luh, 1983; Siciliano and Valavanis, 1998; Corke and Khatib, 2011).

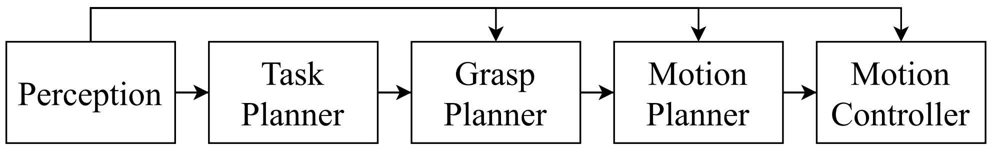


Figure 2.2: The typical relationships among manipulation software subsystems. Arrows point out the typical orders in which information is processed by these subsystems.

Pick-and-place problems, in particular, concern the generation of the motion of a robotic arm and the fingers of its gripper that move to grasp an object and move the object according to the requirements of a task. Most pick-and-place systems can be divided into subsystems that deal respectively with the *grasp planning* problem and the *motion control* problem (Billard and Kragic, 2019). The grasp planning problem seeks to find a pose for the gripper such that when it closes its fingers, the contacts established between the fingers and the target object satisfy certain criteria, such as stability, equilibrium, and so on (Bicchi and Kumar, 2000); while the motion control problem seeks to move a robotic manipulator such that it interacts with the object and the environment to achieve a task-specific goal, possibly subject to constraints such as obstacle avoidance or grasp stability (Corke and Khatib, 2011). Advanced manipulation systems take it further, attempting to incorporate a diversity of manipulation skills beyond object grasping and placing. These advanced systems are typically hierarchical control systems, integrating a high-level task planner and a low-level motion generator (Kroemer *et al.*, 2021).

The common workflow of a pick-and-place system operates in a cycle of grasp planning and motion control, with embedded perception processes such as object pose tracking or image feature extraction. Accordingly, this chapter will first introduce the grasp planning problem with its classic and modern solutions (section 2.2), then, the motion control problem with its classic and modern solutions (section 2.3). Thirdly, the classic and modern frameworks for hierarchical robotic control, the integration of task-level and motion-level control, will be introduced (section 2.4). Last but not least, as the focus of this thesis is learning-based methods which generally require a large amount of simulation data, a review of the existing robot simulators will also be provided (section 2.5).

2.2 Grasp planning

2.2.1 Background

The grasp planning problem seeks to find a gripper configuration under which the target object can be grasped in a way that supports the downstream manipulation task. There are several assumptions needed to be further clarified for this problem before the analysis and development of any specific algorithm can be conducted. Any practical solution to this problem will have to include the following specifications:

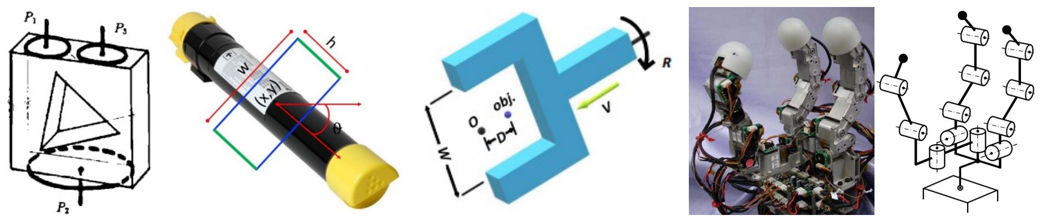
- The type, representation and detection of objects.
- The type of gripper.
- The representation of a grasp.

These specifications have been evolving since researchers started to develop grasp planning solutions. The following content of this subsection will briefly review the three assumptions made by researchers since the start of the field decades ago, and specify what assumptions are taken by this thesis. The introduction will be brief as they are not the main focus of the research problem of this thesis, however, behind each of them there is a profound research area.

Objects. The objects investigated in the 80s and 90s are mainly rigid objects in basic geometric shapes such as triangles and rectangles in 2D, tetrahedrons and hexahedrons in 3D (Shimoga, 1996). This slowly changes to rigid objects with irregular shapes (Du *et al.*, 2021), and to deformable objects (Arriola-Rios *et al.*, 2020) in recent years. Accordingly, the representation of objects has gone through significant changes, from exact models like geometrical primitive shape assembles to more complex models such as meshes, graphs, images and point clouds. Most research works nowadays use the coordinate frame of the object's centre of mass to represent the state of an object. For some cases, an associated bounding box of the object in the image of point cloud space is also provided. This simple representation is widely used for representing rigid objects with known geometric models in the manipulation research community as well as the industry (Du *et al.*, 2021). Finally, any grasp planning system requires the detection and recognition of target objects, which went from exact model-based approaches (Chin and Dyer, 1986), to vision-input detectors based on hand-crafted features and nowadays GPU-accelerated DL techniques (Zou *et al.*, 2019b; Du *et al.*, 2021).

The type of gripper. The absence of significant progress in the development of gripper types by both researchers and manufacturers since the 1990s may be regarded as surprising. They can be roughly categorised as parallel-jaw grippers, multi-finger hands, special-purpose grippers such as a suction cup or nano gripper, and soft grippers. At the beginning of the industrial automation wave, the industry was more interested in the easiest solution regarding simple-shape and rigid object pick and place tasks, which in turn brought most research attention to the design of rigid grippers with two padded-tip fingers (Lundström, 1974) and suction (Kolluru *et al.*, 1998). These grippers satisfactorily perform simple pick-and-place and assembly tasks for rigid and regularly shaped industrial components with known object models. However, they are not capable of conducting complex manipulation tasks achievable by human-like end-effectors, which can be underactuated, soft, multi-fingered, and equipped with various sensors. As a result, researchers have started focusing on the design of multi-fingered (or dexterous, anthropomorphic) hands (Oomichi *et al.*, 1990; Rahman *et al.*, 2016) as well as soft grippers and tactile sensing (Chitta *et al.*, 2011; Yuan *et al.*, 2017;

Rayamane *et al.*, 2022). Despite the importance and value of multi-fingered and soft grippers in real-life and daily manipulation tasks, most grasping and manipulation algorithms developed today still target rigid and two-fingered grippers (Du *et al.*, 2021). This is surprising since the focus of state-of-the-art research has been on moving robots from structured and perfect-sensorised industrial environments to unstructured and noisy-sensorised daily environments, such as homes, offices, and industrial settings that are unstructured (Billard and Kragic, 2019). In addition, algorithms for multi-fingered grasp generation and finger control involve a substantially more complex kinematic chain and grasp representation compared to two-fingered grippers, making it a valuable and active but under-explored research area (Rahman *et al.*, 2016; Turpin *et al.*, 2022).



(a) (b) (c) (d)

Figure 2.3: Representations of a grasp. (a): object-centric contact analysis (Nguyen, 1988). (b): planner gripper-centric representation (Kumra and Kanan, 2017). (c): 3D space gripper-centric representation (Fang *et al.*, 2020a). (d): A three-fingered hand and its joint configuration (Takahashi *et al.*, 2008)

The representation of a grasp. To clarify, a *grasp* refers to a system in which a target object is gripped by the fingers of a robot hand (Shimoga, 1996). For convenience from the algorithm development perspective, a grasp is normally represented in the task space, which, in many cases, is the space with its origin as the robot arm's base frame. There are two common representations. The first one is to represent a grasp in an object-centric way by the exact contact locations where the fingers are supposed to touch (Nguyen, 1988; Li and Sastry, 1988). The contact locations are then used to compute the joint configuration of the gripper fingers and the robot arm through inverse kinematics (Nguyen, 1988; Shimoga, 1996; Li *et al.*, 2022). In recent years researchers have become more interested in gripper-centric grasp representations, which specify the location where the gripper should move to and close its fingers. The reason is possibly due to the ease of data collection favoured by DL methods and the liberation from analysing every object geometry (Du *et al.*, 2021). These include 2D pixel locations for planner grasping, 3D coordinate locations in the task space, rectangular representations in the image and point cloud spaces (Du *et al.*, 2021). Location-based representation is easy to use for grasping tasks with parallel-jaw grippers, but it lacks enough information to allow more complex manipulation tasks that require multi-fingered grippers. Gripper-centric representations for multi-fingered hands normally include all the joint parameters of the gripper (Mayer *et al.*, 2022; Turpin *et al.*, 2022).

The grasp planning problem is to find a grasp configuration under previously given knowledge about the target objects, the gripper type and the representation of a grasp. The grasp is either constructed from object-centric methods or gripper-centric methods, and the output of the algorithm is normally a configuration of the gripper. In the case of a grasp planning system, the following specifications are usually made:

- The targets are assumed rigid and regular shape objects with known geometric information provided by a simulator or an off-the-shell object detector;

- The type of gripper used is a two-fingered parallel-jaw gripper; and
- The grasp representation is gripper-centric, consisting of the gripper tip coordinate and the gripper orientation in the task space.

To pave further the path towards manipulation, the next two subsections will introduce representative classic and DL-based solutions towards the grasp planning problem and some task constraints that have appeared in the last few decades.

2.2.2 Classic methods and constraints

Grasp planning algorithms are always developed to serve downstream manipulation tasks. To ensure the success of manipulation tasks, researchers in the 80s started to develop algorithms based on contact analysis and form-closure or force-closure conditions. Various manipulation-related constraints have been developed to improve grasp stability, equilibrium, and grasping success rates (Shimoga, 1996).

In simple terms, classic solutions attempt to find contact locations on the object such that the gripper will stably sustain the object in hand by exerting forces through the contact locations. Therefore, the grasp planning problem is transformed into a contact location optimisation problem. To compute the grasps, certain metrics or constraints are required to assess the quality of a set of contacts on an object. Among them, the two basic criteria are form-closure and force-closure (Bicchi, 1995). The *form-closure* criterion, once satisfied, guarantees that an object will not move unless it penetrates through at least one of the contact points in a purely geometric sense (Reuleaux, 2013). It was shown that for planar cases at least four contact points are required to achieve the form-closure criterion, and for spatial cases, at least seven are needed. The *force-closure* criterion, on the other hand, ensures that an object can be held static relative to the contact points despite any external disturbance (Bicchi, 1995). In general, the force-closure condition is preferred in grasp planning algorithms because it takes into account the forces from the gripper fingers, while form-closure is pure geometric constraints. This is especially true when considering a change of contact locations and forces from the fingers may change in the following manipulation processes. Also, analysing contacts without considering the kinematic facts of an actual gripper is less meaningful, because grippers differ in their capabilities of producing contacts and forces. A typical workflow of these optimisation methods is to search over the contact space and evaluate each of them with the closure property metric (Nguyen, 1988).

However, it turned out that satisfying the force-closure criterion alone is not robust enough for object manipulation. For example, not every force-closure grasp can be realised by a certain gripper. Therefore, researchers developed further grasp quality constraints on the contact locations or the forces to compute more realisable grasps (Shimoga, 1996; Roa and Suárez, 2015). These constraints include but are not limited to finding force-closure grasps that:

- are away from singular contact or finger joint configuration;
- allow forces to be applied uniformly through the contact points;
- span the contact points uniformly over the object's surface;
- minimise the influence of inertial and gravitational forces;
- are away from the boundaries of the force-closure grasp space;
- allow as many positioning errors of the actual execution;
- consider the force or joint limits of the actual gripper; and
- allow the hand to move in any direction with the same gain.

Considering these constraints, classic solutions perform well with rigid objects, near-perfect sensing conditions, and regular and known object geometries, which together feature most of the structured environments and grasping requirements in many mass-production industries. However, these algorithms fail in cases with diverse objects and objects with unknown or hard-to-define geometry. In addition, the calculation of these quality measurements often requires ideal sensing conditions and much computation time with complex object geometry, which is not realistic (Sahbani *et al.*, 2012; Roa and Suárez, 2015).

However, the main limitation is that they do not take into account the purpose of the target object to be grasped for the task. In other words, they only concern the general stability and manipulability of a contact or hand configuration. This kind of grasp is known as task-agnostic grasp (TAG) because they are computed without considering a specific downstream manipulation task (Ortenzi *et al.*, 2019). When considering a specific manipulation task, the grasping location could be further constrained semantically and more practically. For example, a hammer needs to be gripped differently when it is to be used to hammer a nail rather than to be handed out to someone. As humans, we choose to grasp an object in a certain way according to what is to be done with the target object. Such grasps are named task-oriented grasps (TOG).

The problem of TOG generation is not entirely new. It could still be considered from the same computation framework as the classic grasp planning problem, by simply adding constraints on the contacts or the forces concerning a specific manipulation task. For example, to constrain the contacts within a task ellipsoid in the wrench space (Li and Sastry, 1988; Li *et al.*, 1989; Prats *et al.*, 2007). These task-wrench spaces are defined according to human experiences. There have been attempts to learn these spaces from demonstrations (Aleotti and Caselli, 2010). On the contrary, there are also TOG solutions from the gripper-centric point of view. Dang and Allen (2012) built a semantic affordance map to relate object local geometry to a set of predefined grasp poses, such that grasp poses could be found according to different task requirements.

To conclude, one can see that for either TAG or TOG generation, analytic methods rely substantially on the known geometry of the object and a perfect sensing condition to search or solve for a grasp configuration. These two assumptions resulted in the failure of analytic approaches when the world demands a solution for unstructured, partially observable and highly stochastic environments with objects of diverse geometries and unknown physical properties. What raised to face the challenge is deep neural networks, big data and GPU-accelerated computers (Du *et al.*, 2021).

2.2.3 Deep learning task-agnostic grasp

Deep learning (DL) has made a great impact on various fields, including the robotic manipulation community. It may be necessary for the readers to get familiar with the basics of DL before reading the following contents. A brief introduction of DL is given in subsection 3.2.1 or for a thorough one in (Goodfellow *et al.*, 2016).

Before discussing specific grasp planning methods, there is a distinction needed to be made. The grasp planning solutions discussed in the last subsection require a known object model. However, note that many vision-based grasp planning solutions employ object detection methods, feature-based or learning-based, to estimate the location and orientation of the target object (Zou *et al.*, 2019b; Du *et al.*, 2021), but the grasps are still computed based on the same analytic principles. The main effort is to correspond a perceived object with a known object model in the database and thus the computed grasps. These approaches may be called *model-based vision-based grasp planning*, so that the readers would not be confused as the following content will discuss DL solutions for what may be named *model-free vision-based grasp planning*. The main difference with the last subsection is the absence of a known object model. A graphical relationship is shown in Figure 2.4.

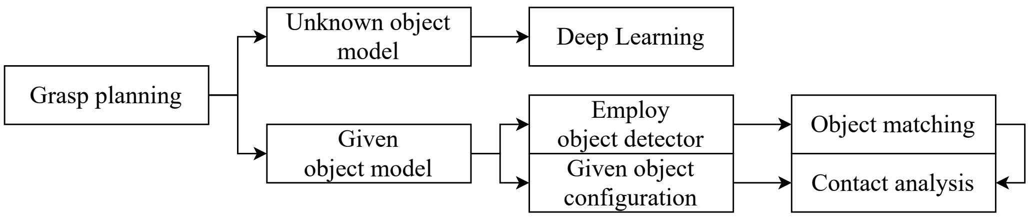


Figure 2.4: Grasp planning method taxonomy.

To liberate grasp planning from the requirement of requiring knowledge about the geometric model of the object, DL solutions can be employed. In this approach, a DNN is trained to extract latent features from the data, which will then be mapped to the desired outputs. In the case of vision-based grasp planning, the input is sensory data, such as an image or a point cloud, and the output can be the grasps or the evaluation metrics of the grasps. The assumption behind this is that, with enough amount of data, the neural network can learn features that are representative enough that it can generalise to unseen inputs. Or in other words, the neural network, as a function, will be able to approximate close enough the desired mapping between two distributions given a sufficient amount of data (Goodfellow *et al.*, 2016). These methods can be classified into supervised learning and RL methods. The following content introduces briefly some representative works regarding the two topics. For a thorough review, please refer to (Newbury *et al.*, 2022).

Supervised learning refers to methods that train a neural network with a predefined and unchangeable dataset. For 2D planar grasp planning, researchers seek to evaluate the grasp quality of either contact points or rectangular grasps on the image plane using popular convolutional neural network architectures (Lenz *et al.*, 2015; Kumra and Kanan, 2017; Mahler *et al.*, 2017). For example, Levine *et al.* (2018) employed 14 real-world robotic manipulators to collect 800k grasps to train a DNN that outputs a success rate given an image and a motion command (a 3D vector relative to the robot's base frame). This work successfully demonstrates the power of data-driven supervised learning methods, albeit costly and impractical. Mahler *et al.* (2017) trained a DNN which takes as inputs a depth image aligned to the labelled grasping centre and orientation, and a 2.5D point cloud image labelled with gripper distance. It then outputs a probabilistic quality score of the given grasp. After training, the DNN was treated as a fixed objective function with the Cross-Entropy Method (CEM) to generate grasps that optimise the DNN output. Following a similar idea, Lu *et al.* (2020) provided a DNN with an 8-channel representation of an object (RGBD, surface normal, curvature) and the configuration of a multi-fingered hand to evaluate the grasp quality. They proposed another DNN for comparison, taking three image patches of an object corresponding to the palm pose and fingertip locations of a multi-fingered hand. It has been demonstrated that the patch-DNN performs much poorer than the config-DNN, which indicates that access to the configuration space allows DNNs to catch more valuable information for multi-fingered grasp quality evaluation.

3D spatial grasp generation methods also adopt a similar idea of grasp quality evaluation for 3D points (Mousavian *et al.*, 2019; Fang *et al.*, 2020a; Zhao *et al.*, 2021). For example, the *6Dof GraspNet* system uses a variational auto-encoder to randomly sample grasp configurations in the 3D space and assesses and refines the proposed grasps with a quality evaluation network (Mousavian *et al.*, 2019). The auto-encoder maps example grasps to a latent space, which is minimised towards a normal distribution during training. The evaluation network is trained to reject grasps that result in collision, penetration and grasps that are far away from the object. The refinement process finally looks for a transformation matrix that increases the success probability by differentiating the evaluation network with respect to the grasp pose. Their dataset is generated through simulation by sampling grasps, executing them and evaluating them with a predefined shaking motion. *GraspNet1Billion* takes another network architecture for 3D grasp evaluation, including an approach net, an operation net and a tolerance net (Fang *et al.*, 2020a). The approach net takes point cloud features as inputs and predicts approaching vectors and the graspability scores. The operation net then takes point cloud features and predicts evaluation scores for grasps generated around and along the approaching vectors. The tolerance net learns to further predict the tolerance towards perturbation for each grasp sampled by the operation net. The training dataset was again generated by computer simulation with the use of the force-closure criterion. Another representative work, *REGNet* proposed another kind of pipeline, consisting of a score network, a grasp region network and a refine-network (Zhao *et al.*, 2021). The score network takes into point cloud features and assigns each point a grasp confidence score. The grasp region network then takes the point cloud features for the k_1 points with the highest scores ranked by the score-network and proposes coarse grasp candidates for each point as grasp anchor. Finally, the refine-network seeks to predict the distance between ground-true grasp poses and the proposed grasp anchors. Similar to other works, the dataset is generated in simulation with force-closure and collision avoidance measurements.

RL methods seek to produce a policy that predicts grasps by learning from trials and errors. The main difference with supervised learning is that RL methods do not rely on a fixed dataset, but generate training data with the current policy. For an introduction to RL, please refer to chapter 3.

Following the success of Deep Q Network (DQN) (Mnih *et al.*, 2013) in other fields, James and Johns (2016) proposed a relatively simple neural network to estimate the q function for simulation grasping tasks. Arguably the first attempt on a large-scale RL project for vision-based grasp generation would be the Qt-Opt system, where a neural network is learnt to evaluate randomly sampled grasps from over 580k data points collected by 7 Kuka robots (Kalashnikov *et al.*, 2018). Another representative work proposed to use the Trust Region Policy Optimization algorithm (Schulman *et al.*, 2015) to train a neural network-based policy for robotic grasping (Breyer *et al.*, 2019). The action of the policy includes the relative pose and openness of the gripper. The policy is represented as a Gaussian distribution with means and variances given by a neural network. The reward function is hand-engineered to give a monotonically increasing signal towards the task goal. The policy is trained in a simulation environment with varying task constraints and states and then transferred to the real world. Lastly, Wu *et al.* (2019) proposed a novel 4-branch policy network that employs an attention mechanism to learn a grasping policy for vision-based 4-DoF multi-fingered grasping. This work takes only depth images as input and recursively cropped the image to generate attended regions where the relevant information for making decisions exists. The action space consists of 4 parts: 1) the end-effector position for grasping or centre location for zooming; 2) the zooming decision and scale; 3) the roll, pitch and yaw of the end-effector; 4) the pre-grasp joint angles. It uses sparse task completion reward signal and the Proximal Policy Optimisation algorithm (Schulman *et al.*, 2017b).

In sum, DL techniques have enabled the first attempts at model-free vision-based grasp planning algorithms, especially for the generation of task-agnostic grasps. However, the question of how to generate grasping poses that are not just stable but also compatible with the following manipulation task (Mason, 2018) remains underdeveloped. This leads researchers to the study of DL-based task-oriented grasping (TOG).

2.2.4 Deep learning task-oriented grasp

Computing grasping poses that suit different downstream manipulation tasks is not a very new idea (Li and Sastry, 1988; Sahbani *et al.*, 2012), but researchers only started in recent years to employ deep (reinforcement) learning methods to tackle the TOG generation task (Mason, 2018; Ortenzi *et al.*, 2019; Billard and Kragic, 2019). Similar to the TAG generation problem, TOG generation methods before the rise of DL exhibited low generalisation performances (Li and Sastry, 1988; Sahbani *et al.*, 2012). It is difficult to have hand-designed features or geometric models that adapt to unknown objects for TAG generation, let alone the more complicated cases of TOG generation.

The most straightforward way to generate TOGs is to classify TAGs according to some manipulation task objectives. There are two paths taken by researchers, with the first one following the idea of DL-based image segmentation (Minaee *et al.*, 2021), labelling parts or regions of an image or an object for different tasks. For example, Detry *et al.* (2017) proposed to predict which region of an object to grasp will suit a specific task. Kokic *et al.* (2020) employed the same idea but proposed to learn such task-related grasping regions from human activity videos. This idea of using image segmentation has also been pursued by researchers in the field of affordance learning (Kokic *et al.*, 2017; Mandikal and Grauman, 2021a). The predicted regions can then be used to select task-oriented grasp poses from task-agnostic ones. The second path is to perform classification on the grasp configurations directly. For example, Murali *et al.* (2021) proposed a graph convolutional network to evaluate whether a TAG grasp suits a specific task by supervised learning. Noticeably, they constructed arguably the largest 6-DoF task-oriented grasp dataset at the time this thesis is written (191 objects, 250K TAG poses and 56 task labels per object). Similarly, Sun and Gao (2021) proposed to achieve TOG by evaluating TAG poses for specific tasks, but they relied on translational distance models to learn the relationships among tools, actions and target objects, and to evaluate the task-oriented grasp quality.

Other methods exist without assuming access to a TAG generator. Yang *et al.* (2019) proposed to predict stable grasps with suitable task labels for stacked objects. They hand-labelled a dataset with 8000 depth images of stacked objects (10 object categories and 11 tasks). Fang *et al.* (2020b) took a self-supervised approach that jointly trained a task-oriented grasp evaluation neural network and a manipulation policy for a diverse set of hammer-like objects. The cross entropy method was used to optimise grasps over the grasp quality network. Wen *et al.* (2022) also proposed a self-supervised approach to learn category-level task-oriented hand-object heatmap. They modelled the heatmap over the canonical model of an object category and achieved zero-shot sim-to-real transfer through various transferring techniques. Kwak *et al.* (2022) used a graph convolutional network to model the relationships among the semantic meanings of a scene, including the type of gripper, task, object, object component, and grasping force (each component comes with a predefined grasp and each task comes with a given manipulation motion).

2.2.5 Summary

This section provides an introduction to the grasp planning problem, along with the problem assumptions considered in this thesis and its classic and modern, DL-based solutions. Despite that state-of-the-art grasp planning methods demonstrate promising results in generating task-agnostic and task-oriented grasp configurations for a variety of daily objects, such as shown by (Fang *et al.*, 2020a; Murali *et al.*, 2021). there remain many challenges for future work. For example, DL-based TAG generation is far more complicated with multi-fingered and soft grippers due to the complexity of contact dynamics. Existing TOG generators fail to reason about possible manipulation outcomes in a detailed manner. For more details, please refer to relevant literature (Billard and Kragic, 2019).

This thesis is less related to TAG generation and more to TOG generation and applications. In particular, this thesis focuses on how to develop manipulation systems that reason about the long-term relationship between a grasp and a manipulation task in a more detailed manner. To this end, a necessary review of the manipulation trajectory generation problem and advanced hierarchical manipulation systems will be given in the next two sections.

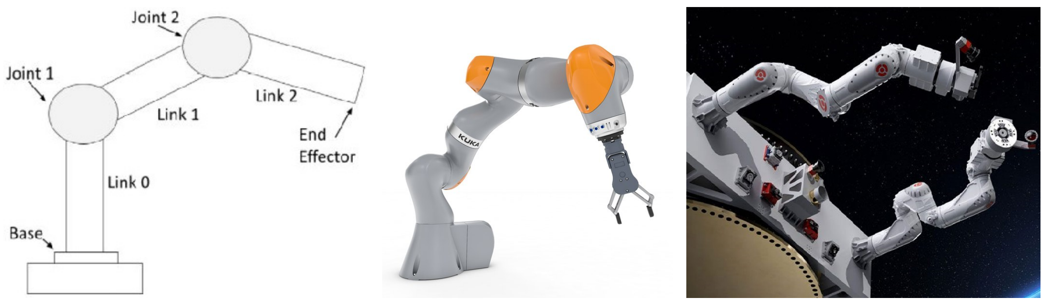
2.3 Manipulator control

2.3.1 Background

The problem of manipulator control seeks to generate a trajectory for the manipulator that changes the state of the object according to the task specification. There are several levels and different perspectives regarding this problem to be considered. The following sections will introduce the fundamental concepts of manipulator control algorithms, which serve as the foundation for comprehending the classic and modern solutions in this field. Further detailed descriptions of the fundamentals can be found in (Siciliano *et al.*, 2008; Lynch and Park, 2017).

The manipulator and its kinematics

The type of manipulator varies in its design and function. In this thesis, we consider robotic arms that are composed of rigid links, a common choice for pick-and-place tasks. A robotic arm mostly refers to a human-arm-like device equipped with a gripper at the end of its chain of links. The relative movement between each pair of links is always constrained, and the constraint is normally referred to as a joint. For example, a manipulator with 7 links is composed of 6 joints, each of which normally allows only one degree of freedom for the relative movement of each pair of links. Most industrial manipulators are composed of prismatic and revolute joints, which allow either only translation or rotation. With each joint providing one degree of freedom, a 7-link manipulator will have 6 degrees of freedom at the frame of its last link. Additionally, there is normally a changeable link that is attached to the last link of the manipulator, called an *end-effector*, such as a gripper. For a 7-link manipulator, the end-effector frame will have 7 degrees of freedom. In pick-and-place tasks, when an object is grasped, it is considered to be an extended part of the end-effector as it is assumed to be static relative to the gripper (the connection is thus a fixed joint of no degree of freedom). Some example robotic arms are shown in Figure 2.5.



(a) (b) (c)

Figure 2.5: Examples of robot arms. (a) A robot arm with 3 links and 2 joints. (b): The LBR iiwa robotic arm (7 joints) (Kuka, 2022). (c): The ModuLink space manipulators (Motiv-Space-Systems, 2022)

A chain of links connected by joints is referred to as a kinematic chain. Given every constraint that the joints impose on each pair of links (i.e., the parameter of each joint), the pose of the end-effector can be calculated using kinematic equations, which are composed of spatial transformation matrices. This computation process is called *forward kinematics*. Inversely, the process of finding the parameter of each joint according to an end-effector pose is called *inverse kinematics*. The vector space of all the parameters of the joints is called interchangeably the *joint space* or the *configuration space*, while the position and orientation of the end-effector are in the 3D Euclidean space, which is referred to interchangeably as the *task space* or the *Cartesian space*. It is normally assumed that the origin of the task space is aligned with the frame of the base link of the robot arm for computational convenience.

With these fundamentals, it is ready to consider how to control the robot arm to manipulate an object. There are at least three levels of algorithms in the object manipulation task.

Control

The lowest level focuses on how to move a robot from one configuration to another configuration. This level is regarded as the control level, in which a feedback controller outputs the velocities or the torques/forces that are required at each joint according to the motion or force requirement. There are various kinds of controllers to deal with different control requirements, such as motion controllers, force controllers, and impedance/admittance controllers. The controller that is assumed available for the tasks of this thesis is the joint position/velocity PID controller and task space position PID controller. The field of robot control is well-developed (Luh, 1983) and will not be reviewed here.

Motion generation

The second level focuses on finding a trajectory (a set of waypoints) for the end-effector or the object from the current system state to a target state. These algorithms assume access to a lowest-level feedback controller to realise the robot motion that follows the output trajectories.

The joint space trajectory generation problem is often referred to as the *motion planning* problem. It concerns finding a trajectory from one point to another in the configuration space considering various constraints. Motion planning can be performed online (if fast enough) or offline (if accuracy calls), with exact or approximate inputs, with full or partial observations, with or without time constraints, with or without obstacles, etc. The field of motion planning is very mature and the community provides readily applicable open-source toolkits such as the MoveIt! package (Chitta *et al.*, 2012).

Most motion planning algorithms need to convert object representations into the joint space and plan for a trajectory in the joint space. However, object manipulation tasks involve complex dynamics (high dimensional joint space and complex irregular shape objects) that are difficult to model and perceive in the first place, let alone to be converted into the joint space. Therefore, instead of building an explicit joint space obstacle representation, collision-checking algorithms are used to trade accuracy for computation complexity (Janson *et al.*, 2015). Overall, there remain many challenges with motion planning for complex real-world problems (Kroemer *et al.*, 2021). In recent years, researchers have turned to (deep) learning algorithms for new perspectives.

The first idea is to equip the existing motion planning algorithms with a DL-powered system, which estimates the pose information or state required by the planning algorithms (Sadat *et al.*, 2020). This idea relies substantially on 2D and 3D object detection to enhance the visual perception capability (Zou *et al.*, 2019b). The second one is to replace the motion planning module with a function approximator that takes the observation of the environment and produces motion commands for the robot to follow. Such a function approximator is typically called a *policy*, which is optimised with different objectives: imitation learning (IL) or RL. Thirdly, motion generation methods commonly assume access to a known dynamic model of the robot and sometimes the environment. Some modern approaches seek to replace the human-expert-based dynamic model with a model learnt from data, with which planning can still be performed under the estimated (partial) model (Moerland *et al.*, 2020).

Task planning

From the highest level, a manipulation task may be complex and may need to be decomposed into several subtasks, achieved by different skills. The task level representation is often in the task (Cartesian) space or camera space, as it is much easier to specify and interpret. For example, moving towards a grasping configuration, grasping the object, moving towards a dropping location and finally releasing the fingers. This is referred to as task planning, which concerns chaining a set of abstract states, manipulation skills or primitives to achieve a final task goal (Karpas and Magazzeni, 2020). In many cases, decomposing a task into a set of subtasks and chaining a series of skills can be equivalent. A skill may be obtained in various manners, such as hand-engineering, motion planners or learning algorithms.

Existing task planning methods mostly concern the problem of how to chain a set of manipulation skills by planning over the discrete, symbolic skill space, where each skill is hand-engineered or comes from a motion planning algorithm (Garrett *et al.*, 2021). Modern learning-based methods attempt to replace planners with reactive policies learnt from data. This can happen at the task level as well as at the motion generation level (Kroemer *et al.*, 2021). Important research topics are many. For example, the design and learning of motion skills, the planning and composition of motion skills, the exploration and searching difficulty due to long task horizons, the transferability of the motion skills, the simultaneous training of the whole hierarchical system, and so forth (Kroemer *et al.*, 2021; Pateria *et al.*, 2021b).

Section organisation

According to the analysis of the three levels of manipulator control, the literature review will be divided into two sections. Methods regarding the lowest level are out of the scope of this thesis. The rest of this section will focus on model-based planning and data-driven/learning solutions for the second level – motion generation. The next section will review hierarchical control systems that deal with long-horizon tasks and planning with skills.

2.3.2 Classic motion planning

The motion generation problem seeks to find a trajectory over discrete time steps for the robot to follow, reaching a task goal state and subject to some task constraints. A trajectory comprises a series of coordinates in the joint space, task space or camera space. It is generally assumed that a trustworthy low-level controller exists to control the robot to move from one point of the trajectory to the next one (Latombe, 2012).

Motion planning methods are one of the most popular methods for robot motion generation. It seeks to find trajectories in the configuration space. Therefore, it converts the objective, all the obstacles and constraints into the robot's configuration space. In general, denote the configuration space as $\mathcal{C}$, the subset of configuration space that is occupied by obstacles as $\mathcal{C}_{obs}$, and the free space as $\mathcal{C}_{free} = \mathcal{C} \setminus \mathcal{C}_{obs}$, the motion planning problem is to find a path in $\mathcal{C}_{free}$ that connects a start joint configuration $\boldsymbol{q}_0$ and a goal joint configuration $\boldsymbol{q}_G$[1].

There are four categories of motion planning algorithms (Lynch and Park, 2017) which are described in the following paragraphs. The first three are

[1]However, in practice, for tasks with high-dimensional configuration space and complex obstacles that cannot be explicitly represented in the configuration space, a collision-checking algorithm is required to ensure the planned solution is collision-free approximately (Janson *et al.*, 2015; Noreen *et al.*, 2016). The details of collision-checking methods are beyond the scope of this thesis and are not discussed.

based on graph theory (West *et al.*, 2001), while the last one uses energy functions.

Path planners deal with the pure geometric variant of the motion planning problems, finding a geometrically collision-free path to connect the start and end point without considering the dynamics of the robot arm or any constraints on the motions. It assumes access to the exact geometry of the environment and that the path is realisable as long as it does not penetrate any obstacle geometrically.

Grid search algorithms search over a discretised configuration space for the desired path. It is called "searching" because it constructs a path in a graph by selecting child nodes according to some heuristic cost functions. These include the Breadth-first search method, Dijkstra's algorithm, the A^* algorithm, etc. They become impractical when the search space is too large for high-dimensional configuration space with a high-resolution discretisation. In contrast, low-resolution discretisation of the search space gives poor-quality paths.

Sampling algorithms can construct paths on higher-dimensional spaces including the combined space of configuration and velocity, without discretisation of the map. For example, the rapid-exploring random tree (RRT) method chooses expansion nodes from the start by sampling points with a random distribution over the search space, selecting the nearest point in Euclidean distance and planning a motion to move a small distance away in the direction of the nearest node. There are a variety of works proposing different choices on the sampling distribution, the nearest point evaluation function and the small distance motion planner (Noreen *et al.*, 2016). Another example is the probabilistic roadmap (PRM) algorithm which samples points from the space to construct a non-directional graph representation of the search space. Typically a search algorithm such as A^* is applied to find a path after the roadmap is constructed.

Virtual potential field methods are inspired by the concept of natural potential field in physics. Simply put, they construct energy functions of a goal configuration and the obstacle configuration such that these functions impose a gradient field pulling towards the goal and pushing away from the obstacles. The principle is similar to a gravitational field. A manipulator may use a potential field of the task space to avoid obstacles and move to the goal location by following the direction of the gradient of the field.

Summary

Motion planning is a well-developed class of motion generation methods. Its development towards more real-world and high-dimensional problems and its integration with learning-based methods are still ongoing and active nowadays. For example, using a learnt neural network to construct the sampling distribution (Wang *et al.*, 2020) or evaluate which direction is more valuable to explore for the RRT* algorithm (Chiang *et al.*, 2019). For tasks that are easy to be represented in the configuration space or whose accuracy and computational requirements can be satisfied with a collision-checking algorithm, motion planning is efficient and reliable.

However, when it comes to object manipulation, motion planning faces the difficulty of high dimensionality, high-precision requirements and complex contact dynamics. For example, an object may be moved when a manipulator follows a planned trajectory to a grasp pose. To cope with such dynamic behaviours, motion planning requires to re-plan a trajectory considering constraints based on the complex contact dynamics, which is difficult to model. Consider another example where a manipulator needs to separate a grasped object from a pile of objects, a complex object dynamic model is required to plan a trajectory. Even if such a model is available, it is still inefficient to perform searching or sampling algorithms in the combined configuration space of the manipulator and the objects or conduct collision-checking with a high-precision dynamic model. The curse of dimensionality is not easy to break.

Therefore, researchers in recent years have been developing methods that learn a manipulation policy that outputs motion commands from observations in a reactive way. The main benefit is that, either through human demonstrations or a reward signal, the policy can learn from data to control the robot without prior knowledge of a dynamic model. This is particularly preferred when an analytic model of the world is not accessible, such as in many object manipulation problems.

2.3.3 Motion policy learning

This subsection introduces methods that seek to find a policy that produces motion trajectories in an online and reactive manner based only on observational feedback of the environment. From this perspective, the problem is most commonly formalised as a Markov decision process (MDP), where a robot takes an action based on its observation of the environment in a discrete-time horizon (Sutton and Barto, 2018). In such settings, a policy is defined as a mapping from the observation space to the action space, and the problem is to find a (sub-)optimal policy for a certain optimisation objective. It is considered learning because these algorithms seek to optimise the policy using data from interaction with the environment.

Depending on how the objective is formalised, there are loosely two broad sets of algorithms. *Imitation learning* refers to algorithms that find a policy to match another policy using data. *Reinforcement learning* refers to algorithms that find a policy that maximises some cumulative reward signal.

Imitation learning

The objective of Imitation Learning (IL) is to optimise a policy by minimising its distance from a target policy. For example, to mimic the locomotion behaviour of an animal or the manipulation skill of a human expert. In general, it is assumed that the exact analytic form of the target policy is unavailable, and the learner policy is optimised using data collected by the target policy. This is the idea of learning from demonstrations. As IL is not the focus of this thesis, the following will only present a few representative design choices specific to IL methods: the data collection process, the action representations, and the final policy refinement. For a thorough review, please refer to (Hussein *et al.*, 2017; Fang *et al.*, 2019a).

Data collection in IL is mostly performed by humans. A common choice is to use simulation software with interactive devices such as a mouse, keyboard, joystick controller or virtual reality device (Fang *et al.*, 2019a). Many robot manipulators support kinesthetic teaching, which allows a human to move the robot arm by pushing and pulling (Kormushev *et al.*, 2011). With human efforts, these methods are not likely to result in a large amount of data due to their inefficiency. When a large amount of data is necessary, some works use hand-crafted programmes, state-based policies or a motion planner to collect data for a learner policy that is learning over a different observation space or seeking to improve over the teacher policy (Sasaki *et al.*, 2018).

The action representation generally can be categorised into discrete and continuous. For discrete actions, the IL problem is essentially a classification problem. A learner policy is predicting the probability of the current observation belonging to the actions, i.e., classifying an observation into actions. For continuous actions, it is a deterministic regression problem or a stochastic distribution matching task. The learner policy is predicting the same actions given by the collected data. Both examples can be found in (Guo *et al.*, 2014).

Policy refinement is generally preferred because the policy learnt from a fixed set of data does not guarantee an optimal match to the target policy. In fact, it only guarantees to match the actions on observations existing in the data. Because the dataset is finite and mostly insufficient, IL methods require some refinement process. There are a few pathways to choose from. The first one is to continue training the policy with RL. In other words, using IL to initialise a policy for RL (Guenter *et al.*, 2007). Secondly, it can be used to form an initial guess of the solution for optimisation approaches (Ortega *et al.*, 2013). Another way to refine the imitated policy is to aggregate the dataset with new data (Ross *et al.*, 2011).

IL approaches are promising in that they can leverage offline datasets. For example, there is a vast amount of videos on the internet. Also, it can be used to provide a good initial policy for further training and refinement. For robotic manipulation tasks, there are several challenges for IL (Fang *et al.*, 2019a). One of them that is related to one contribution of this thesis is the difficulty of collecting kinesthetic demonstrations. This is well-known as real-world robots require certain expertise to operate, let alone the repetitive labour required to collect demonstration data. The problem is further exacerbated when considering manipulation tasks with long time horizons and multiple subtasks (Nair *et al.*, 2018). It is also why researchers are interested in learning from a small number of demonstrations (Abdo *et al.*, 2013; Johns, 2021). However, the fewer demonstrations, the higher the proportion of the mismatched policy distribution.

The method proposed in chapter 4 takes advantage of an abstract type of demonstration data, although the data is not used for IL, but for accelerating RL. It demonstrates the value of using abstract demonstrations, instead of low-level kinesthetic demonstrations, in the learning of long-horizon multi-step tasks.

Reinforcement learning

Reinforcement learning (RL), especially deep reinforcement learning, has been studied since the last decade because it provides a natural framework and mathematical expression of behavioural learning (Sutton and Barto, 2018). However, it only became one of the main research interests in robotics in recent years due to its successes in the computer game domain thanks to DL and advanced computing hardware (Lazaridis *et al.*, 2020). In short, RL refers to a set of machine learning algorithms that learn a behaviour policy that maximises the cumulative reward value. DRL refers to RL methods that use deep neural networks as function approximators. Since RL is the focus of this thesis, a more in-depth introduction of the mathematical foundations and related advances are provided in chapter 3. This subsection will briefly review the norms and the challenges faced by applying RL in robotic manipulation motion learning.

Typically, an RL algorithm iterates among the processes of data collection, policy evaluation and policy improvement (Sutton and Barto, 2018). For specific algorithms, representative works of robotic behaviour learning include deep deterministic policy gradient (DDPG), soft actor-critic (SAC) and proximal policy optimisation (PPO). Briefly, DDPG seeks to learn a deterministic policy with data coming from another exploratory policy (off-policy data), PPO seeks to learn a stochastic policy with data collected by the same policy (on-policy data), while SAC seeks to learn a stochastic policy from off-policy data to maximise return as well as the policy entropy. Their details are given in chapter 3. Manipulation motion learning research with RL tends to apply variants of these algorithms (Singh *et al.*, 2021). Despite the algorithmic advantages and disadvantages of the three (see chapter 3), they encounter more or less the same challenges when dealing with robotic manipulation motion learning.

Reward design is the first challenge for learning manipulation skills. A known issue with RL is that its policy exploits the reward function. Three pathways have been studied. The most used one is to manually design a dense reward function. For example, Gu *et al.* (2017) tailored different reward functions for a reaching task, door-opening task and pick-and-place task based on desired object positions and arm and finger movements. Note that more human priors in the design of the reward function results in the easier success of learning a specific behaviour, but a less representative, general and transferable policy. On the contrary, less prior means longer exploration and slower convergence, but less biased and more general behaviours (Eschmann, 2021). Shaping a reward that is unbiased and able to guide faster learning requires an expert understanding of the target application domain. When this is too difficult, the other two paths can be considered.

The first choice is to use a *sparse* reward function that only provides meaningful signals at task completion and system terminal states. This is employed by many manipulation learning works (Liu *et al.*, 2021), as well as this thesis. The primary focus of sparse reward RL is how to guide the exploration process so that the policy is more likely to reach the rewarding states (Ladosz *et al.*, 2022). This will be discussed later. Another path is inverse reinforcement learning (IRL) or apprenticeship learning, trying to learn the reward function from demonstrations collected by other policies before learning a policy (Abbeel and Ng, 2004). Human experts are the source of most demonstrations. For example, Abbeel *et al.* (2010) used IRL to learn a reward function for controlling a helicopter and Orbik *et al.* (2021) for controlling a multi-fingered hand. IRL could be regarded as IL via reward function learning. In practice, it faces the same challenges as other IL methods, such as the difficulty of collecting robot device demonstrations and the distribution mismatch problem in the reward function space (Arora and Doshi, 2021).

Exploration is an essential part of RL algorithms and the main concern of solving sparse reward tasks. The sparsity is generally caused by the fact that the unrewarding section of the combined space of states and actions is too large. Exploration is hard for robotic manipulation RL tasks. Imagine a task of searching for a single point in a high-dimensional space, a similar issue happened to the classic search-based motion planning problem: the curse of dimensionality with too less feasible solutions. It can be improved in a task-agnostic or task-specific fashion.

Task-agnostic exploration seeks to encourage the algorithm to explore the environment as much as possible, which is mainly realised by giving an extra reward irrelevant to the specific task. This is also referred to as curiosity or intrinsic motivation that is from the inside of an agent (Aubret *et al.*, 2019). The mathematical form of the intrinsic rewards may be based on the familiarity with a state (Tang *et al.*, 2017), prediction error (knowledge about a state) (Burda *et al.*, 2019) or information gain (Houthooft *et al.*, 2016). For small-scale experiments, the exploration can be conducted to minimise *regret* (how valuable the untaken actions are) (Ortner *et al.*, 2020). Discouraging exploration can be valuable as well. For example, when the policy is considered well-learnt then too much exploration may decrease performance or even cause divergence. After all, it is necessary to reduce exploration for a policy to converge (Sutton and Barto, 2018). It defaults to linearly or exponentially decrease exploration probability towards a lower bound (Guo *et al.*, 2014; Andrychowicz *et al.*, 2017; Paavai Anand *et al.*, 2021), however, it is more interesting to adjust exploration based on the online learning progress (Haarnoja *et al.*, 2018; Liu *et al.*, 2019).

Task-specific exploration generally requires certain human priors for task design. This can be realised by curriculum learning (Narvekar *et al.*, 2022). For example, decomposing a manipulation task into subtasks. One framework for curriculum learning is GRL (Andrychowicz *et al.*, 2017; Fang *et al.*, 2019b). Another way to embed human priors in exploration is to design a task-specific exploratory policy for data collection, such as an expert policy (Subramanian *et al.*, 2016). Initialising a policy with human demonstrations and IL can be considered a form of task-specific exploration. How demonstrations are collected and used to guide exploration is another important study topic (Vecerik *et al.*, 2017; Ravichandar *et al.*, 2020). As mentioned in the IL part, kinesthetic trajectories demonstrated by other agents such as a human expert (Ravichandar *et al.*, 2020) can teach the robot accurate manipulation skills, but they are time-consuming and difficult to collect in either simulation or the real world, especially for long-horizon manipulation tasks (Vecerik *et al.*, 2017; Nair *et al.*, 2018). In addition, a kinematic demonstration may solve a task once, but how to discover reusable skills from kinematic demonstrations remains largely unsolved (Zhu *et al.*, 2022).

Summary

This subsection discusses relevant literature on learning manipulation motion skills through imitation or reinforcement. Their advantages over classic motion planning methods are the flexibility and fast computation provided by a task-specific policy and the ease of deployment by learning from data. However, learning a reactive policy is no easy task itself, especially for robotic manipulations. Working with complex tasks that cannot provide a dense reward function, it faces the difficulty of collecting data efficiently, either for demonstrations or explorations.

2.3.4 Summary

This section covers motion planning and learning methods for generating manipulation motions. A problem with these motion generation methods is the lack of ability to handle long horizon and multi-step tasks. Typically such tasks cause considerably long planning or exploration time if learnt end-to-end, making it too difficult to find a feasible plan or encounter a meaningful learning signal.

Consider an example of stacking a few blocks into a specific order. The first block needs to be grasped before being placed. The second needs the first block to be placed in advance. Planning a motion trajectory from nothing to the goal configuration is prohibitive due to the large space of solutions and the combinatorial effects. Learning one requires a massive amount of data and poses serious exploration issues under delayed and sparse rewards. The first contribution of this thesis is to deal with such exploration difficulty (chapter 4).

However, learning a long-horizon manipulation policy end-to-end is less meaningful in terms of the reusability of the skills. It is preferred in real systems to use manipulation skills repetitively in many different tasks. Therefore, researchers advocated for the use of hierarchical control systems, in which a task-level planner or policy produces subtasks or subgoals (or select skill primitives) and another planner or policy at the lower level produces motions for the subtask. This is the research interest of the second and third contributions of this thesis (chapter 5 and 6), while the next section will first review what has been studied in the literature.

2.4 Hierarchical manipulation systems

A real-world manipulation system is not likely to only be a single and all-rounder module that is capable of everything. After all, the human brain certainly has a hierarchy of subsystems, each of which makes decisions at a different level of abstraction, as confirmed by the daily activities humans experience and by research evidence (Grafton and Hamilton, 2007). Thus it is not surprising that hierarchical computing frameworks have been developed to reproduce such decision-making processes on robots. To deal with long-horizon and complex assembly tasks in industries or messy and rich daily manipulation tasks in people's homes, robots need to make decisions at different levels (Kroemer *et al.*, 2021).

As introduced in subsection 2.3.1, there is at least a task planning level on top of the motion generation level. In robotics, the representative set of methods is called task and motion planning (subsection 2.4.1). While in modern days, learning-based hierarchical control systems can be an integration of classic planners and learnt reactive policies, or can be comprised of reactive policies learnt from data with different objectives at different levels (subsection 2.4.2). The last subsection will discuss another perspective towards hierarchical control: affordance learning. The focus of the last part will be the integration of the concept of affordance and robotic manipulation. This is also a core concept applied and extended by the last contribution of this thesis (see section 6.2 for a theoretical introduction from the RL perspective and chapter 6 for a manipulation experiment).

2.4.1 Task and motion planning

As mentioned, the motion planning problem formulation needs to be extended to include task-level planning for solving long-horizon object manipulation tasks. One approach is to extend the solution space for planning to include 1) a combined configuration space of the manipulator and the movable objects, and 2) a discrete space associated with discrete changes, such as forming and breaking contacts. The objects are treated as non-actuated free joints that can only be moved by contact forces imposed by interactions. This is the problem of multimodal motion planning, where an algorithm is required to generate plans for a hybrid space of discrete and continuous variables (Garrett *et al.*, 2021). An important notion in multimodal motion planning is the *modes* of the system, referring to different continuous configuration spaces induced by the discrete events that happened to the system. For example, the configuration space of a mode in which an object is not grasped and one of a mode in which an object is grasped will pose different kinematic constraints, thus, different search spaces. Mode switch corresponds to a change in the structure of the system's kinematic tree, such as a free joint to a fixed joint. The classic PRM and RRT algorithms could be extended to be a multimodal variant (Hauser and Latombe, 2010; Barry *et al.*, 2013).

Many works tried to extend motion planning methods to plan simultaneously over discrete and continuous spaces, while others suggest separating the two planning processes. In the AI community, planning over a symbolic and discrete space has been a long-standing area of research interest. A thorough task planning review was conducted by Karpas and Magazzeni (2020). The basic process behind all these methods is to first use human knowledge about the specific task to define a discrete state space and plan over it given an objective. For example, a block pick and place task can be described by the combinations of actions: $\{pick[], place[], moveRob[]\}$, a location extractor: $loc[]$, and objects $\{Block, Robot, Table\}$. A solution can be described as

$$\begin{aligned} & moveRob[loc[Block]] \rightarrow pick[Block] \rightarrow moveRob[loc[Table]] \\ & \qquad \rightarrow place[Block, Table] \end{aligned}$$

A realistic application would be more complicated than this example, requiring the specification of pre-conditions and state-transition constraints, etc. Robotic applications of symbolic planning normally assume access to a motion generator to perform the actual execution of robot movements commanded by each discrete decision (Garrett *et al.*, 2021). Naturally, the focus of such algorithms becomes the design of a language for describing the planning domain, and the most-used one to date is the planning domain definition language (PDDL) (McDermott, 1991).

For manipulation tasks, the integration of both planning methods is an inevitable route. This gives rise to the *task and motion planning* (TAMP) problem and a series of algorithms. Symbolic task planning methods treat the process of motion generation as a predetermined sequence, focusing primarily on planning the order of motion primitives or a plan skeleton. In contrast, TAMP approaches aim to plan both the sequence of motion primitives and their corresponding parameters, taking into account motion constraints. This planning process is driven by specific objectives, such as minimising trajectory length, to achieve efficient and optimal solutions (Garrett *et al.*, 2021). The problem is commonly formulated as a mixed-integer, constrained optimisation problem, for which there are existing solvers (Ploskas and Sahinidis, 2021). Exact solvers can find the optimum for small and demonstrative tasks, but it is generally impossible for realistic tasks, which tend to have a prohibitive number of solution dimensions and constraints.

For many applications, samplers are required to search for a plan skeleton and its action parameters subject to the constraints. Therefore, they can be classified according to how the search is organised. There are methods that 1) find a plan skeleton first and then determine the action parameters, 2) find action parameters to satisfy constraints first and then assemble valid actions into a skeleton, or 3) interleave between action planning and constraint satisfaction. In practice, information obtained during action parameter optimisation can be saved to accelerate computation by using a dataset to store valid action parameters for some constraints, invalid search regions, impossible constraints, and so on. A thorough review of these methods can be found in (Garrett *et al.*, 2021).

Summary

In sum, TAMP formulates the object manipulation problem into a constrained and mixed integer optimisation problem. Existing methods attempt to search over the hybrid space of discrete and continuous parameters for a plan skeleton of actions and their respective parameters subject to motion constraints (Garrett *et al.*, 2021).

For object manipulation tasks, these methods suffer greatly due to the prohibitive number of dimensions, constraints as well as combinatorial effects among discrete contact events. Another major difficulty comes from the assumption of an available dynamic model of the environment involving manipulator-object and object-object interactions, which are highly sophisticated to manually specify and compute. Therefore, researchers started to develop hierarchical frameworks that do not require a dynamic model. They either combine classic planners and machine learning or completely learn from data.

2.4.2 Learning-based hierarchical control

As introduced in subsection 2.3.3, modern motion generation methods employ a learnt reactive policy based on system observations to produce trajectories instead of planning with a dynamic model. Similar principles also apply to task-level planning. Researchers have been trying to replace human designs with data-driven policies. Integrating learnt policies can happen at the task and motion levels. Researchers have modified TAMP systems by using learnt policies to realise the motion for each action primitive, instead of classic motion generators. The reverse has also been practised: learning a high-level planner for action primitives generated by classic motion planners. To the extreme, researchers have also developed fully-learnt hierarchical control systems.

The following will discuss three aspects of these works and their relation to the contributions: 1) skill policy design and learning, 2) learning to plan, and 3) skill adaptation and simultaneous training.

Skill design and learning

A large number of works were devoted to the design and learning of skill policies, which are essential to hierarchical control systems. It is suggested by many works that in such systems the manipulation skills ought to be flexible, transferable and composable. In terms of learning such skills, the techniques employed tend to be RL and IL with human demonstrations (Yang *et al.*, 2020; Strudel *et al.*, 2020; Schwenkel *et al.*, 2020; Rao *et al.*, 2022; Mu *et al.*, 2021). A central question in this research area is how to define skill policies. The most obvious choice is to specify a discrete set of skills, each of which is trained towards its own subtask (Jiang *et al.*, 2019; Strudel *et al.*, 2020; Schwenkel *et al.*, 2020; Rao *et al.*, 2022). The skills are normally associated with some task goals, such as the target pose of a reaching skill or the target height of a lifting skill. Therefore, they are normally conditioned on selectable parameters, such as goals or motion constraints, making them more adaptable and flexible for task planning. Image pixel is also a popular choice to parameterise manipulation skills, such as grasping or pushing (Li *et al.*, 2018; Chitnis *et al.*, 2020; Zeng *et al.*, 2018, 2020).

Another way to learn skills attempts to rely even less on human priors, allowing skills to emerge during training. These works are known as skill discovery or option discovery, often conducted in the HRL framework called *option framework*. They seek to optimise skill policies as well as a planning policy with respect to a single long-horizon task, relying on probabilistic or information theory to encourage the emergence of distinct skills (Krishnan *et al.*, 2017; Zhang and Whiteson, 2019; Khetarpal *et al.*, 2020b). If the options are goal-conditioned policies, the focus is then put on the discovery of subgoals (Jiang *et al.*, 2019; Pateria *et al.*, 2021a; Zhu *et al.*, 2022; Cho *et al.*, 2022). An obvious benefit of employing goal-conditioned RL is the sharing of knowledge between subtasks within a single low-level policy. Also, goal-conditioned policies can learn efficiently with the help of goal relabelling techniques (Andrychowicz *et al.*, 2017; Fang *et al.*, 2019b). However, it is generally agreed to be too difficult to learn, from a single task reward signal, a set of distinct and interpretable low-level policies as well as a high-level planner.

Learning to plan

Many works focus on learning to plan with or compose a given set of manipulation skills, regardless of hand-engineered, motion-planner-generated or data-driven skills. The planning problem needs to be represented and solved differently according to how the skills are formulated.

With a set of fixed skills, the learning-to-plan problem can be formulated as an RL problem with a discrete action space corresponding to the set of skills, such as the option framework (Sutton *et al.*, 1999b; Barreto *et al.*, 2019; Shah *et al.*, 2022). This idea has been explored by many researchers inside and outside the robotic community (Tessler *et al.*, 2017; Strudel *et al.*, 2020; Pateria *et al.*, 2021b).

Recently, researchers started investing more in learning to select a skill as well as specify its parameters. For example, Zeng *et al.* (2022) developed a framework for evaluating grasping success rates of different grasping skills under the same image observation. A new learning framework, named *parameterised action reinforcement learning*, has been proposed to study such hybrid skill planning for long-horizon robotic control (Wei *et al.*, 2018; Dalal *et al.*, 2021). Another interesting solution is to use goal-conditioned policies as skills and learn a high-level planner to select or generate subgoals (Jiang *et al.*, 2019; Pateria *et al.*, 2021a; Zhu *et al.*, 2022).

Another research direction attempts to compose the skill policies. Instead of letting the skill policy produce actions, researchers proposed to learn a high-level policy that weights the actions or the network parameters of the skill policies (Peng *et al.*, 2019; Yang *et al.*, 2020). These may not be categorised as task-level planning methods, but they contribute to how skill policies may be utilised.

Finally, an ambitious direction seeks to learn a temporally extended dynamic model of the world corresponding to temporally extended actions, i.e., skills (Xu *et al.*, 2021; Khetarpal *et al.*, 2021). This is tightly related to the study of model-based RL (Moerland *et al.*, 2020). With such a model, classic planning algorithms will be able to work on complex manipulation tasks. However, the idea is fairly new and demands more theoretic development as well as practical experiments on robotics.

Skill adaptation and simultaneous training

There are important and unique issues for learning-based hierarchical control systems. The two most intriguing ones are 1) the adaptation or transfer of pre-trained skills to various tasks and 2) the simultaneous training of the skills and the planner.

A large body of work separates the training of both levels. They assume the pre-trained skills suffice for the target task without the need for fine-tuning. Such works are good enough to demonstrate the effectiveness of using temporal abstracted skills, but not so to handle real-world tasks. Therefore, researchers have sought to fine-tune the skill policies for different tasks (Li *et al.*, 2019) or even incorporate new ones (Holas and Farkaš, 2020). Another solution is to learn both levels simultaneously. This idea has the benefit of keeping both levels in line with the same task objective but induces a training non-stationarity issue on the high-level policy (Nachum *et al.*, 2018; Levy *et al.*, 2019). This is because the high-level policy is learning with a set of skill policies that are non-optimal and randomly exploratory. In other words, the system transition for the high-level policy is changing over time.

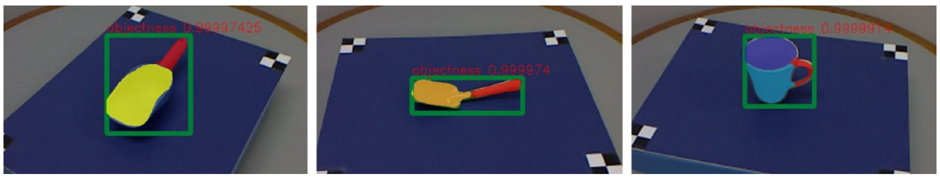
Summary

Learning-based hierarchical control systems are arguably the most promising direction for future manipulation systems. Various design choices have been explored but there is more to be done (Pateria *et al.*, 2021b). The benefit is the potential of 1) realising sophisticated manipulation skills that are difficult to hand-craft or plan, and 2) generalising onto tasks with complex physical dynamics that are difficult to manually specify. On the other hand, the difficulties include but are not limited to: 1) the design or learning of a transferable skill representation/parameterisation, 2) the adaptation of skills, and 3) the non-stationarity of planning over a set of changing skills.

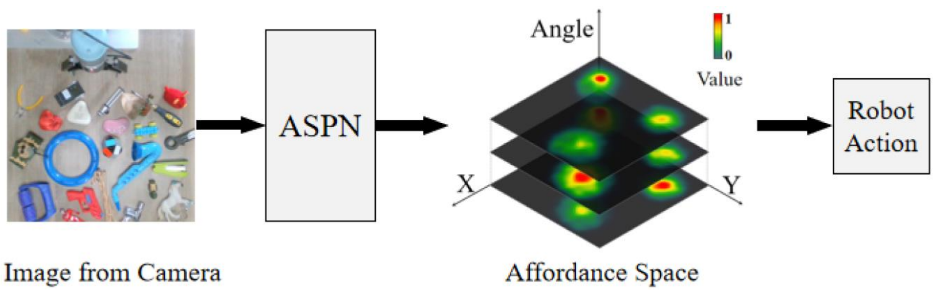
In response, the second contribution (chapter 5, publication 1) of this thesis proposes a new HRL framework that learns multiple final task outcomes by reusing a single goal-conditioned skill policy and proposes to use the first contributive method (chapter 4, publication 3) to accelerate learning and alleviate the non-stationarity problem.

2.4.3 Manipulation affordance learning

In recent years, there is an increasing number of works on the integration of the concept of affordance and robotic manipulation research (Ardón *et al.*, 2021). The conception of affordance regarding object manipulation refers to what can be done with an object and what will happen to the object for an agent (Gibson and Collins, 1982). For example, the handle of a knife affords to grasp by a robotic gripper while the edge of the knife affords to cut. Robotic researchers are most interested in the learning and application of manipulation skill affordances, closely related to skill learning and hierarchical control.



(a) Segmented image from (Chu *et al.*, 2019). Red parts afford to grasp, yellow parts afford to scoop, orange parts afford to support, deep blue parts afford to contain, and blue parts afford wrap-grasping.



(b) Grasping success score prediction Wu *et al.* (2020).

Figure 2.6: Affordance prediction examples

A large body of works proposes to predict what skills are afforded on which part of an object, represented as binary masks, e.g., Figure 2.6a (Do *et al.*, 2018; Chu *et al.*, 2019; Mandikal and Grauman, 2021b; Hämäläinen *et al.*, 2019). A typical limitation of these works is that they provide only binary prediction, while in the real world, different locations of the graspable region of an object will probably provide different success rates. Therefore, another set of works proposes to predict how probable an action is afforded at each pixel location, e.g., Figure 2.6b (Cai *et al.*, 2019; Wu *et al.*, 2020; Yang *et al.*, 2021).

There are different limitations to be further improved. First of all, the affordance studied in these works is mostly task-agnostic graspability. This relates closely to subsection 2.2.3 where task-agnostic grasp generation was discussed. There is a lack of research on object affordance beyond grasping. Secondly, although image segmentation methods provide binary masks for affordance beyond grasping, they do not provide the specific changes or results that the action could cause to the world. For example, pouring water onto the centre of the cup rim has a smaller chance of spilling out water compared to pouring onto a place near the rim edge.

The prediction of the results of skills is closely related to dynamic model learning. An affordance theory has been formulated using the RL framework and extended to temporally extended actions (Moerland *et al.*, 2020; Khetarpal *et al.*, 2021). However, the affordance of skills may not be limited to the form of changes in the observation or state. For example, it can be some important metrics of a particular manipulation task, such as the amount of water that can be poured into the cup in the previous example. In section 6.2, a formal description of the RL-based affordance theory will be reviewed and then extended to *general affordance* beyond the prediction of the resultant system state. In chapter 6, a concrete manipulation study based on the proposed *general affordance* will be conducted, demonstrating its benefits on a difficult contact-rich object disentangling manipulation task.

2.4.4 Summary

This section reviews related methods that develop hierarchical control systems to perform long-horizon manipulation tasks. Typically these frameworks are composed of two levels of hierarchies, corresponding to the task planning and motion generation levels of the robotic manipulator control framework introduced in subsection 2.3.1. The literature discussed ranges from the classic *task and motion planning* framework to the fully data-driven *hierarchical reinforcement learning* framework. In summary, the main benefits of using a learnt hierarchical control system include:

- Similar to the hierarchical structure of the brain, which is also learnable.
- The potential of solving complex long-horizon manipulation tasks with the help of flexible and adaptive policies, without the need for a computable dynamic model.
- Avoid or alleviate the exploration difficulty of long-horizon tasks faced by non-hierarchical control algorithms.
- Improved reusability of the skill policies.

However, there are challenges ahead in the field. In response, this thesis makes two contributions to the field of hierarchical robotic control systems, especially for object manipulation applications. The first contribution (chapter 5, publication 1) proposes a new HRL framework and uses the first contributive method to accelerate its training. The last contribution (chapter 6, publication 5) focuses on the extension of the RL-based affordance theory and its application on an object disentangling manipulation task.

2.5 Robotic simulations

In this final section of the literature review, we will discuss another significant aspect of robot learning methods: simulations. It is common knowledge that computer simulation greatly reduces the cost of the validation process of robot behaviours by avoiding tests on real platforms (Staranowicz and Mariottini, 2011). In addition, with the need of generating learning data, the use of robot simulators is becoming increasingly inevitable in recent years

(Collins *et al.*, 2021). This also brings a new requirement to be considered in selecting or developing a robot learning simulator: the fine integration with Python packages for DL and RL such as PyTorch, OpenAIGym, Tensorflow, Stablebaseline, etc. (Zhao *et al.*, 2020; Collins *et al.*, 2021). Moreover, due to the inaccuracy of the physics model in the simulation, deploying methods that are validated in simulations onto the real world mostly requires some form of adaptation or fine-tuning. Improving the performance of simulation-to-real policy transfer is also an important topic of robot learning (Salvato *et al.*, 2021). Since this PhD project also relies heavily on computer simulation, this section will discuss a few popular robot simulators for manipulation and some key articles regarding sim-to-real policy transfer.

2.5.1 Simulators

The development of computer simulation software started during World War II initially for assessing nuclear detonation. It rapidly demonstrated its value in various areas, including gaming, construction, manufacture, design, robotics, etc., as supported by the ever-growing computation power. For robot simulations, it is commonly required to integrate most of the following: a graphic user interface (GUI), a physic engine, various sensors, the import function for Universal Robot Description Format (URDF) or other similar format files, inverse kinematics computation, motion planning algorithms, collision detection algorithms, realistic rendering (Staranowicz and Mariottini, 2011; Collins *et al.*, 2021). In recent years, the need for fast computation has become greater due to the increasing demand for data for learning-based methods. This leads researchers to seek and develop simulators that support parallel computing and GPU accelerated computation (Collins *et al.*, 2021). Based on the needs of this thesis, four robot simulators are researched and tested during the project. The following will provide a brief introduction to these simulators and the readers are referred to (Zhao *et al.*, 2020; Staranowicz and Mariottini, 2011; Collins *et al.*, 2021) for more detailed surveys on robot simulators.

Gazebo

The classic and widely used Gazebo simulator has been integrated into many robot applications (Koenig and Howard, 2004; Staranowicz and Mariottini, 2011; Collins *et al.*, 2021). Although Gazebo is more used in ground robot applications including legged or mobile, it does support manipulation simulations. One advantage of Gazebo is that it well supports the import function of URDF files and the integration with the robot operating system (ROS), which is a very widely used communication system for real and simulated robots (Quigley *et al.*, 2009). The support of ROS provides users with easy access to various stable inverse kinematics computation and motion planning packages such as MoveIt! (Chitta *et al.*, 2012). In recent years, some researchers have attempted to perform learning-based applications using Gazebo (Zamora *et al.*, 2016; Borrego *et al.*, 2018; Lopez *et al.*, 2019), especially to enable the integration with the OpenAIGym package which is the top programming toolkit for reinforcement learning practitioners (Brockman *et al.*, 2016). However, Gazebo is less preferred for generating training data because mainly of its slow computation of complex dynamics and its unintuitive integration with the modern programming style of deep learning-based methods that rely heavily on Python (Collins *et al.*, 2021).

CoppeliaSim

CoppeliaSim (previously named V-Rep) is another classic and stable choice of robot simulator (Rohmer *et al.*, 2013). It has good support for the key features required by manipulation simulations including the URDF import function, inverse kinematics, motion planning, and various sensors. One advantage of CoppeliaSim is that it provides the user with a GUI to create and modify the simulation scene in much detail. It also provides more realistic rendering and sensor models compared to Gazebo. Although it was not originally developed with the consideration of supporting learning-based applications, a package named PyRep was introduced later to enable easy integration with learning-based applications and OpenAIGym-style programming (James *et al.*, 2019). This leads to a number of manipulation benchmarks and applications with learning-oriented APIs such as (James *et al.*, 2020; Stepputtis *et al.*, 2020; Zheng *et al.*, 2022; Shridhar *et al.*, 2023). However, despite its realistic dynamics and rendering for rigid object manipulations, CoppeliaSim is still less preferred for training data generation because again of its relatively slow computation speed and unintuitive integration with learning-based applications (Collins *et al.*, 2021).

PyBullet

PyBullet is another important simulator that is widely used in the robotic manipulation community due to its open-source nature, Python-based APIs and the support of deformable object modelling (Coumans and Bai, 2016; Collins *et al.*, 2021). It has been used in many learning-based applications because it was designed in consideration of reinforcement learning applications. Compared to Gazebo and CoppeliaSim, it is more difficult to build complex scenes in PyBullet but easier to develop learning-based methods and faster in computation and training speeds with OpenAIGym-style programming APIs (Zhao *et al.*, 2020). However, because PyBullet is open-sourced, many academic researchers choose it for robot learning experiments. The first project presented in chapter 4 also relies on the PyBullet simulator.

Mujoco

Lastly, the Mujoco simulator is arguably the most popular simulator for robot learning applications due to its fast computation speed, stable contact computation and fine integration with the OpenAIGym package (Todorov *et al.*, 2012; Zhao *et al.*, 2020; Collins *et al.*, 2021). Mujoco well supports the required features as a robot simulator except for inverse kinematics and path planning functionalities. However, because of its accurate contact computation, it has been used in many contact-rich tasks including the Rubiks cube recovering task, grasping, humanoid modelling (Agostinelli *et al.*, 2019; Ivaldi *et al.*, 2014; Zheng *et al.*, 2022). Another important advantage of Mujoco is the easy integration with learning-based applications, which makes it a default simulator for benchmarking and evaluating learning algorithms (Zhu *et al.*, 2020; Chen *et al.*, 2020; Collins *et al.*, 2021). One of the shortcomings could be the unintuitive and cumbersome process of building a scene and adjusting its parameters as Mujoco does not provide a GUI, much similar to the PyBullet simulator. However, because of its well-maintained learning-oriented programming APIs and fast computation of contact-rich dynamics, the first and third projects (chapter 5 and 6) in this PhD also rely on Mujoco to validate the proposed manipulation methods.

2.5.2 Sim-to-real policy transfer

Deploying the robot behaviours validated in simulation onto real-world platforms is one of the last steps of almost all robotic research projects. As simulators are becoming so much more powerful in recent years that almost all robotic applications are developed in simulation first, closing the gap between simulation and the real world has never been so interesting to researchers in many disciplines. Although the methods proposed in this thesis are not yet validated in the real world, they are developed in consideration of real-world deployment that can be achieved in the future. The following will briefly introduce the key problem and the main techniques that are adopted by the robot learning community.

Key problems and the main solutions

Essentially, there are three main sources of mismatches between the simulation and the real world. Firstly, the dynamic behaviours and interactions among objects are simplified in simulations. Although this may not be of big concern for applications that only involve rigid objects in structured environments, it could present serious mismatches in the physical behaviours in scenarios with deformable objects, contact-rich manipulation, unstructured scenes and unknown object properties (Billard and Kragic, 2019; Horak and Trinkle, 2019; Pfrommer *et al.*, 2021; Lee *et al.*, 2022).

Secondly, the mismatch of accessibility and quality of sensory observations between simulation and the real world (Salvato *et al.*, 2021; Muratore *et al.*, 2022). For example, most RL applications assume access to the Cartesian coordinates of objects which is of no concern in simulation but normally not true in real-world scenarios (Petrík *et al.*, 2021). Even for tasks where the needed sensory observations are obtainable in the real world, there is a difference between the simulated and real readings. For example, object states are estimated in the real world by a stochastic process that tends to induce noise while they are exactly accurate in simulations (Jin *et al.*, 2021). Also, the images or point clouds rendered in the simulations are vastly different from those captured in the real world (Chebotar *et al.*, 2019).

In order to deal with the mismatch of dynamics and sensory observations, there are two mainstream methodologies adopted by the community. The first kind of approach seeks to increase the diversity of training data in the simulation such that the training data distribution may cover the target data distribution in the final real-world deployment. A simple solution is to inject noise into the observations or dynamics of the simulation (Andrychowicz *et al.*, 2017; Peng *et al.*, 2018; Yu *et al.*, 2019). Many methods apply randomisation to various aspects and parameters of the simulation, such as various visual effects (Alghonaim and Johns, 2021), physics parameters (Peng *et al.*, 2018; Exarchos *et al.*, 2021), robot kinematics (Exarchos *et al.*, 2021), and many others (Zhao *et al.*, 2020; Salvato *et al.*, 2021). Another resort is to train an adversarial agent that optimises for the diversity of the training environment such that the learning agent strives to avoid local minima, overfitting and adapt to the real tasks (Zhang *et al.*, 2019; Hamaya *et al.*, 2021; Lechner *et al.*, 2021, 2023). Instead of enriching the training data distribution, another set of methods seeks to fine-tune the policy in the real world after simulation training is finished (Salvato *et al.*, 2021; Ibarz *et al.*, 2021; Smith *et al.*, 2022).

Lastly, there is a lack of consideration in the simulation regarding safety constraints that are vital in real-world robotic applications, especially those that potentially will involve human activities. As a convention for robot learning applications, especially those using RL, practitioners program the robot to explore the environment with random actions to collect a large amount of training data. This was preferred as it helps to remove human priors and encourage emerging behaviours, however, it tends to slow down the learning process for large solution spaces and fails to enforce safe robot behaviours when deployed in the real world. In recent years, the research community has started again to use more human priors to accelerate learning, encourage desired behaviours and ensure safe exploration (Liu *et al.*, 2020; Pertsch *et al.*, 2021; Jauhri *et al.*, 2022; Wang *et al.*, 2022; Agrawal, 2022; Brunke *et al.*, 2022). This direction demands the careful design of the constraints in motion trajectory generation, the explorable region of the environment, the mathematical and computable representations of safety metrics and the integration of multiple safety-related objectives with the task completion objective (Billard and Kragic, 2019; Kroemer *et al.*, 2021; Lechner *et al.*, 2021; Brunke *et al.*, 2022).

Summary

This subsection very quickly covers the key topics and solutions in the field of sim-to-real policy transfer. This topic is not the focus of the research presented in this thesis but more of a related future direction. The main purpose is to provide a background of the possible actions that could and need to be taken in the future development of this PhD. The readers are referred to the mentioned review articles for detailed information about sim-to-real policy transfer (Zhao *et al.*, 2020; Salvato *et al.*, 2021; Muratore *et al.*, 2022).

2.5.3 Summary

To sum up, the last section of this chapter first briefly introduces a number of popular robot simulators at the time of carrying out these PhD projects. With PyBullet and Mujoco being the two fastest simulators, this PhD relies on them to perform training and evaluation for the proposed learning algorithms. Physics simulation for this PhD is merely a tool for training and evaluation rather than a research topic. However, it is nevertheless of vital importance to discuss the possibility and methods required for deploying the simulation-based policy in the real world. The possible actions that need to be taken in future research will then be covered in the conclusion chapter.

Chapter 3

Preliminary

3.1 Standard Reinforcement Learning

The reinforcement learning (RL) paradigm (Figure 3.1) was inspired by the observation that humans or some animals can learn certain behaviours associated with some form of reward signals. Such as a dog learns to shake hands to earn more food. The entity that learns behaviours in RL is named an (RL) agent. In practice, this could for example be a robotic arm.

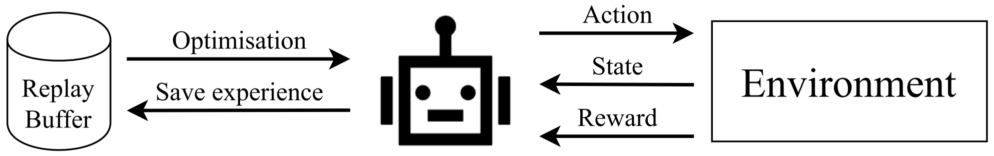


Figure 3.1: The reinforcement learning paradigm. In RL, an agent (in the middle) acts on the environment which feeds back with the next state and a reward. The agent collects and uses interaction experiences to serve its purpose of optimising some tasks. Modern algorithms commonly use a buffer to store and sample collected data for training deep neural networks.

In the process of learning, the agent needs to explore its local environment by taking actions, so that it can discover the consequences of its actions. Modern RL methods will store the experiences about the agent's actions and the consequences in a buffer, and use them to perform learning. The RL agent is expected to exploit what it has learnt about its actions to earn more rewards, and in turn, to increase its confidence in taking rewarding actions. Thus, we have the term "*reinforcement*" learning (Sutton and Barto, 2018). The mathematical models, which represent and store what the RL agent has learned, were once finite dimensional matrices (Watkins and Dayan, 1992) and parameterised distributions (Engel *et al.*, 2005). With the rise of DL techniques, they are now replaced by deep neural networks. Thus, the term "*deep reinforcement learning*" (DRL) (Mnih *et al.*, 2015).

In order to deduce algorithms, a mathematical framework has to be defined for RL problems. The first subsection will introduce such a framework, the Markov Decision Process (MDP). Based on the definition of MDP, the second subsection will introduce the foundation of RL algorithms (before DL) as well as important terminologies. Thereafter, the next three subsections will be devoted to the fundamentals of three kinds of model-free RL algorithms. This section is to build up the foundation for understanding modern DRL algorithms introduced in the following sections.

3.1.1 Markov decision process

The formal mathematical framework of an RL agent interacting with the environment through actions is the MDP. An MDP features a sequential decision-making process, where an agent takes actions according to the feedback of the environment, i.e., states and rewards, in a sequential fashion. Being sequential means that the interaction between the agent and the environment can be represented by a list of action and state pairs in a sequential order such as timesteps (Figure 3.2).

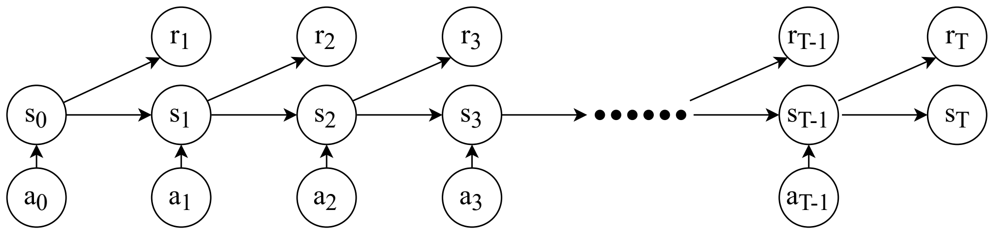


Figure 3.2: An interaction sequence of an episode of a finite-horizon MDP

There are variations of MDPs in terms of observability, time horizon, discounting, etc. In particular, the set of RL problems that this thesis studies will be modelled as discrete time, finite-horizon, fully observable and discounted MDPs. Here, *finite-horizon* means the agent can only take a limited number of actions within each interaction cycle or a so-called episode. In each episode, the sequential list of states and actions is called an episodic trajectory. Being *fully-observable* means that the agent can obtain feedback from the environment at the physical level, such as the 6D pose of an object. Lastly, a *discounted* MDP calculates the sum of the rewards at each interaction step with an exponentially multiplied weight.

Formally, a MDP is represented by a tuple, $(\mathcal{S}, \mathcal{A}, p, p_0, r, \gamma)$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is the dynamic transition distribution, $p_0 : \mathcal{S} \rightarrow [0, 1]$ is the initial state distribution, $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is the reward function and $\gamma \in [0, 1]$ is the discount factor.

The interaction process starts with an initial state $s_0 \in \mathcal{S}$ sampled from $p_0(s)$. At each timestep t, the RL agent selects an action $a_t \in \mathcal{A}$. The state of the world, s_t, then changes to s_{t+1} according to the transition probability distribution $p(s_{t+1}|s_t, a_t)$. A reward can be computed from the reward function $r_{t+1} = r(s_t, a_t)$. The agent is tasked to take actions that maximise the discounted return (cumulative rewards) $G_t = \sum_{t=0}^{T} \gamma^t r_{t+1}$ where T is the length of an episodic trajectory. The closer the discount factor γ is to the value of 1, the more important future rewards are to the agent.

Markov property: An important assumption for the kind of problems that most RL algorithms are addressing, including the works of this thesis, is the so-called *Markov property* (Puterman, 2014). Put simply, if the future states of a system are independent of what happened before the current state, then such a system is *Markovian*. For example, the future trajectory of a flying football can be fully computed by the current position and velocity, regardless of how it ended up in the current situation (past history). In other words, the information captured within a state of the system is enough to determine or predict the future of the system. This property is important because the three basic components (policy, state and action value functions) of RL algorithms are based on only the current state. We will see, in the next subsection as we define the three components, that RL algorithms assume that the current state is informative enough for taking actions and estimating future return (Sutton and Barto, 2018).

3.1.2 RL algorithm foundations

An RL algorithm drives the RL agent towards its task of maximising future return. While DRL has seen great advancement in recent years tackling difficult problems that are unthinkable to solve in the last century, the fundamental principles of RL remain the same as what was developed in the last century (Sutton and Barto, 2018). At the core of RL algorithms, there are three functions that may be used to store the knowledge of the world, the task and what actions to take: the policy π, the state value function $v^{\pi}(s)$, and the action value (q) function $q^{\pi}(s, a)$.

The policy, π, is a mapping from the perceived state to actions. It can be represented as a stochastic distribution of actions given a state $\pi(a|s) : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$, or a deterministic function that maps a state to an action $a = \pi(s) : \mathcal{S} \rightarrow \mathcal{A}$. The agent can query its policy to obtain an action. Formally, the agent's task is to learn (or find) an optimal policy, π_*, such that its future return, G, is maximised. **The state value function**, $v^{\pi}(s)$, is defined as the expectation of the future return that the agent can collect when it starts at state $S_t = s$ and selects actions thereafter according to some policy π. Similarly, **the action value function**, $q^{\pi}(s, a)$, is also the expected future return, but this time, when the agent starts with an action

$A_t = a$ at state $S_t = s$. For any $t \in \{0, 1, ..., T-1\}$, they can be written down as:

$$v^{\pi}(s) = \mathbb{E}_{\pi}\left[G_t \mid S_t = s\right] \tag{3.1}$$

$$q^{\pi}(s, a) = \mathbb{E}_{\pi}\left[G_t \mid S_t = s, A_t = a\right] \tag{3.2}$$

where, S_t and A_t represents the real sampled state and action at timestep t. Notice that the value functions are bounded to a policy (subscription symbols), meaning that different policies will result in different values. Naturally, the optimal value functions are bounded to the optimal policy, and vice versa. Now, the key question that any RL algorithm seeks to answer is: *how to find the optimal policy π_* that maximises the future return*? The cost function to be maximised can be formally written as:

$$J(\pi) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{T} \gamma^t r_t(s_t, a_t)\right] \tag{3.3}$$

There are a few pathways that different RL algorithms take to approach this question, and they are normally classified as shown in Figure. 3.3. The major branching lies in whether a dynamic model of the environment is required by the algorithm, thus the terms "model-based" and "model-free". The following texts will touch briefly on model-based methods, but more on model-free methods. Because the algorithms used or developed in this thesis are all within the model-free class.

Model-based algorithms have access to an "approximate" model of the dynamics of the world. Fundamentally, they can compute the resultant state of an action being applied to the current state through the dynamic model (transition function) of the MDP, $p(s_{t+1}|s_t, a_t)$. This can be leveraged to simulate interaction trajectories. These trajectories may be directly used as references to select actions to act on the real environment, in which case it

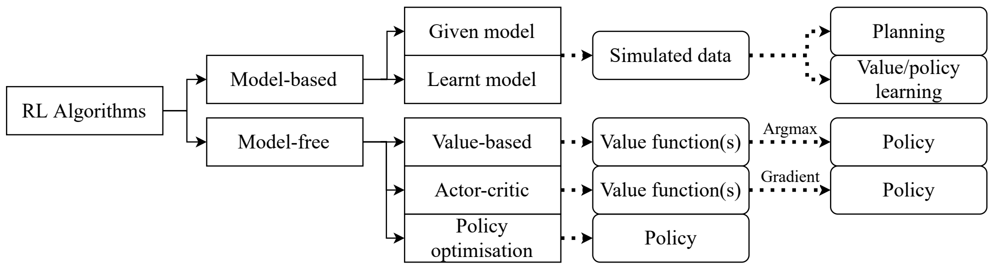


Figure 3.3: The taxonomy of RL algorithms

is generally referred to as **planning** with a dynamic model (Sutton, 1991). The policy is directly generated by the planning processes with the dynamic model. On the other hand, simulated trajectories may be used to estimate the value functions. These values are then used to deduce or optimise a policy (Gu *et al.*, 2016). The procedures that deduce or optimise a policy from the value functions are typically the same as what happens in model-free methods, which we will introduce shortly. The only difference here is the source of the trajectories is synthesis or "imagination". Based on whether the dynamic model is given or learnt from data, these algorithms are further categorised. However, their core is the same, that is to utilise a mathematically computable dynamic model to generate action plans or to optimise the value functions or the policy. A thorough review of model-based RL was conducted by Moerland *et al.* (2020) and recommended for interested readers.

Model-free methods, on the contrary, answer the question without access to a model. In other words, they can only learn from the experiences collected during the interaction with the environment. These methods are categorised into value-based and policy gradient methods. To the extremes, value-based methods seek to find the optimal q-function and extract the optimal policy from them, while the purest policy gradient methods use experiences to optimise the policy only. The most classic value-based method shall be the Q-learning method (Watkins and Dayan, 1992), while for policy gradient, the REINFORCE method (Williams, 1992). Note that at the extreme, they never compute either the value functions or the policy. However, in between the choice of using the value functions or the policy, there are methods that optimise both of them. More specifically, these methods use the learnt value functions to compute the optimisation objective for the policy, and thus are named Actor-critic methods (Konda and Tsitsiklis, 1999).

In the following subsections, the foundations of the three kinds of model-free algorithms will be introduced as stepping-stones to help understand the modern, DL-aided methods.

Exploration: Before we get into how the three kinds of methods work, it is necessary to introduce the *exploration vs. exploitation* problem. Let's say the agent is greedy and it *always* takes the action that has the largest one-step reward. While at the start, it knows nothing about the system, so it randomly chooses an action which gives a reward of 1. Now because the agent is one-step-greedy so it will always take that very first action to maximise its one-step reward. The problem with the one-step-greedy agent is that it does not *explore* and always *exploit*. Here comes one of the most important trade-offs in RL methods: *exploration vs. exploitation*. By exploration, the agent will take actions to discover unseen system states, rewards and action consequences. By exploitation, the agent will maximise its future return. Therefore, for any RL algorithm to find the optimal solution, an exploration strategy is required during learning. We will discuss more exploration strategies for each specific algorithm in later sections.

3.1.3 Value-based method

We will start with value-based methods which seek to compute the optimal values as they are the foundation of other methods. From here, we use Q_m to denote the q values in a matrix, called Q table, at the m-th optimisation iteration. There are two distinct ways to estimate the values: Monte Carlo (MC) sampling[1] and Temporal-Difference (TD) learning[2].

Monte-Carlo method

In simple terms, MC methods repeat the process of collecting a full trajectory of state-action pairs, evaluating the *expected* returns of every state-action pair. When the task is simple enough, MC methods would provide an efficient estimation of the values. MC methods iterate in a cycle of collecting a trajectory and using the trajectory to update the q table. After a sufficient number of episodes, the q value should converge to the optimal case. However, theoretically speaking, the q value only converges when each of the state-action pairs is visited an infinite number of times. In practice, different tasks will take different numbers of episodes to move the q table close enough to the optimum. For detailed discussions and advanced MC methods, readers are recommended to read chapter 5 of (Sutton and Barto, 2018).

There is an obvious limitation of MC methods. When the system has a large state space and long task horizon, the estimate provided by MC methods can have high variance and the agent may wait too long in between each learning update of the value functions. By "high variance", it generally refers to the possible q values that a state-action pair may have. If there is a large number of future state-action combinations plus a long task horizon,

[1]Chapter 5 in (Sutton and Barto, 2018)
[2]Chapter 6 in (Sutton and Barto, 2018)

there will be many possible q values. So there comes the rise of temporal-difference (TD) methods, which can update the agent's value functions on every single transition it collects, with a smaller variance and without the need to wait for a full trajectory (Tesauro *et al.*, 1995).

Temporal-difference method: Sarsa

Before we explain TD methods, it is necessary to introduce the recursive form of the q function. For a sampled transition tuple, $\{S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}\}$, and any policy π, the following recursive property of the q function holds:

$$\begin{aligned} q^\pi(s,a) &= \mathbb{E}_\pi\left[G_t \mid S_t = s, A_t = a\right] \qquad (3.4)\\ &= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k\, R_{t+k+1} \,\middle|\, S_t = s, A_t = a\right]\\ &= \mathbb{E}_\pi\left[R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k\, R_{t+k+2} \,\middle|\, S_t = s, A_t = a\right]\\ &= \mathbb{E}_\pi\left[R_{t+1} + \gamma\, q^\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a\right] \qquad (3.5) \end{aligned}$$

Notice that, the MC agent collects full trajectory and calculates the full return to update its q table, which in effect is calculating the target q value using Eq. 3.4. While TD methods update at every new transition with a target q value computed from Eq. 3.5:

$$\begin{aligned} Q_1(s,a) &= Q_0(s,a) + \alpha\, [\hat{Q}(s,a) - Q_0(s,a)]\\ &= Q_0(s,a) + \alpha\, [r + \gamma Q_0(s',a') - Q_0(s,a)] \qquad (3.6) \end{aligned}$$

where, $r+\gamma Q_0(s',a')-Q_0(s,a)$ is the temporal-difference (TD) error, denoted by δ_t when the equation is time-subscripted. More specifically, Eq. 3.6 is called the Sarsa method, which updates the q value at every single collected transition after each timestep. In problems with larger state and action spaces as well as longer horizons, TD methods have the benefit of converging faster over MC methods. However, these two particular methods introduced here share one common property: they both learn from some collected data and then discard them. From the examples above, they discard either a whole trajectory or each of the transitions after an update is made to the q table. This characteristic is called *on-policy* learning.

On-policy vs. off-policy. The terms here generally refer to whether the policy that generates the data is the same one that is to be updated using the data. As we can see, the MC and TD methods above are both *on-policy*. On-policy methods are considered simpler to understand and faster to converge. However, they do not converge to the optimal but a near-optimal and still exploring policy. Being on-policy means that the collected data is not reusable, even if they do contain useful information. Therefore, there are two reasons why *off-policy* learning is preferred. First, the learning of the optimal policy can be separated from the one that explores the environment. The policy that is updated is referred to as the *target policy*, while the exploratory one is called the *behavioural policy*. Normally, the target policy is the one that maximises the action value. Secondly, in order to scale up to real-world problems, being *off-policy* means the agent can learn from various sources of data, such as a hand-crafted controller or a human demonstrator. This indeed is one of the essential reasons for the success of many DL-based algorithms. Recently, the extreme case of *off-policy* becomes increasingly popular, featuring the problem of learning fully on an off-line dataset without online exploration (Levine *et al.*, 2020).

Temporal-difference methods: Q-learning

The Q-learning method (Watkins and Dayan, 1992) essentially is the *off-policy* version of Sarsa. However, without discarding used experiences, Q-learning has proved powerful with the use of deep neural networks in high-dimensional domains such as Atari in recent years (Mnih *et al.*, 2015). Simple enough, the update rule of Q-learning can be obtained by changing one operation in Sarsa:

$$
\begin{aligned}
Q_1(s,a) &= Q_0(s,a) + \alpha\,[\hat{Q}(s,a) - Q_0(s,a)] \\
&= Q_0(s,a) + \alpha\,[R_{t+1} + \gamma \max_{a'} Q_0(s',a') - Q_0(s,a)] \qquad (3.7)
\end{aligned}
$$

The only change in the update is that it now takes the maximum q value when computing the estimate of the q target $\hat{q}$, instead of using the action taken in the past. This operation is what liberates the algorithm from *on-policy* data. The proof of convergence confirms that, as long as each state-action pair continues to be visited and the state value is bounded, the q function will converge to its optimal with a probability 1 (Watkins and Dayan, 1992).

The main difference among the three value-based methods can be summarised by Figure. 3.4.

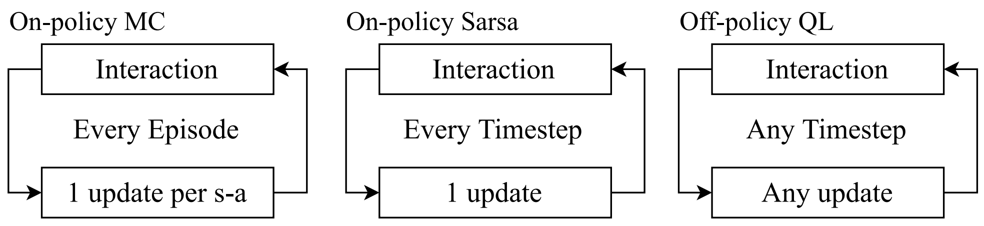


Figure 3.4: A graphic comparison of MC, Sarsa and Q-learning.

Function approximation

So far, we have assumed the q functions to be tabular matrices, which are not likely to suffice for more complex tasks with larger state and action spaces. Therefore, function approximation and gradient-based optimisation are required. In particular, the learning of the value functions is essentially a regression problem, where the *mean-square value error* is used as the minimisation objective function for calculating the gradients:

$$\begin{aligned}\nabla J(\boldsymbol{w}) &= \frac{1}{2}\nabla \sum_s d(s)[\hat{q}(s,a) - \tilde{q}_{\boldsymbol{w}}(s,a)]^2 \\ &= \sum_s d(s)[\hat{q}(s,a) - \tilde{q}_{\boldsymbol{w}}(s,a)]\nabla \tilde{q}_{\boldsymbol{w}}(s,a) \\ &\approx \mathbb{E}_\pi\left[\ [\hat{q}(S_t,A_t) - \tilde{q}_{\boldsymbol{w}}(S_t,A_t)]\nabla \tilde{q}_{\boldsymbol{w}}(S_t,A_t)\ \right] \end{aligned} \tag{3.8}$$

where, $\boldsymbol{w}$ represents the parameters of the value function approximator to be optimised, $\hat{q}$ and $\tilde{q}$ are the true (target) and approximated q values, and $d(s)$ is the weight function or distribution that balances the importance of the error regarding a state. $d(s)$ is typically chosen to be the fraction of time spent in state s or the state distribution induced by the policy distribution. Similar to tabular cases, the true (target) q value $\hat{q}$ needs to be estimated from samples and the parameterised q function. However, the q function in the true q value estimation is to be treated as a scalar value. It is used only for computing the regression target for the objective, with the fixed weight $\boldsymbol{w}_{i-1}$ from the last update. For example, the gradient update at the i-th iteration for Q-learning in the function approximation case can be written as:

$$\boldsymbol{w}_{i+1} = \boldsymbol{w}_i + \alpha\left[r + \gamma \max_{a'} \tilde{q}_{\boldsymbol{w}_{i-1}}(s',a') - \tilde{q}_{\boldsymbol{w}_i}(s,a)\right]\nabla \tilde{q}_{\boldsymbol{w}_i}(s,a) \tag{3.9}$$

Policy extraction

As mentioned, the main reason we focus on learning the q function is that it is straightforward to extract a policy from it. By definition, the action that has the highest q value according to the optimal q function is the optimal action. Therefore, one may extract the optimal policy by $\pi_*(s) = \arg\max_a q^{\pi_*}(s, a)$, where, the *argmax* operation works straightforwardly when there is a fixed number of discrete actions that the agent can perform. This is called the *greedy* policy.

Policy evaluation vs. policy improvement. In model-free methods, as we saw with the value-based methods introduced above, the algorithmic architecture may be unified: they iterate between policy evaluation and improvement. What varies is how they conduct these two steps in particular. In the MC, Sarsa and Q-learning algorithms, interaction samples are used to learn the q value function. This, in essence, is a policy evaluation process: they evaluate the policy that collects those samples in different ways. On the other hand, policy improvement is done by taking the *argmax* of the updated q values.

However, when the action space is continuous and the q-function needs to be represented by function approximators, it typically can only evaluate one state-action pair per input-output. Thus, other techniques are needed to find an optimal action with the highest values (policy improvement), such as the cross-entropy method (Kalashnikov *et al.*, 2018), which may be inefficient to optimise with respect to large state and action space. Therefore, one may resolve to the other extreme of the model-free algorithm, which directly optimises a policy without consulting an explicit action-value function. This kind of method is called policy gradient.

3.1.4 Policy gradient

Different from value-based methods, policy gradient methods seek to directly compute the optimal policy. The term "gradient" indicates that the policy is parameterised and differentiable so that its weights can be updated with the gradients from an optimisation objective function. Here, the policy is denoted explicitly as a parameterised distribution: $\pi_{\boldsymbol{\theta}}(a|s)$, where $\boldsymbol{\theta}$ is the weights of its parameterisation in any chosen form, such as a neural network. For episodic tasks, where the MDP has a finite number of interaction steps, the objective function is normally the state-value function (Sutton and Barto, 2018). In order words, the policy gradients point to the direction where the expected return of the starting states is improved (policy improvement). Denote the maximisation objective function as $J(\boldsymbol{\theta})$, according to the *policy gradient theorem* (Sutton *et al.*, 1999a; Silver *et al.*, 2014; Lillicrap *et al.*, 2015), the gradient with respect to the policy parameters in the episodic case can be written as:

$$\begin{aligned}
\nabla J(\boldsymbol{\theta}) &= \nabla v^{\pi_{\boldsymbol{\theta}}}(s_0) \\
&= \sum_s d^{\pi_{\boldsymbol{\theta}}}(s) \sum_a q^{\pi_{\boldsymbol{\theta}}}(s,a) \nabla \pi_{\boldsymbol{\theta}}(a|s) \qquad (3.10) \\
&= \sum_s \sum_{k=0}^{\infty} \gamma^k Pr(s_0 \to s, k, \pi) \sum_a q^{\pi_{\boldsymbol{\theta}}}(s,a) \nabla \pi_{\boldsymbol{\theta}}(a|s) \\
&= \mathbb{E}_\pi \left[\gamma^t \sum_a q^{\pi_{\boldsymbol{\theta}}}(S_t,a) \nabla \pi_{\boldsymbol{\theta}}(a|S_t) \right] \\
&= \mathbb{E}_\pi \left[\gamma^t \sum_a q^{\pi_{\boldsymbol{\theta}}}(S_t,a) \pi_{\boldsymbol{\theta}}(a|S_t) \frac{\nabla \pi_{\boldsymbol{\theta}}(a|S_t)}{\pi_{\boldsymbol{\theta}}(a|S_t)} \right] \qquad (3.11)
\end{aligned}$$

where $\sum_a q^{\pi_{\boldsymbol{\theta}}}(s,a)\nabla\pi_{\boldsymbol{\theta}}(a|s)$ indicates that the state value represented by the summed action-values weighted by the gradients of the probabilities over all actions, $\sum_s d^{\pi_{\boldsymbol{\theta}}}(s)$ indicates that the quantity is the sum of state-values of visited states weighted by the expected number of timesteps required to reach the state s according to the policy $\pi_{\boldsymbol{\theta}}$. In simple terms, Eq. 3.10 increases the probability of selecting an action if that action results in a high expected discounted future return (the fewer steps required the better), and vice versa.

REINFORCE

The most straightforward way of calculating q^{π} to estimate eq. 3.11 is to approximate it with the actual return collected from an interaction trajectory (policy evaluation). This gives rise to the classic REINFORCE algorithm, which, at its core, is a Monte-Carlo policy gradient method. Explicitly, continuing from eq. 3.11, we have the gradient for the maximisation objective:

$$\begin{aligned}
\nabla J(\boldsymbol{\theta}) &= \mathbb{E}_{\pi}\left[\gamma^{t}\sum_{a} q^{\pi_{\boldsymbol{\theta}}}(S_t, a)\pi_{\boldsymbol{\theta}}(a|S_t)\frac{\nabla\pi_{\boldsymbol{\theta}}(a|S_t)}{\pi_{\boldsymbol{\theta}}(a|S_t)}\right] \\
&= \mathbb{E}_{\pi}\left[\gamma^{t} q^{\pi_{\boldsymbol{\theta}}}(S_t, A_t)\frac{\nabla\pi_{\boldsymbol{\theta}}(A_t|S_t)}{\pi_{\boldsymbol{\theta}}(A_t|S_t)}\right] \text{ (using sample} A_t) \\
&= \mathbb{E}_{\pi}\left[\gamma^{t} G_t\frac{\nabla\pi_{\boldsymbol{\theta}}(A_t|S_t)}{\pi_{\boldsymbol{\theta}}(A_t|S_t)}\right] \text{ (using Monte-Carlo estimate of } q^{\pi_{\boldsymbol{\theta}}}) \\
&= \mathbb{E}_{\pi}\left[\gamma^{t} G_t\nabla\log\pi_{\boldsymbol{\theta}}(A_t|S_t)\right] \text{ (using } \nabla\log x = \frac{\nabla x}{x})
\end{aligned} \tag{3.12}$$

Now provide with a learning rate α, the REINFORCE update rule can be written down for moving the weights of a policy towards the higher return direction with respect to its parameters at iteration i:

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i + \alpha\gamma^{t} G_t \nabla\log\pi_{\boldsymbol{\theta}_i}(A_t|S_t) \tag{3.13}$$

The REINFORCE algorithm when converged provides directly a distribution of the optimal policy, from which the sampled actions shall maximise the expected discounted return. However, similar to the Monte-Carlo value-based method discussed in the last subsection, REINFORCE is an op-policy algorithm that only updates after an episode is finished.

This produces high variance and requires a full trajectory to compute the gradients. The high-variance issue can be alleviated by subtracting the approximated q value with some baseline $b(s)$. This subtraction does not change the expectation of the gradient, but it reduces the variance significantly with the appropriate baseline. The most natural choice of a baseline is the state-value function (Sutton *et al.*, 1999a).

Instead of moving the policy parameters towards the direction of the q value, one moves the policy towards the direction where the q value is *advantageous* over the expected state value. This is yet another way to evaluate a policy. In order words, the probabilities of actions that are expected to collect higher returns over the average state value will be increased, and vice versa (Sutton *et al.*, 1999a). Applying it to the REINFORCE algorithm, we have:

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i + \alpha\gamma^t[G_t - \tilde{v}^{\pi_{\boldsymbol{\theta}_i}}_{\boldsymbol{w}_j}(S_t)]\nabla \log \pi_{\boldsymbol{\theta}_i}(A_t|S_t) \tag{3.14}$$

where $\boldsymbol{w}_j$ is the parameters for the parameterised state-value function at iteration j. This algorithm then updates the state value and the policy together, resulting in reduced variances.

Additionally, there are techniques that allow for faster and more frequent updates in policy gradient methods, similar to how TD learning achieves advantages over Monte Carlo in value-based methods. This leads to the most popular set of RL methods for continuous control problems: the "*actor-critic*" methods.

3.1.5 Actor-critic

The name *actor-critic* (AC) comes from the architecture of the algorithm, where a policy "*acts*" upon the environment to collect data and a value function "*criticises*" the actions taken by the policy to provide the gradients for optimising the policy using the policy gradient theorem, Eq. 3.10. The policy is called the **actor** and the value function is called the **critic**. We only introduce off-policy actor-critic here as it is the foundation of the DDPG and SAC algorithms that are used in this thesis.

Off-policy actor-critic

Remember that in practice, off-policy learning means that the data are not always collected by the *target* policy $\pi_{\boldsymbol{\theta}}(a|s)$ that we would like to evaluate and optimise. The data come from some other policy $\beta_{\boldsymbol{\theta}}(a|s)$.

For policy gradient for the on-policy episodic setting in subsection 3.1.4, the policy is optimised towards the direction of higher future return conditioned on the starting state s_0 (Eq. 3.10). For off-policy AC, the policy is improved at all possible states towards the direction of a parameterised q value function, $\tilde{q}^{\pi_{\boldsymbol{\theta}}}_{\boldsymbol{w}}(s, a)$, under the continuous state distribution of the behavioural policy. Thus, the gradient of the maximisation objective is:

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) &\approx \int_{\mathcal{S}} d^{\beta_{\boldsymbol{\theta}}}(s) \int_{\mathcal{A}} \tilde{q}^{\pi_{\boldsymbol{\theta}}}_{\boldsymbol{w}}(s, a) \nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(a|s) \mathrm{d}s\mathrm{d}a \text{ proof in (Degris } et\ al.\text{, 2012)} \\ &\approx \int_{\mathcal{S}} d^{\beta_{\boldsymbol{\theta}}}(s) \int_{\mathcal{A}} \tilde{q}^{\pi_{\boldsymbol{\theta}}}_{\boldsymbol{w}}(s, a) \pi_{\boldsymbol{\theta}}(a|s) \frac{\nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(a|s)}{\pi_{\boldsymbol{\theta}}(a|s)} \mathrm{d}s\mathrm{d}a \end{aligned} \tag{3.15}$$

The approximated off-policy gradients can be estimated by samples from the behavioural policy, $S_t \sim d^{\beta_{\boldsymbol{\theta}}}(s)$, $A_t \sim \beta_{\boldsymbol{\theta}}(a|S_t)$. Thus we have the gradient estimation and maximisation update rule at iteration i:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \approx \mathbb{E}_{S_t \sim d^{\beta_{\boldsymbol{\theta}}}, A_t \sim \beta_{\boldsymbol{\theta}}} \Big[\rho_t \tilde{q}^{\pi_{\boldsymbol{\theta}}}_{\boldsymbol{w}}(S_t, A_t) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(A_t|S_t) \Big] \tag{3.16}$$

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i + \alpha_\theta \rho_t \tilde{q}^{\pi_{\boldsymbol{\theta}_i}}_{\boldsymbol{w}_j}(S_t, A_t) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}_i}(A_t|S_t) \tag{3.17}$$

where $\rho_t = \frac{\pi_{\boldsymbol{\theta}}(A_t|S_t)}{\beta_{\boldsymbol{\theta}}(A_t|S_t)}$ is the importance-sampling ratio, which is needed because the samples come from a different policy that the gradient is affecting.

Notice the expectation is taken over the distribution of the $\beta_{\boldsymbol{\theta}}$, instead of $\pi_{\boldsymbol{\theta}}$. Despite the source of the data, the improvement direction is clear: increase the probabilities of actions that are evaluated as leading to higher q values. Now we see more clearly from Eq. 3.16 that a good policy improvement depends on an accurate policy evaluation result, i.e., an accurately learnt critic function.

The learning of the critic also needs to go off-policy. Naturally, Q-learning can be directly applied for discrete action spaces, where the maximisation operation is straightforward. However, when the action space is continuous, this becomes problematic as the maximisation operation within each update (Eq. 3.9) becomes too costly. An alternative is to use actions that the actor would take to compute the expected action value for the next state. This is valid because the policy is being optimised to the direction of higher q values (Silver *et al.*, 2014). This results in the following gradient update to the critic:

$$\boldsymbol{w}_{j+1} = \boldsymbol{w}_j + \alpha_w \rho_t \delta_t \nabla \tilde{q}_{\boldsymbol{w}_j}^{\pi_{\boldsymbol{\theta}_i}}(S_t, A_t) \tag{3.18}$$

where, ρ_t is the importance-sampling ratio, and

$$\delta_t = \left[R_{t+1} + \gamma \sum_a \left[\pi_{\boldsymbol{\theta}_i}(S_{t+1}) \tilde{q}_{\boldsymbol{w}_{j-1}}^{\pi_{\boldsymbol{\theta}}}(S_{t+1}, a) \right] - \tilde{q}_{\boldsymbol{w}_j}^{\pi_{\boldsymbol{\theta}}}(S_t, A_t) \right]$$

is the TD error, where the q function from the last update is again treated as a scaler to compute the expected true q value with the weight $\boldsymbol{w}_{j-1}$ being fixed. The off-policy AC algorithm also iterates among collecting samples, evaluating the policy (Eq. 3.18) and improving the policy (Eq. 3.17). One thing to keep in mind is that a proper importance sampling ratio is required as the data come from a policy different from the one being optimised. This broadens the source of learning data, enabling off-policy algorithms to be easily incorporated with various exploratory policies (i.e., data generation strategies).

3.1.6 Summary

In summary, this section has looked into the formulation of RL problems and the mathematical foundations of model-free RL algorithms. Specifically, the key idea behind all these methods is to use data sampled from the interaction with the environment to learn a policy that produces high-return actions. Among them, value-based solutions learn a q function that estimates the q values (expected returns) of state-action pairs. A policy is then extracted from the q values. Policy gradient methods optimise directly a parameterised policy function to produce high-return actions. In between them, actor-critic methods learn a q-value estimator and use it to optimise a policy through gradients. With the brief background, the fundamentals of how RL algorithms approach the return maximisation task have been covered in this section. For more details on RL fundamentals, please refer to (Sutton and Barto, 2018). In the next section, the focus will be on modern solutions that extend these foundations to high-dimensional observation and action spaces, using deep neural networks as function approximators.

3.2 Deep Reinforcement Learning

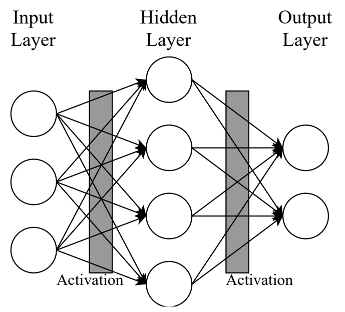
There would be very little doubt nowadays that one of the most important breakthroughs in the early 21st century is the rise of DL and the breathtaking achievements that it has enabled in various fields, including the previously unthinkable applications of RL algorithms on high-dimensional, large state and action spaces, image-based tasks (Lazaridis *et al.*, 2020). The most well-known examples would be the Deep Q Network agent that exceeds human levels on Atari games (Mnih *et al.*, 2015), the AlphaGo agent that defeats the human champion in the game of Go (Silver *et al.*, 2016), the AlphaFold agent that predicts protein structures (Jumper *et al.*, 2021).

Robotic applications of DRL methods are developing much slower than other application areas (Ibarz *et al.*, 2021; Singh *et al.*, 2021), especially when it comes to robotic manipulation (Liu *et al.*, 2021). As discussed in Chapter 2, robotic-specific difficulties, such as unstructured environment, stochasticity, sensor noise, observation redundancy, complex physical contact and interaction, etc., pose new challenges to state-of-the-art DRL methods. Therefore, in order to built-up the foundation to understand what this thesis contributes in this regard, this section will be devoted to the basics of DL and the three most fundamental off-policy DRL algorithms proposed in recent years.

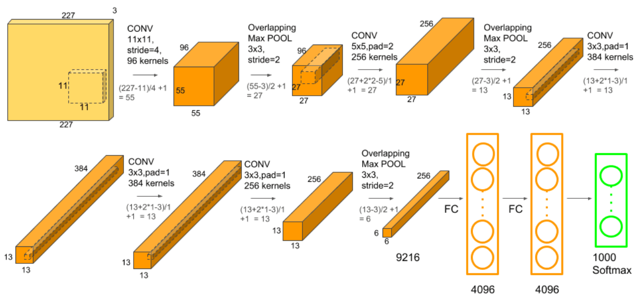
3.2.1 Deep learning basics

The term "deep learning" refers to a set of statistical optimisation algorithms that search for a set of parameter values of a DNN so that the DNN matches a mapping from a distribution to another distribution (Goodfellow *et al.*, 2016). A DNN is essentially a non-linear function whose structure is inspired by how neurons communicate and whose weights can be adjusted. For example, a simple three-layer neural network shown by Figure. 3.5a or a sophisticated one such as the famous AlexNet shown by Figure. 3.5b. Since this thesis focuses on developing and applying DRL methods on robotic-specific problems, detailed and advanced mathematics of DL solutions are excluded. The following will introduce only a few basic notations and terminologies for the sake of the development of DRL methods.

A DNN can be represented as a parameterised function $\boldsymbol{y} = f(\bar{\boldsymbol{x}}; \boldsymbol{\theta})$ where



(a) A simple multi-layer perceptron (MLP) network.


227
3
227
11
11
CONV 11x11, stride=4, 96 kernels
(227-11)/4 +1 = 55
96
55
55
Overlapping Max POOL 3x3, stride=2
(55-3)/2 +1 = 27
96
27
27
CONV 5x5,pad=2 256 kernels
(27+2*2-5)/1 +1 = 27
256
27
27
Overlapping Max POOL 3x3, stride=2
(27-3)/2 +1 = 13
256
13
13
CONV 3x3,pad=1 384 kernels
(13+2*1-3)/1 +1 = 13
384
13
13
CONV 3x3,pad=1 384 kernels
(13+2*1-3)/1 +1 = 13
384
13
13
CONV 3x3,pad=1 256 kernels
(13+2*1-3)/1 +1 = 13
256
13
13
Overlapping Max POOL 3x3, stride=2
(13-3)/2 +1 = 6
256
6
6
9216
FC
4096
FC
4096
1000 Softmax

(b) The AlexNet (Krizhevsky *et al.*, 2017).

Figure 3.5: Examples of neural networks.

$\bar{\boldsymbol{x}}$ and $\boldsymbol{y}$ are the input and output vectors, and $\boldsymbol{\theta}$ is the vector of changeable parameters or weights. $\boldsymbol{y}$ is also called the prediction. Given $J(\boldsymbol{y})$ as a cost (objective) function of the prediction of the DNN, the optimisation problem of training a DNN could be formalised as follow:

$$\begin{aligned}
\text{Min/Max} \quad & J(\boldsymbol{y}) \\
\text{s.t.} \quad & \boldsymbol{y} = f(\bar{\boldsymbol{x}}; \boldsymbol{\theta}) \\
& g(\boldsymbol{\theta}) = \boldsymbol{a} \qquad (3.19) \\
& h(\boldsymbol{\theta}) \geq \boldsymbol{b} \qquad (3.20)
\end{aligned}$$

where Eq. 3.19 and 3.20 are equality and inequality constraints. Different forms of the objective function and constraints will induce different types of learning problems. For example, supervised learning seeks to minimise some distance between the known and predicted results, $\bar{\boldsymbol{y}}$ and $\boldsymbol{y}$, from the same input $\bar{\boldsymbol{x}}$, while unsupervised learning seeks to optimise some information-theoretic constraints or objectives so that the weights become easier-adapted onto new problems (Goodfellow *et al.*, 2016). For DRL, the problem would be to minimise some variants of the *mean-squared value error* (Eq. 3.8) or to maximise some form of the future return (e.g., Eq. 3.10).

However, because analytically solving for the optimal weights is impossible for large datasets and DNN models, DL methods use gradient descent to approach the (local-)optima. Recall that this is the same as how the gradient-based methods in RL work. The gradient update can be simply written down as $\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i + \alpha \nabla J(\boldsymbol{y})$, where α is the learning rate.

The specific method that computes the gradient, $\nabla J(\boldsymbol{y})$, and updates the weights is referred to as the **optimiser**. Vanilla gradient descent calculates the gradient at every update with all available data points, which is highly expensive for large datasets. Stochastic gradient descent calculates the gradient for one datapoint at every update, which is computationally expensive and produces high variance gradients. While **mini-batch** stochastic gradient descent is preferred as it computes the gradients with a selectable number of randomly sampled data points at every step (Li *et al.*, 2014). With the raised demands of processing large datasets, optimisers have been upgraded in a steady course. Arguably the most used one in the DRL community would be RMS-Prop or Adam (Zou *et al.*, 2019a).

In supervised learning tasks, such as image classification, the data points used to train a DNN are sampled from a *fixed* dataset (Goodfellow *et al.*, 2016). In DRL, most of the time these data points are sampled from an *ever-changing* dataset that is constantly renewed by adding new and deleting old experiences. This process is called **experience replay** (Lin, 1992; Mnih *et al.*, 2015). The idea is simple: reusing past interaction data, which brings us back to the topic of off-policy learning. The integration of mini-batch optimisation, experience replay and off-policy Q-learning is indeed so powerful that most modern successful DRL algorithms are built upon it. However, as shown in subsection 3.2.5, there are still improvements to be made to the fundamentals of deep q learning.

3.2.2 Deep q-learning

As the name implies, deep q-learning is the DNN-enhanced version of the q-learning algorithm (Mnih *et al.*, 2015). Instead of representing the q function by a matrix or linear function, a neural network is now used. It is referred to as the deep q-network (DQN). Similar to the linear function case, for discrete action spaces, the DQN can directly output all the action values given a state; but for continuous action spaces, it would only be able to evaluate one state-action pair for each input.

Recall that the minimisation objective for iteratively learning a parameterised q function through gradient is the mean square value error between the true q value and the predicted q value. In DRL research, the true q value is commonly referred to as the *target* q-value, as it is the regression target. Because the algorithm has access to only data, it needs to estimate the target using what it has learnt before and the new samples (Eq. 3.8). This leads to the objective of the Q-learning algorithm in the form of an expectation over mini-batch samples. Denote the target value estimate at iteration i as $\hat{q}_i$, and the n-th transition from a reply buffer $\mathcal{D}$ with a batch size N as $\xi_n = \{s_n, a_n, r_n, s'_n\}$, the objective function to be minimised can be written as:

$$\begin{aligned} J(\boldsymbol{w}_i) &\approx \mathbb{E}_{\xi_n \sim \mathcal{D}} \left[\frac{1}{2} (\hat{q}_n - \tilde{q}_{\boldsymbol{w}_i}(s_n, a_n))^2 \right] \\ &\approx \mathbb{E}_{\xi_n \sim \mathcal{D}} \left[\frac{1}{2} (r_n + \gamma \max_{a'_n} \tilde{q}_{\boldsymbol{w}_i^-}(s'_n, a'_n) - \tilde{q}_{\boldsymbol{w}_i}(s_n, a_n))^2 \right] \end{aligned} \tag{3.21}$$

where the expectation is taken over the distribution for sampling mini-batches from the replay buffer $\mathcal{D}$. The gradient update for DQN is then:

$$\boldsymbol{w}_{i+1} = \boldsymbol{w}_i + \alpha_w \frac{1}{N} \sum_{n=0}^{N} \left(r_n + \gamma \max_{a'_n} \tilde{q}_{\boldsymbol{w}_i^-}(s'_n, a'_n) - \tilde{q}_{\boldsymbol{w}_i}(s_n, a_n) \right) \nabla \tilde{q}_{\boldsymbol{w}_i}(\cdot) \tag{3.22}$$

where, α_w is the learning rate, $\tilde{q}_{\boldsymbol{w}_i^-}(s'_n, a'_n)$ is a copy of the q network being optimised and it is treated as a scalar value for computing the target value $\hat{q}$. This copied network is called the *target q-network*, while the one being optimised is called the *main network*. In the DQN algorithm, the target network is a delayed copy of the main network. Specifically, it copies the weight values from the main network at every C optimisation step. Replacing $\boldsymbol{w}_i^-$ with the main weights from last update $\boldsymbol{w}_{i-1}^-$ results in the original Q-learning update (Eq. 3.9). This delayed copy is suggested because it improves training stability compared to computing $\hat{q}$ with the network from the last update, which very easily leads to divergence with function approximation (Mnih *et al.*, 2015).

As with the original Q-learning, the DQN algorithm is off-policy. It learns the q function for a greedy policy: $\pi = \arg\max_a \tilde{q}_{\boldsymbol{w}_i}(s_n, a_n)$, while exploring the environment by taking actions from another policy. The exploratory policy is called ϵ-greedy, which takes a random action with a probability ϵ and otherwise takes the greedy action according to the current q value. In addition, the probability ϵ is linearly reduced from 1 to a lower bound over the course of training.

The main modifications that DQN has made compared to Q-learning basically include 1) the use of DNN-based function approximators, 2) the use of experience replay with mini-batch stochastic optimisation, and 3) the use of a target q network. Additionally, for the algorithm to work in the Atari game tasks, the authors also applied image preprocessing to construct a more compact input (Mnih *et al.*, 2015). In later years, DQN has been improved in various ways and researchers have proposed a number of optimisation tricks to stabilise training and improve convergence quality. A few important ones will be discussed in subsection 3.2.5.

Despite its success in Atari games and other discrete action domains, DQN struggles in continuous action tasks. As discussed in section 3.1, deploying q learning on continuous action spaces is expensive due to the maximisation operation for computing the target value during update or action selection. Therefore, the next two subsections will introduce two DL-based off-policy actor-critic algorithms, first with a deterministic actor, and then with a stochastic, entropy-maximisation actor.

3.2.3 Deep deterministic policy gradient

The deep deterministic policy gradient (DDPG) is the second important DRL algorithm that impressed the community. It is an off-policy actor-critic algorithm that optimises a DNN-parameterised deterministic policy $a = \pi_{\boldsymbol{\theta}}(s)$ (Lillicrap *et al.*, 2015). The core of DDPG is the *deterministic policy gradient theorem* proposed specifically for continuous state and action space problems (Silver *et al.*, 2014). So, what is the advantage of a deterministic actor over a stochastic one?

Recall that in subsection 3.1.4, the policy gradient theorem for a stochastic policy $\pi(a|s)$ was introduced (Eq. 3.10), with its on-policy and off-policy estimations based on samples. Here we start the derivation of DDPG from the off-policy stochastic policy gradient estimation using samples and a parametrised q function (Eq. 3.16):

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) &\approx \int_{\mathcal{S}} d^{\beta_{\boldsymbol{\theta}}}(s) \int_{\mathcal{A}} \tilde{q}_{\boldsymbol{w}}^{\pi_{\boldsymbol{\theta}}}(s,a)\pi_{\boldsymbol{\theta}}(a|s)\frac{\nabla_{\boldsymbol{\theta}}\pi_{\boldsymbol{\theta}}(a|s)}{\pi_{\boldsymbol{\theta}}(a|s)}\mathrm{d}s\mathrm{d}a \\ &\approx \mathbb{E}_{S_t\sim d^{\beta_{\boldsymbol{\theta}}}, A_t\sim\beta_{\boldsymbol{\theta}}}\Big[\rho_t \tilde{q}_{\boldsymbol{w}}^{\pi_{\boldsymbol{\theta}}}(S_t,A_t)\nabla_{\boldsymbol{\theta}}\log\pi_{\boldsymbol{\theta}}(A_t|S_t)\Big]\end{aligned}$$

where $\rho_t = \frac{\pi_{\boldsymbol{\theta}}(A_t|S_t)}{\beta_{\boldsymbol{\theta}}(A_t|S_t)}$ is the importance-sampling ratio. Notice the fact that the expected state value requires integration over the action space because every action has some probability to be selected given a stochastic policy. However, the case is different when it comes to a deterministic policy, $a = \pi_{\boldsymbol{\theta}}(s)$, whose gradient to maximise the objective using replay buffer samples can be written as:

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) &\approx \nabla_{\boldsymbol{\theta}}\left[\int_{\mathcal{S}} d^{\beta_{\boldsymbol{\theta}}}(s)\tilde{q}_{\boldsymbol{w}}^{\pi_{\boldsymbol{\theta}}}(s,\pi_{\boldsymbol{\theta}}(s))\mathrm{d}s\right] \\ &\approx \int_{\mathcal{S}} d^{\beta_{\boldsymbol{\theta}}}(s)\nabla_a \tilde{q}_{\boldsymbol{w}}^{\pi_{\boldsymbol{\theta}}}(s,a)\big|_{a=\pi_{\boldsymbol{\theta}}(s)}\nabla_{\boldsymbol{\theta}}\pi_{\boldsymbol{\theta}}(s)\mathrm{d}s \\ &\approx \mathbb{E}_{\xi_n\sim\mathcal{D}}\Big[\nabla_a \tilde{q}_{\boldsymbol{w}}^{\pi_{\boldsymbol{\theta}}}(s_n,a)\big|_{a=\pi_{\boldsymbol{\theta}}(s_n)}\nabla_{\boldsymbol{\theta}}\pi_{\boldsymbol{\theta}}(s_n)\Big] \qquad (3.23)\end{aligned}$$

where $s_n \in \xi_n$ is the state of the n-th sampled transition of a N-size mini-batch. As the policy is deterministic, the state value actually equals to the state-action value in expectation. In addition, there is no need to integrate over the action space, thus, no need to correct the gradient with the importance-sampling ratio. This in fact has a great impact in practice, as the stochastic version would require more samples because more actions need to be considered for a good estimate of the gradients. The gradient update rule of the deterministic actor with mini-batches is:

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i + \alpha_\theta \frac{1}{N}\sum_{n=0}^{N}\Big(\nabla_a \tilde{q}_{\boldsymbol{w}_j}^{\pi_{\boldsymbol{\theta}_i}}(s_n,a)\big|_{a=\pi_{\boldsymbol{\theta}_i}(s_n)}\nabla_{\boldsymbol{\theta}}\pi_{\boldsymbol{\theta}_i}(s_n)\Big) \qquad (3.24)$$

Similar to the stochastic off-policy AC algorithm, the critic update uses Q-learning. However, this time its actions come directly from a deterministic policy because it is inefficient to compute the maximisation operation on continuous action space. Thus, the gradient update rule for the critic with mini-batches is:

$$\boldsymbol{w}_{j+1} = \boldsymbol{w}_j + \alpha_w \frac{1}{N} \sum_{n=0}^{N} \left(\delta_t \nabla_{\boldsymbol{w}} \tilde{q}_{\boldsymbol{w}_j}^{\pi_{\boldsymbol{\theta}_i}}(s_n, a_n) \right) \tag{3.25}$$

where, the TD error $\delta_t = r_n + \gamma \tilde{q}_{\boldsymbol{w}_j^-}^{\pi_{\boldsymbol{\theta}_i^-}}(s_n^{'}, a')|_{a=\pi_{\boldsymbol{\theta}_i}(s_n)} - \tilde{q}_{\boldsymbol{w}_j}^{\pi_{\boldsymbol{\theta}_i}}(s_n, a_n)$

Similar to DQN, $\boldsymbol{w}^-$ and $\boldsymbol{\theta}^-$ are parameters of the separate copies of the main networks. They are only used to calculate the target q values with fixed weight values. However, different from DQN, DDPG uses another scheme to update the weights of the target networks. In particular, they are updated softly using Eq. 3.26, instead of a hard copy of the main network.

$$\boldsymbol{w}_{j+1}^- = \lambda \boldsymbol{w}_{j+1} + (1-\lambda)\boldsymbol{w}_j^-, \quad \boldsymbol{\theta}_{i+1}^- = \lambda \boldsymbol{\theta}_{i+1} + (1-\lambda)\boldsymbol{\theta}_i^- \tag{3.26}$$

where $0 < \lambda \ll 1$ is the soft update ratio.

As discussed, a deterministic policy requires another policy to explore the environment and collect data. This is called the behavioural policy. In DDPG, the authors proposed to use the Ornstein-Uhlenbeck process to generate temporally correlated noises that perturb the actions produced by the actor network at every state. They hypothesised that this will result in better exploration in physical environments that have momentum (Lillicrap *et al.*, 2015). However, this was empirically shown to be unnecessary and researchers opted to use variations of Gaussian noises (Andrychowicz *et al.*, 2017; Fujimoto *et al.*, 2018).

In sum, the DDPG algorithm uses a behavioural policy to collect data, uniformly samples mini-batches from a replay buffer to update the critic with Q learning (Eq. 3.25), the actor with the deterministic policy gradient theorem (Eq. 3.25), and their target networks softly (Eq. 3.26). It improved the sample efficiency over on-policy DRL methods for tasks with continuous state and action spaces and became one of the most used algorithms by the robotic community (Singh *et al.*, 2021). However, DDPG is difficult to use because it is highly sensitive to training parameters such as learning rate and the size of the mini-batch. In addition, though a deterministic policy may converge faster in certain task settings, it will have difficulty facing stochastic environments where uncertainty should be accounted for and it is only a special case for the stochastic policy gradient theorem (Silver *et al.*, 2014). Therefore, one may still prefer to develop stable and sample-efficient off-policy algorithms for learning stochastic policies. Soft actor-critic is an important state-of-the-art solution to this problem.

3.2.4 Soft actor critic

The foundation of soft actor-critic (SAC) is the framework of maximum entropy reinforcement learning (MaxEnt RL) (Haarnoja *et al.*, 2018). The root difference is that MaxEnt RL augments the maximisation objective of the original RL problem (Eq. 3.3) with an extra entropy term as follows:

$$\begin{aligned} J(\pi) &= \mathbb{E}_\pi\left[\sum_{t=0}^{T}\gamma^t(r(s_t,a_t)+\alpha_{\mathcal{H}}\mathcal{H}_\pi(\cdot|s_t))\right] \\ &= \sum_{t=0}^{T}\mathbb{E}_\pi\left[\gamma^t(r(s_t,a_t)-\alpha_{\mathcal{H}}\log\pi(\cdot|s_t))\right] \end{aligned} \tag{3.27}$$

where $\alpha_{\mathcal{H}}$ is the temperature parameter that determines the balance between maximising the two terms. To clarify, the entropy of a stochastic distribution measures the randomness of that distribution. Therefore, by maximising Eq. 3.27, the RL policy will, in expectation, maximise return and remain robust in the presence of uncertainty. In other words, the entropy maximisation term guarantees the policy to have non-zero probabilities for all the actions, preventing it from collapsing into a deterministic policy. This is important for stochastic environments, where the optimal policy is stochastic.

Before the SAC algorithm, empirical results of on-policy and discrete action MaxEnt RL policies have shown improvements in terms of convergence speed, sample efficiency, and higher performance compared to their classic RL counterparts (Haarnoja *et al.*, 2017; Schulman *et al.*, 2017a). Based on them, the success of SAC was not much of a surprise.

As with all actor-critic algorithms, SAC also iterates among data collection, soft policy evaluation and soft policy improvement. The soft version of the q function and value function are defined as follows:

$$q^{\pi}(s_t, a_t) = r(s_t, a_t) + \gamma \mathbb{E}_{\pi}[v^{\pi}(s_{t+1})] \tag{3.28}$$

$$v^{\pi}(s_t) = \mathbb{E}_{\pi}[q(s_t, a_t) - \log \pi(a_t|s_t)] \tag{3.29}$$

Similar to DDPG, SAC employs Q-learning but with a stochastic policy and instead uses the soft version of the objective to learn the parameters of a q network. Given a transition $\xi_n = \{s_n, a_n, r_n, s'_n, a'_n\}$, the objective function to be minimised for the soft critic associated with a parametrised q function and policy is:

$$J(\boldsymbol{w}) = \mathbb{E}_{\xi_n \sim \mathcal{D}} \left[\frac{1}{2} (\hat{q}_n - \tilde{q}^{\pi_{\boldsymbol{\theta}}}_{\boldsymbol{w}}(s_n, a_n))^2 \right] \tag{3.30}$$

where $\hat{q}_n = r_n + \gamma \tilde{q}^{\pi_{\boldsymbol{\theta}}}_{\boldsymbol{w}^-}(s'_n, a'_n) - \alpha_{\mathcal{H}} \log \pi_{\boldsymbol{\theta}}(a'_n|s'_n)$ is the soft target value, $\boldsymbol{w}^-$ represents the weights of the target network.

For policy improvement, it is different from the common practice for standard RL. Instead of pushing the parameters of the policy towards the direction of the q function, soft policy improvement minimises the Kullback-Leibler divergence between the current policy and the exponential of the *updated* Q function, $\tilde{q}^{New}_{\boldsymbol{w}}$. The objective function to be minimised using replay samples is:

$$\begin{aligned}J(\boldsymbol{\theta}) &\approx \mathbb{E}_{\xi_n\sim\mathcal{D}}\left[\mathrm{D_{KL}}\left(\pi_{\boldsymbol{\theta}}(\cdot|s_n)\middle|\middle|\frac{exp(\tilde{q}^{New}_{\boldsymbol{w}}(s_n,\cdot))}{Z_{\boldsymbol{w}}(s_n)}\right)\right]\\ &\approx \mathbb{E}_{\xi_n\sim\mathcal{D}}\left[\int_{\mathcal{A}}\pi_{\boldsymbol{\theta}}(\cdot|s_n)\log\left(\frac{\pi_{\boldsymbol{\theta}}(\cdot|s_n)Z_{\boldsymbol{w}}(s_n)}{exp(\tilde{q}^{New}_{\boldsymbol{w}}(s_n,\cdot))}\right)\mathrm{d}a\right]\\ &\approx \mathbb{E}_{\xi_n\sim\mathcal{D}}\Big[\mathbb{E}_{a\sim\pi_{\boldsymbol{\theta}}}\left[\alpha_{\mathcal{H}}\log\pi_{\boldsymbol{\theta}}(a|s_n)-\tilde{q}^{New}_{\boldsymbol{w}}(s_n,a)\right]\Big]\end{aligned} \tag{3.31}$$

Eq. 3.31 here is obtained by dropping the partition function $Z_{\boldsymbol{w}}(s_n)$ and multiplied by the temperature parameter $\alpha_{\mathcal{H}}$. In order to compute the gradients, the policy is specifically chosen from a set of parameterised distributions such as Gaussian, whose mean and deviation are produced by a differentiable neural network. By applying the reparameterisation trick, the policy gradient can be estimated from samples readily (Haarnoja *et al.*, 2018).

Empirical results of alternating the updates between Eq. 3.30 and 3.31 have shown impressive improvements over DDPG, TD3 (enhanced DDPG) and other popular algorithms at the time (Haarnoja *et al.*, 2018). Later on, the authors proposed a technique to automatically adjust the temperature parameter, liberating users from manually fine-tuning it for every new task. The solution begins with formulating the MaxEnt RL problem as a constrained optimisation problem that constrains the average entropy of the policy:

$$\max_{\pi}\ \mathbb{E}_{\pi}\left[\sum_{t=0}^{T}\gamma^t r(s_t,a_t)\right]\quad \text{s.t. } \mathbb{E}_{\pi}[-\log\pi(a_t|s_t)]\geq\bar{\mathcal{H}}$$

Solving the dual problem of this constrained optimisation problem leads to the soft actor-critic update as well as an update to the dual variable, which is exactly the temperature parameter in the standard formulation. The deriving of the solution is omitted here and can be found in the original paper (Haarnoja *et al.*, 2018). The objective function to be minimised with respect to the temperature along with its sample-based approximation are:

$$J(\alpha_{\mathcal{H}}) = \mathbb{E}_{\pi}[-\alpha_{\mathcal{H}} \log \pi(a|s) - \alpha_{\mathcal{H}}\bar{\mathcal{H}}] \tag{3.32}$$

$$\approx \mathbb{E}_{\xi_n \sim \mathcal{D}}\Big[\mathbb{E}_{a \sim \pi_{\boldsymbol{\theta}}}\left[-\alpha_{\mathcal{H}} \log \pi_{\boldsymbol{\theta}}(a|s_n) - \alpha_{\mathcal{H}}\bar{\mathcal{H}}\right]\Big] \tag{3.33}$$

where $\bar{\mathcal{H}}$ is the target entropy, which in practice normally equals the negative number of action dimensions.

In sum, the SAC algorithm seeks to find the optimal stochastic policy with respect to an augmented objective which in addition proportionally maximises the entropy of the policy. This prevents the policy from fully collapsing to a deterministic one very fast, which is suboptimal when the environment is stochastic. Also, the automatic temperature update ensures the policy is deterministic enough when good and bad actions can be distinguished clearly. SAC indeed has achieved impressive results on model-free continuous control problems (Haarnoja *et al.*, 2018).

3.2.5 Optimisation tricks

Up till now, the three pioneer DRL algorithms for model-free off-policy learning have been introduced. A decade after the publication of the first DQN algorithm (Mnih *et al.*, 2013), different improvements and variations of these algorithms are still being proposed to adapt these algorithms into more practical and realistic tasks (Lazaridis *et al.*, 2020). From a theoretical point of view, the derivation process and the original algorithms look very promising. However, making DRL algorithms actually work turns out to be very difficult and case-specific, especially with the use of deep neural networks. In the past few years, researchers have discovered various optimisation tricks that enable, stabilise or accelerate the training of DRL agents. This section will introduce some very common ones that are also used in this thesis.

Double Q learning. The off-policy learning benefit of q learning needs no more words to explain. However, it is well known in the community that the maximisation operation during the update will cause an overestimation of the q values because the next-state q value in the computation of the approximation of the true q value also comes from an imperfectly learnt function. This approximation to the Bellman update is called *bootstrapping*. In other words, learning what is new partly from the old knowledge. The problem becomes more complicated with neural network approximators, whose every weight changes when one state-action pair is updated. This is the reason why a target q network was introduced in the original DQN paper (Mnih *et al.*, 2015). However, there is always room to reduce the overestimation error, and double q learning is one of the popular approaches (Hasselt, 2010). Different forms of double q learning with DNNs have emerged. The very first one uses a target network to evaluate actions selected greedily by the main network (Van Hasselt *et al.*, 2016). While the latest and most successful one uses two q functions with respective target functions for actor-critic algorithms (Fujimoto *et al.*, 2018). The idea is to find the minimum q value estimate possible by taking the minimum among the predicted values from the two target networks:

$$\tilde{q} = r + \gamma \min_{k=1,2} \tilde{q}^{\pi_{\boldsymbol{\theta}^-}}_{\boldsymbol{w}_k^-}(s', \pi_{\boldsymbol{\theta}^-}(s')) \tag{3.34}$$

where $\boldsymbol{w}_k^-$ is the weights of the k-th target q network, and $\boldsymbol{\theta}^-$ is the weights of the target policy network. Empirically, using Eq. 3.34 exhibits far less overestimation error and speeds up learning for both deterministic and stochastic actors (Haarnoja *et al.*, 2018). However, the use of a target policy network was suggested to be less important and sometimes discarded. Another modification that helps in learning the q function with a deterministic actor is to add a small amount of Gaussian noises into the action computing in Eq. 3.34, averaged over mini-batches, resulting in:

$$\tilde{q} = r + \gamma \min_{k=1,2} \tilde{q}_{\boldsymbol{w}_k^-}^{\pi_{\boldsymbol{\theta}^-}} (s', \pi_{\boldsymbol{\theta}^-}(s') + \epsilon) \tag{3.35}$$

where, $\epsilon \sim \text{clip}\,(\mathcal{N}(0, \sigma), -c, c)$ is the clipped zero-mean Gaussian noise, σ is the user-specified variance of the Gaussian and c is the absolute bound value (Fujimoto *et al.*, 2018).

Value clip. The use of a q-learning-style update is bound to lead to overestimation errors. A trick that can be employed is to clip the value estimate within a rational range. This can be easily done in most tasks with a given reward function, by analytically estimating the highest and lowest possible returns given a fixed episode length and a discount factor. At each update to the q function or the policy, the target q value can be clipped within this rational range. In the case of updating the policy, this can result in a clipped gradient (Andrychowicz *et al.*, 2017; Fujimoto *et al.*, 2018).

Input normalisation. Another thing that may help to speed up training is to normalise the inputs of the neural networks (Sola and Sevilla, 1997). For pixel-input tasks, the input is normalised into $[0, 1]$ by dividing 255. For non-image inputs, this is less straightforward for RL problems. As mentioned, the dataset or the data distribution, from which the mini-batch samples are drawn to update the networks, is ever-changing in RL problems. This is because the data depend partly on an ever-changing policy. Therefore, the input normalisation method used in most RL algorithms, especially for continuous state tasks, is the mean-deviation normalisation with the statistical mean and deviation calculated from the data collected by the algorithm so far (Lillicrap *et al.*, 2015; Andrychowicz *et al.*, 2017). However, state normalisation does not guarantee to work and is not always necessary (Fujimoto *et al.*, 2018; Haarnoja *et al.*, 2018).

3.2.6 Summary

Up until now, the readers should be familiar with the foundation of modern DRL algorithms. They tightly centred around the use of off-policy q learning and its extension to continuous problems with a parameterised actor (deterministic or stochastic). On-policy DRL algorithms such as the proximal policy optimisation (PPO) method have their contributions and advantages on certain tasks (Andrychowicz *et al.*, 2020), but they are too unrelated to the focus of this thesis. The interest of this thesis is set upon robotic manipulation tasks where off-policy demonstrations and exploration data are much easier to obtain and more valuable than on-policy data.

However, although DRL algorithms have made impressive progress, it is not surprising that the formulation of the standard RL problem given the complexity of the real world. Therefore, various new RL formulations have been proposed to cope with problems that are not obvious to be modelled by the standard MDP. In the following three sections, the three important extensions of the standard RL problem will be introduced. They are the foundations of the contributions made by this thesis.

Chapter 4

A^2: Accelerate Reinforcement Learning for Multi-step Robotic Manipulation

4.1 Introduction

Reproducing the object manipulation skills manifested by humans on robots has been one of the central research topics in the robotic community. For decades scientists identified various subproblems in this area and developed methods to tackle them, yet today's robotic manipulation systems still have a long way to go (Billard and Kragic, 2019).

One of the difficult problems is how to enable a robot to learn multistep manipulation tasks. These tasks typically can be decomposed into a number of subtasks and accomplished by a number of motion skills. For instance, assembling a number of car parts, building a Lego house, or pushing a block into a closed chest as shown by Figure 4.1.

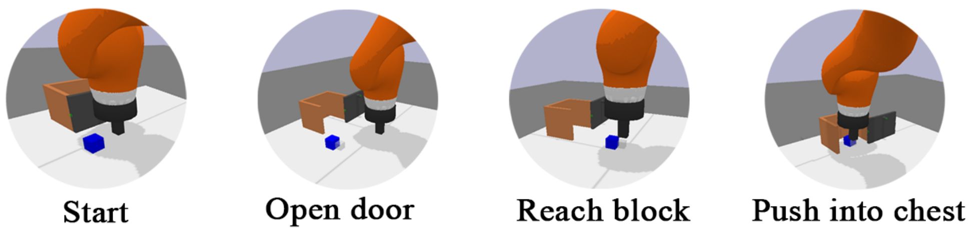


Figure 4.1: The multistep manipulation task of pushing a block into a closed chest.

There are several solution frameworks for such tasks. The most classic approach is *task and motion planning* (TAMP), which uses symbolic languages to find a skeleton of motion skills and identifies the particular parameters for these motion skills (Karpas and Magazzeni, 2020; Garrett *et al.*, 2021). The primary limitation of TAMP methods is that it requires substantial human knowledge to design a computable model of the dynamics of the environment, either for subtask reasoning or motion skill generation. This is problematic when object interaction is involved in the task because it is generally difficult to model rich contact dynamics. Therefore, data-driven methods that do not rely on a dynamic model are increasingly desirable (Kroemer *et al.*, 2021).

In recent years researchers have developed and tested a diversity of data-driven methods for manipulation skill learning (Kroemer *et al.*, 2021). These methods seek to learn a data-driven reactive policy that produces robot action commands when given an observation of the world. They attempt to free the control algorithms from the assumption of a given dynamic model. The data may come from a number of different sources. If it comes from a human expert or a hand-engineered programme, it is called demonstration data. If it comes from a policy that randomly takes actions to see how the environment reacts, it is called exploration data. Learning methods can be classified according to how they optimise the policy using these data. *Imitation learning* (IL) seeks to optimise a policy so that it matches the policy that produced the demonstration data (Hussein *et al.*, 2017). RL seeks to optimise a policy that maximises the expected cumulative rewards (Sutton and Barto, 2018).

In practice, IL and demonstrations alone are often not enough, because demonstrations can only provide a limited set of experiences, resulting in serious distribution mismatch (Ross *et al.*, 2011). This is exacerbated in robotics as collecting robot demonstrations is a time-consuming, labour-intensive and specialised work (Fang *et al.*, 2019a). Therefore, it is common for further training to take place on top of the use of demonstrations. In many such cases, the policy is fine-tuned with RL, either it is initialised from demonstrations or it is trained with a mixture of demonstration and exploration data (Ramírez *et al.*, 2022).

On the other hand, using demonstrations along with exploration is also preferred by RL algorithms, because exploration alone tends to be insufficient

(Ramírez *et al.*, 2022). The first reason is that for many complex robotic manipulation tasks, the learning signal only comes from the task completion condition, so random exploration will take too long to encounter useful data. (Liu *et al.*, 2021). This is known as the sparse reward problem, which happens to tasks where a dense reward function is difficult to specify. Secondly, this situation is further exacerbated in multistep tasks, where the length of the successful trajectory and the diversity of required manipulation skills increases. Take the pushing task shown in Figure 4.1 as an example. The robot does not receive a positive learning signal until it accidentally opens up the chest door, reaches the block and pushes it into the chest. A dense reward function is difficult to design for such behaviours, and the likelihood of this series of events happening is very low. There have been examples of utilising demonstrations to accelerate the learning of such multistep and long-horizon manipulation tasks (Nair *et al.*, 2018; Gupta *et al.*, 2019).

However, demonstrations are not easy to collect, especially for robotic tasks. The common type of robot demonstration is kinesthetic trajectories, which consist of time-ordered series of observation-action pairs. They may be collected through kinesthetic teaching, teleoperation, motion capture or external sensor recording (Ramírez *et al.*, 2022). As mentioned, any one of them is costly to perform.

Therefore, the first idea proposed in this chapter takes advantage of *abstract demonstrations* to accelerate learning. In simple terms, abstract demonstrations refer to the sequences of subtasks or skills required to achieve the overall task. For example, a Lego house set normally comes with a manual that specifies the number of steps to assemble the house. The same happens to various furniture or products that require users to assemble by themselves. This kind of demonstration may be preferred because it is easier to collect by humans. However, there are two assumptions that should be satisfied for the power of such demonstrations to be fully exposed.

Firstly, the method assumes access to a task decomposition scheme. Such a scheme can be easily provided by a human expert for many tasks, as conducted in this chapter. However, it can also be automated, for example, by a neural network. This thought actually leads to the problem of subgoal, skill or option discovery, which is another broad research field (Khetarpal *et al.*, 2020b; Pateria *et al.*, 2021a; Cho *et al.*, 2022), out of the scope of this thesis.

Secondly, the policy is assumed to be able to learn the kinesthetic trajectories required to move from one subtask to another, through any other techniques. There are many choices to satisfy this assumption. For example, one may use kinesthetic demonstrations provided by a classic motion planner, which is known to be stable enough for short trajectory generation (Latombe, 2012). In this chapter, the GRL framework and the hindsight experience replay (HER) technique (Schaul *et al.*, 2015; Andrychowicz *et al.*, 2017) are used to guarantee the successful learning of the short trajectory in between each subtask in the face of reward sparsity.

This chapter also proposes to improve another condition that occurs in the application of RL in multistep manipulation tasks. As one can see, one characteristic of multistep tasks is the dependencies among steps. For example, the robot cannot push the block into the chest unless it opens the chest door in advance. This means that the robot cannot learn about the reward of successfully pushing the block into the chest unless it knows that it should always open the door first. This would require the policy to reduce exploration for subtasks or skills that it has already mastered, but existing exploration strategies focus more on visiting unseen states, prediction error or the number of collected training samples (Ladosz *et al.*, 2022).

Therefore, the second idea of this chapter suggests adapting exploratory behaviours according to the performance of each subtask in the multistep task setting. It is desired to explore more when the subtask performance is low, and vice versa. In particular, this idea is implemented and experimentally demonstrated on three popular RL algorithms: deep q-learning (Mnih *et al.*, 2013), deep deterministic policy gradient (Lillicrap *et al.*, 2015) and soft actor-critic (Haarnoja *et al.*, 2018).

4.1.1 Summary and chapter organisation

In sum, this chapter seeks to improve end-to-end RL for multi-step manipulation with delayed and sparse reward signals. According to the analysis given above, state-of-the-art methods rely on kinesthetic demonstrations to improve sample efficiency (Nair *et al.*, 2018; Gupta *et al.*, 2019). This chapter proposes two techniques to improve: using abstract demonstrations and adapting exploration according to subtask performances. Overall, the two techniques are named $\mathbf{A^2}$.

The rest of the chapter is organised as follows. Section 4.2 will introduce the GRL framework formally, the problem descriptions of the multi-step tasks of interest, and the $\mathbf{A^2}$ method in detail. Section 4.3 will illustrate the experiment design and discuss empirical results. Lastly, section 4.4 concludes this chapter.

4.2 Method

This section will first describe the mathematical formulations and assumptions employed in the study, then formally illustrate the ideas of abstract demonstrations and adaptive exploration. The reader is suggested to become familiar with section 3.1 and 3.2 for the standard DRL framework and algorithms, before reading the following contents. However, cross-references are applied in places of this chapter in cases when necessary.

4.2.1 Goal-conditioned reinforcement learning

As mentioned, this chapter is interested in improving long-horizon manipulation tasks. In particular, the task will be studied in the framework of GRL (Schaul *et al.*, 2015; Andrychowicz *et al.*, 2017), because it is convenient to describe the multiple steps of a task as a number of subspaces of the goal space (see subsection 4.2.3). To demonstrate so, a formal description of the GRL framework is first given in this subsection, along with a goal-relabelling technique named hindsight experience replay that is used to accelerate learning.

In simple terms, the GRL problem differs from traditional RL by optimising the return not only based on a single-objective reward function that depends on states and/or actions but also incorporates a goal vector. This multi-objective reward function aims to optimise the return with respect to achieving specific goals in addition to traditional reward considerations. The input to the value functions and the policy is therefore augmented by an extra term: the goal. The learnt policy is expected to exhibit different behaviours at the same state according to different assigned goals, achieving some degree of information or knowledge sharing (Schaul *et al.*, 2015; Andrychowicz *et al.*, 2017).

Goal-augmented MDP

As introduced in subsection 3.1.1, a standard MDP is a tuple of state, action, transition probability, reward and a discount factor. In this chapter, we will stay within the same discrete time, finite-horizon, fully observable and discounted setting. Please refer to subsection 3.1.1 for recalling the definitions of these terms. The goal-augmented MDP is the same tuple with an extra goal space: $(\mathcal{S}, \mathcal{A}, p, \mathcal{G}, p_0, r, \gamma)$. The reward function is now defined as a mapping from the state, action, as well as the goal spaces to some real number: $r : \mathcal{S} \times \mathcal{A} \times \mathcal{G} \rightarrow \mathbb{R}$.

In practice, the representation of the goal vector is usually some transformation of the state: $g = m(s)$. A simple case would be identity mapping, such that $g = s$, or part of the state, such as the position of an object (Andrychowicz *et al.*, 2017). More complicated cases could be languages (Jiang *et al.*, 2019) or an image (Xu *et al.*, 2021). An assumption for the GRL problem is that there is always a goal to be achieved given a state: $\forall s \in \mathcal{S}, \exists g \in \mathcal{G}$ s.t. $g = m(s)$.

An example would help understand the concepts of goal and its representation mapping $m(s)$. Let a state of the pushing task shown in Figure 4.1 be represented as a concatenated vector of the poses of the gripper tip, the block and the chest, and the opened width of the door, denoted as $\mathbf{s} = \{\mathbf{x}_{grip}||\mathbf{x}_b||\mathbf{x}_{chest}||w_{door}\}$. We may define the representation of the goal to be the position of the block, denoted as $\mathbf{g} = m(\mathbf{s}) = \mathbf{x}_b$. However, according to the task requirement, we could define the representation of the goal to also include the position of the gripper tip, denoted as $\mathbf{g} = m(\mathbf{s}) = \{\mathbf{x}_{grip}||\mathbf{x}_b\}$. Here, the two goal representation mappings are different but both are part of the state vector. For these two cases, it is also obvious that, given a state, there is always a goal that can be achieved at that state. Another example is to use language to represent the goal. For example, use the phrase "the block is at ____". Such a representation mapping would be much more difficult to manually define and may require other natural language processing techniques.

However, what is really the user's concern is a subset of the goal space that the algorithm is supposed to find. For instance, the target position of the block. Hence, we introduce the distinction among *achieved*, *desired* and *undesired* goals. The achieved goal, g_t, is whatever that is achieved at the current state according to $m(s_t)$, such as the current block position; the desired goal, $g^+ \in \mathcal{G}^+$, is what the task requires to achieve; while the undesired goal is whatever outside of the subset of desired goals $g^- \in \mathcal{G}^- = \mathcal{G} - \mathcal{G}^+$.

Reward function. Typically, the reward function is designed based on some distance measure between an achieved goal and a desired goal, $r(s_t, a_t, g^+) = d(m(s_{t+1}), g^+)$ where $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$. For example, the negative Euclidean distance. Nonetheless, there are many cases in the real world where a dense reward function based only on some distance measurement is insufficient. For example, when a robot is separated from its navigation destination by a wall, the Euclidean distance measure would not provide useful information that motivates the robot to find another way to bypass the wall and reach the goal. It would only cause the robot to keep going towards the wall, behind which is the destination. On the other hand, shaping the reward function to induce specific behaviour patterns may help in certain scenarios, but it tends to be more difficult and thus less preferred in the real world for the sake of multi-goal learning and generalisation. Therefore, a binary reward function that informs whether a desired goal is achieved by thresholding the distance measurement is more commonly used. For instance: $r(s_t, a_t, g^+) = \mathbb{1}\left[d(m(s_{t+1}), g^+) \leq \delta_d\right]$, where $\mathbb{1}$ is the indicator function. This reward function simply gives a value 1 when a desired goal is achieved, and 0 otherwise (Andrychowicz *et al.*, 2017).

Goal-augmented algorithms. Extending the standard algorithms to the GRL setting is in fact straightforward. First of all, notice that the given desired goal is independent of the dynamic of the system. In other words, it does not affect the resultant next state of an action taken at the current state. What it does affect in the MDP is only the reward function, and thus the values and the policy. With a trajectory of length T, the goal-conditioned q value and the greedy policy are defined as follows:

$$q^{\pi}(s_t, a_t, g^+) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{T} \gamma^t r(s_t, a_t, g^+)\right] \tag{4.1}$$

$$\pi(a_t|s_t, g^+) = \arg\max_{a_t} q^{\pi}(s_t, a_t, g^+) \tag{4.2}$$

The goal-augmented q function and policy are called *universal* q function and policy, respectively, because they are the general cases of the single objective RL problem (Andrychowicz *et al.*, 2017). Consider a special MDP where the state is the combination of the state and the goal in a goal-augmented MDP. Because the goal does not affect the transition, the special MDP retains the dynamical property of the goal-augmented one, and it becomes the standard MDP discussed in section 3.1. Note that preserving the goal information within the state representation is common in practice. However, factoring out the goal from the state enables the agent to learn a set of value functions and policies with respect to different goals (Sutton *et al.*, 2011; Schaul *et al.*, 2015). This is a valuable benefit of GRL in terms of knowledge sharing and multi-task learning.

Therefore, due to the independence between the goal and the system dynamic, the goal-augmented extensions of the DQN, DDPG and SAC algorithms are fairly straightforward. What needs to be done is to simply extend the input with an extra desired goal vector, because changing the goal has no impact on the expectation estimated during the derivation processes. Thus, following the notation conventions developed in chapter 3, given the n-th goal-conditioned MDP transition $\xi_n = \{s_n, a_n, r_n, g_n^+, s'_n, a'_n\}$ from a mini-batch of size N, we can modify the optimisation objectives of DQN, DDPG and SAC algorithms to be goal-conditioned as follows:

DQN and DDPG critic minimisation objective (Eq. 3.21):

$$J(\boldsymbol{w}) \approx \mathbb{E}_{\xi_n \sim \mathcal{D}} \left[\frac{1}{2} ((r_n + \gamma \max_{a'} \tilde{q}_{\boldsymbol{w}^-}(s'_n, a', g_n^+)) - \tilde{q}_{\boldsymbol{w}}(s_n, a_n, g_n^+))^2 \right] \tag{4.3}$$

DDPG policy maximisation objective (Eq. 3.23):

$$J(\boldsymbol{\theta}) \approx \mathbb{E}_{\xi_n \sim \mathcal{D}} \left[\tilde{q}^{\pi_{\boldsymbol{\theta}}}(s_n, a, g_n^+)|_{a=\pi_{\boldsymbol{\theta}}(s_n, g_n^+)} \right] \tag{4.4}$$

SAC critic minimisation objective (Eq. 3.30):

$$J(\boldsymbol{w}) \approx \mathbb{E}_{\xi_n \sim \mathcal{D}} \left[\frac{1}{2} (\hat{q}_n - \tilde{q}^{\pi_{\boldsymbol{\theta}}}_{\boldsymbol{w}}(s_n, a_n, g_n^+))^2 \right] \tag{4.5}$$

where $\hat{q}_n = r_n + \gamma \tilde{q}^{\pi_{\boldsymbol{\theta}}}_{\boldsymbol{w}^-}(s'_n, a'_n, g_n^+) - \alpha_{\mathcal{H}} \log \pi_{\boldsymbol{\theta}}(a'_n | s'_n, g_n^+)$

SAC policy minimisation objective (Eq. 3.31):

$$J(\boldsymbol{\theta}) \approx \mathbb{E}_{\xi_n \sim \mathcal{D}, a \sim \pi_{\boldsymbol{\theta}}} \left[\alpha_{\mathcal{H}} \log \pi_{\boldsymbol{\theta}}(a | s_n, g_n^+) - \tilde{q}^{New}_{\boldsymbol{w}}(s_n, a, g_n^+) \right] \tag{4.6}$$

where, D is the replay buffer, $\boldsymbol{w}$ and $\boldsymbol{\theta}$ stand for the parameters of neural networks.

Goal relabelling

Aside from the potential value of learning multiple goals, such as knowledge sharing and possibly accelerated adaptation onto unseen goals, learning more than one goal is naturally more difficult than learning a single goal. For example, it is always easier but less meaningful to learn to navigate to a single destination, and vice versa. However, it turns out that, by factoring out the goal from the state representation, the GRL framework provides an appealing way to boost learning sample efficiency: the relabelling of goals (Andrychowicz *et al.*, 2017; Eysenbach *et al.*, 2020).

The idea of goal relabelling is quite straightforward to understand. It takes a transition tuple $\xi_n = \{s_n, a_n, r_n, g^+_n, s'_n\}$, and finds a task for which the action in that old transition is optimal. In other words, it learns from the hindsight perspective. What was done by the agent may not be optimal for the given desired goal when the experience was collected, but it might have been optimal for another task. In fact, that experience would be valuable for learning many other tasks or goals instead of the originally given one. While it can be generalised to arbitrary reward functions with inverse RL (Eysenbach *et al.*, 2020), this thesis will only focus on the first goal relabelling technique, HER, which replaces the original goal of a transition according to some goal-sampling strategy (Andrychowicz *et al.*, 2017).

Formally, goal-relabelling replaces the desired goal and reward in an old transition with another goal and its associated reward. The desired goal, g^+, and the substitution goal, $\bar{g}^+$, are sampled from two different distributions. In practice, the desired goal is normally sampled uniformly from the set of desired goals $\mathcal{G}^+$, while the distribution of the substitution goals varies from method to method, with different purposes. In HER, the authors propose to copy transitions of an episode and replace their desired goals with substitution goals sampled from four distributions or strategies (Andrychowicz *et al.*, 2017), including:

- final: the desired goal is replaced by the achieved goal in the last state of the episode, i.e., $\bar{g}^+ = m(s_T)$.
- future: the desired goal is replaced by K achieved goals uniformly from the future transitions of the same episode,i.e., $\bar{g}^+_k = m(s_{t_k})$ where, $t_k \sim \mathcal{U}\{t, T\}$, $k \in \{0, 1, ..., K\}$. When $(T - t_k) < K$, t_k is set to $(K - T)$.

$\mathcal{U}\{t, T\}$ denotes a uniform distribution over the set of integers in $[t, T]$.

- episode: the desired goal is replaced by K achieved goals uniformly from the same episode, i.e., $\bar{g}_k^+ = m(s_{t_k})$ where, $t_k \sim \mathcal{U}\{0, T\}$, $k \in \{0, 1, ..., K\}$.
- random: the desired goal is replaced by K achieved goals uniformly sampled from the replay buffer, i.e., $\bar{g}_k^+ = m(s_k)$, where, $s_k \sim \mathcal{U}(s \mid s \in \mathcal{D})$, $k \in \{0, 1, ..., K\}$.

The improvement brought by the goal-relabelling techniques using these four strategies is substantial, enabling the DDPG agent to efficiently tackle sparse reward robotic manipulation tasks that it previously had no hope of tackling. Readers are encouraged to read the detailed experimental results of the original paper (Andrychowicz *et al.*, 2017). For the contributions of this thesis, the future sampling strategy is applied for experiments in this chapter, and the episode strategy for chapter 5, both with $K = 4$.

4.2.2 Problem description and assumptions

Following the formulation of the goal-augmented Markov decision processes described above, the following assumptions are made for the multi-step manipulation tasks setting in this chapter:

- The task is episodic, meaning that the task is reset to its initial state after T actions are taken.
- For each task, there exists a non-empty subset of the goal space covering all the desired goals for the task: $\mathcal{G}^+ \neq \emptyset$ and $\mathcal{G}^+ \in \mathcal{G}$.
- Each episode starts with a desired goal uniformly sampled from the corresponding subset of the goal space $g^+ \sim \mathcal{U}(\mathcal{G}^+)$ where $\mathcal{U}$ denotes a uniform distribution.

- At each state, there is always a goal that can be achieved at that state: $\forall\ s \in \mathcal{S}, \exists\ g \in \mathcal{G}$ s.t. $g = m(s)$.
- The reward function gives a reward of 0 when the Euclidean distance of the achieved and desired goals is smaller than a threshold δ_d, and a reward of -1 otherwise.

These assumptions feature a typical episodic and sparse reward goal-condition reinforcement learning (GRL) problem. At the start of each episode, a desired goal is provided, and the GRL agent is tasked to find a state which corresponds to an achieved goal close enough to the desired goal. These assumptions can be implemented in a grid world task in Figure 4.2 as follows. The task of the agent (red triangle) in this example is to collect the key, go through the door and reach the green cell in the right room. The grey cells are walls. In this case, the episodic assumption means that the task resets and restarts after the agent takes T actions, the agent and the key are reset to a random position in the left room, and the green cell is randomised in the right room. If a goal is represented by the $x - y$ coordinate of a cell, the set of all the coordinates of the cells in the right room constitutes the set of possible desired goals for this task. When a new episode starts, a desired goal, i.e., a green cell is uniformly sampled from this set of desired goals, i.e., all cells in the right room. Because the goal representation is the coordinate of the cell, there is always a coordinate value at a cell. Thus, the agent always achieves a goal – not necessarily the desired one – at a state. Finally, the agent will always have a reward of -1 except if it lands on the green cell. This can be measured by simply taking the arithmetic difference

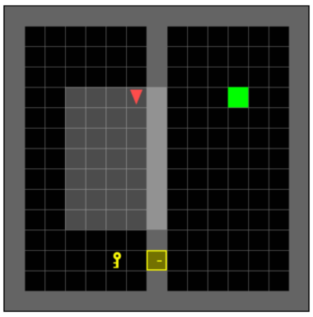


Figure 4.2: The Grid-KeyDoor problem.

between the achieved and the desired goals (coordinates).

As illustrated in subsection 4.2.1, standard RL algorithms such as DQN, DDPG and SAC can be modified straightforwardly to learn these tasks by including an extra input for the goal into the value function or policy (Eq. 4.3, 4.4, 4.5 and 4.6). Also, we show in subsection 4.2.1 that the HER method can be applied to relabel the desired goals of past experiences to help the agent to learn from its failures. As demonstrated by Andrychowicz *et al.* (2017), the use of HER substantially improves the sample efficiency of GRL training in the face of reward sparsity. However, according to later research (Gupta *et al.*, 2019; Fang *et al.*, 2019b) and the experiments in this chapter, HER alone is not sufficient if the desired goal is too far away from the RL agent's initial states. As such, for long-horizon sparse reward GRL problems, the next two subsections provide two techniques to improve the sample efficiency over HER.

4.2.3 Abstract Demonstration

As mentioned, the idea of abstract demonstrations is to leverage task decomposition and provide the correct sequence of subtasks or skills as training guidance. The main benefit is to avoid the labour-intensive process of kinesthetic demonstration collection – a costly process for robots (Ramírez *et al.*, 2022). It also reduces the human bias that may be embedded into the learnt behaviours when using kinesthetic demonstrations.

However, for standard RL algorithms, it is difficult to train a policy that optimises towards a set of reward functions corresponding to the subtasks. It is even more unnatural to gradually reduce the influence of some subtasks and let the policy optimise to achieve the final task. Optimising one policy towards different reward functions makes the RL learning objective inconsistent. An alternative would be training and chaining a set of policies for all subtasks, however, this is less desired due to the cost of parameter storage and waste of resources. Therefore, it is still more desirable to have a single policy that learns several subtasks, potentially sharing knowledge about the task context, environment and behaviour patterns.

Fortunately, GRL provides a natural framework for goal-based skill learning and knowledge sharing (Sutton *et al.*, 2011; Schaul *et al.*, 2015). In GRL, the policy is optimised towards the same reward function, while what may change is the sampling distribution of the desired goal. This rules out the inconsistency problem that occurs in the standard RL framework when different subtasks are to be optimised in some orders. To achieve this, the following additional assumptions need to be made:

- A task decomposition scheme is available to divide the task into a total of I subtasks.
- For each subtask, indexed by $i \in \mathbb{N}, i \leq I$, there exists a non-empty subset of the goal space covering all the desired goals for that subtask: $\mathcal{G}_i^+ \neq \emptyset$ and $\mathcal{G}_i^+ \in \mathcal{G}$.
- One of the subtasks (normally the last one) induces a desired goal space that is identical to that induced by the original (or the final) task $\mathcal{G}_i^+ = \mathcal{G}^+$ where $i \in \mathbb{N}, i \leq I$.
- There exists a sequence of trajectories $\{\tau_j\}, j \in \mathbb{N}$ that connects the initial state $s_0 \sim p(s_0)$ and the state region associated with the final subtask goal, passing through a sequence of subsets of states and goals associated with subtasks. In other words, $\forall\ \{\tau_j\},\ j \in \mathbb{N}$, such that $p(s_{end}^+|s_0, \tau_0, \tau_1, ...) > 0$ and $\sum_{j=0} |\tau_j| \leq T$, where $s_{end} = m^{-1}(g_I^+)$, $g_I^+ \sim \mathcal{U}(\mathcal{G}_I^+)$ and $|\tau|$ denotes the length of the trajectory.

Using the grid world example in Figure 4.2, these assumptions can be implemented as follows. For this simple task, we can easily decompose it into three subtasks including reaching the key, the door and then the green cell. Practically, the subgoal spaces for these three subtasks are the cells in the left room, the wall in the middle and the right room. Naturally, the last subtask is identical to the original final goal space, i.e., all cells in the right room. Finally, the last assumption is guaranteed with a large enough episode length, as the left room is always connected to the wall in the middle, which is always connected to the right room. Thus, there are many trajectories starting from a cell in the left room and ending at the key, then leading to the door and finally to the green cell. This last assumption about trajectory ensures that the goal space is reachable from the starting position within the maximum length of an episode.

Therefore, the aim of abstract demonstrations is to guide the policy to approach the final goal region gradually, through a series of subgoal regions. An abstract demonstration for a task is assumed to be the optimal sequence of some subgoal spaces that ends at the final goal space and maximises the cumulative rewards. Its optimality should be based on return maximisation, which, because of the given reward function defined above, points to the shortest path in the space of subtasks. It can be simply represented by a sequence of integers, each associated with a subtask: $\{x^*_n\}$ where $n \in \mathbb{N}$ is the index of the integer series, $x^*_n \in \mathbb{N}$ is the demonstrated index of the subtasks and $x^*_n \leq I$. To actually guide the policy in training, desired subgoals are sampled from the subgoal spaces in the order indicated by $\{x^*_n\}$. When a desired subgoal is achieved according to the threshold δ_d, the next subgoal is sampled from the next subgoal space according to $\{x^*_n\}$.

Following the grid world example, the abstract demonstration will simply be represented as $\{1, 2, 3\}$ where each integer denotes the index of a subtask. Here, it tells the agent to first reach the key, then the door, and then the green cell. When an episode starts, the agent is given the coordinate of the key as the desired goal. When the agent reaches the key, its desired goal is replaced by the coordinate of the door. Then, it changes to the coordinate of the green cell when the agent reaches the door.

Trajectory extrapolation

Only by following the abstract demonstrations, a goal-condition policy can learn to produce the optimal trajectory that connects each consecutive pair of subgoal spaces. However, it cannot learn to produce a full trajectory from the start to the final goal without such guidance. In other words, it cannot find the path when given only the final goal. The trajectories learnt for different desired subgoals need to be connected for the final goal. To address this, when the policy achieves a desired subgoal, the trajectory that leads to it is duplicated, denoted as $\tau'_j = \tau_j$. For each of its transitions, the desired subgoal is replaced by the next desired subgoal $g^+_{x^*_n} = g^+_{x^*_{n+1}}$, where $g^+_{x^*_n} \in \xi$, $\forall \xi \in \tau'_j$. It then is used as the next trajectory $\tau_{j+1} = \tau'_j$. New transitions collected while achieving the next subgoal are appended to the new trajectory. All trajectories are processed by HER and appended to the replay buffer. As such, when the policy is able to reach the final goal region, it will be able to recognise that the previous trajectories are all valid for achieving the desired final goal.

We can continue with the grid example for a practical illustration. Assume that the agent now has successfully reached the key and thus achieved the first subtask according to the abstract demonstration, resulting in a trajectory of 5 steps: $\tau_0 = \{(s_0, a_0, g^+_1, s_1, a_1), ..., (s_4, a_4, g^+_1, s_5, a_5)\}$. This trajectory is then processed by HER and stored in the replay buffer. The agent will be given the next subgoal g_2^+ according to the demonstration. Without trajectory extrapolation, the agent will start collecting new transitions from s_5 to fill up a new empty trajectory. With trajectory extrapolation, τ_0 is copied and the desired goals of its transitions are replaced by g_2^+. The agent will then continue taking actions to achieve the new subgoal and new transitions will be appended into the new trajectory τ_1. For instance, at the sixth step, $\tau_1 = \{(s_0, a_0, g_2^+, s_1, a_1), ..., (s_5, a_5, g_2^+, s_6, a_6)\}$. This operation happens again when the second subgoal is achieved. The agent will thus be able to relate the necessary states and actions required to achieve the final goal starting from s_0, instead of from the state where the previous subgoal is achieved.

Demonstration proportion

Another implementation detail of abstract demonstrations is when to use them. Instead of applying demonstrations throughout training, it is better to allow the policy to collect random exploration data from time to time. This is suggested by many previous works (Sutton and Barto, 2018; Ladosz *et al.*, 2022), and confirmed by experiments in the next section. Therefore, a parameter, $\eta \in [0, 1]$, is used to control how many demonstrations are applied during training. One may see η as a probability and sample from a Bernoulli distribution to determine whether or not to use demonstrations. Another way is to see it as a proportion parameter and calculate how many episodes are to be demonstrated based on the total number of training episodes. The following experiments adopt the second way. The best value of η needs to be decided by examining the results of experiments with a range of values (subsection 4.3.2).

4.2.4 Adaptive Exploration

This subsection illustrates the idea of adapting exploration according to subtask performances. Recall that, in section 3.2, the DQN, DDPG and SAC algorithms come with different exploration strategies. What is in common is their randomness depends only on the number of passed environment steps. In the following contents, these strategies will be modified to be dependent on a I-dimensional vector of subtask performance metric, denoted as $\boldsymbol{S} \in \mathbb{R}^I$. Each element, S, of it relates to the individual performance of each subtask.

Performance metric

It is essential to choose a good metric for evaluating the performance of an agent. In RL, the average return and success rate are the most common. In order to adapt exploration, a variable in $[0, 1]$ is preferred because it is natural to be used as a scale factor. Thus, the average success rate is a natural choice. If one chooses to use the average return, it would require normalisation, because its scale varies for different tasks. This could be done by calculating the maximal return G_{max} and using it as a normalisation factor to divide any return value achieved by an agent, that is $G_{normalised} = \frac{G_{original}}{G_{max}}$.

To obtain a good estimate of the performance, multiple evaluation runs are required for each subtask to calculate the average value. The number of runs is commonly user-specified. For instance, the following experiment will evaluate the policy on each subtask for 30 episodes. It is recommended to use a larger number of evaluation episodes as it gives a more accurate estimate. However, more runs mean more computations and it takes a large number of runs to get an accurate estimate of the agent's performance. To cope with this issue, the Polyak average (Scieur and Pedregosa, 2020) is used to estimate the performance instead of the arithmetic average. This is inspired by how the target networks in DDPG and SAC are updated (Lillicrap *et al.*, 2015; Haarnoja *et al.*, 2018). A typical equation would be:

$$\overline{S}_i = (1 - \tau_S)\overline{S}_{i-1} + \tau_S S_i \tag{4.7}$$

where $\overline{S}_i$ is the Polyak-averaged estimate after the i-th evaluation, S_i is the arithmetic average, and $\tau_S \in [0, 1]$ is the update rate. S_0 and $\overline{S}_0$ are normally initialised to 0. Replacing the arithmetic average with the Polyak average brings a smoothing effect on the changes in the performance metric and a more accurate estimate over time.

Specific implementations

Given a good estimate of the performance of the subtasks, there are different ways to use them to influence the exploration behaviours of an RL algorithm. The following will illustrate how this can be done with the DQN, DDPG and SAC agents. The three are representative RL algorithms for deterministic discrete action tasks, deterministic continuous action tasks and stochastic continuous action tasks.

DQN, as introduced in subsection 3.2.2, normally explores the environment using the ϵ-*greedy* strategy with a linear or exponential decay scheme. In simple words, a DQN agent takes a uniformly random action with probability, ϵ_{dqn}, otherwise takes the action with respect to the maximal q value according to the current q function. The probability ϵ_{dqn} is decayed to a lower bound over the course of training. For example, an exponential decay scheme can be written as: $\epsilon_{dqn} = \epsilon_{end} + (\epsilon_{start} - \epsilon_{end})e^{\frac{-k}{\beta}}$, where ϵ_{start} and ϵ_{end} are the upper and lower bounds, k is the total elapsed environment timesteps and β is the decay coefficient parameter. Intuitively, ϵ decays as the number of elapsed timesteps grows and its speed is controlled by β.

To make it adaptive, the exponential term is to be replaced by the performance metric. For I subtasks, after every new performance estimate is calculated with Eq. 4.7, the following update rule is used to change their exploration parameters:

$$\boldsymbol{\epsilon}_{dqn} = \epsilon_{end} + (\epsilon_{start} - \epsilon_{end})(1 - \overline{\boldsymbol{S}}) \tag{4.8}$$

where $\boldsymbol{\epsilon}_{dqn} \in [0, 1]^I$ is a I-dimensional vector, whose elements are the exploration parameters for the ϵ-*greedy* strategy associated with different subtasks.

DDPG, as introduced in subsection 3.2.3, uses a behavioural policy that explores the environment with a mixture of random and noisy actions. In this experiment, the baseline exploration strategy is the continuous version of the ϵ-*greedy* method, named ϵ-*Gaussian*. It has been widely used to replace the Ornstein-Uhlenbeck exploration strategy proposed in the DDPG paper (Lillicrap *et al.*, 2015; Andrychowicz *et al.*, 2017; Fujimoto *et al.*, 2018). In short, the ϵ-*Gaussian* strategy takes a uniformly random action with a probability ϵ_{ddpg}, otherwise takes the action produced by the current policy with added noise sampled from a zero-mean Gaussian distribution. Different from the DQN case, ϵ_{ddpg} is commonly non-decaying and set to a small value. This behavioural policy for the goal-conditioned DDPG can be written as:

$$\pi_b(a|s, g) = \begin{cases} a \sim \mathcal{U}(\mathcal{A}), & z \leq \epsilon_{ddpg} \\ a \sim \mathcal{N}(\pi(s, g^+), \sigma_{ddpg}), & z > \epsilon_{ddpg} \end{cases} \tag{4.9}$$

where $\mathcal{U}$ denotes a uniform distribution, $\mathcal{N}$ denotes a Gaussian distribution, $z \sim \mathcal{U}(0, 1)$ and $\pi(s, g^+)$ is a learnt, deterministic, goal-conditioned policy. Similar to what is done with the DQN above, the probability ϵ_{ddpg} and the variance of the Gaussian noise σ_{ddpg} can be made adaptive using the performance estimate. After every update to the parameters, a new behavioural policy can be deduced to collect experiences until the next evaluation of the performances. Given the upper bounds of the both parameter ϵ_{upper} and σ_{upper}, after every new performance estimate is calculated with Eq. 4.7, they are to be updated by:

$$\boldsymbol{\epsilon}_{ddpg} = \epsilon_{upper}(1 - \overline{\boldsymbol{S}}), \quad \boldsymbol{\sigma}_{ddpg} = \sigma_{upper}(1 - \overline{\boldsymbol{S}}) \tag{4.10}$$

SAC, as introduced in subsection 3.2.4, explores the environment with actions sampled from its own stochastic policy (Haarnoja *et al.*, 2018). There is no other behavioural policy with a parameter to be modified. The mean and deviation of the policy distribution are normally generated by a neural network. Therefore, for the SAC algorithm and many others with a stochastic policy, the exploration can be made adaptive according to its performance by altering its deviation. Denote the deviations produced by the SAC policy for the I subtasks as $\tilde{\boldsymbol{\sigma}}_{sac}$, it is adjusted by:

$$\boldsymbol{\sigma}_{sac} = \tilde{\boldsymbol{\sigma}}_{sac} \odot (1 - \overline{\boldsymbol{S}}) \tag{4.11}$$

where $\odot$ denotes element-wise vector product. So far, this subsection has illustrated how to make the ϵ-*greedy*, ϵ-*Gaussian* and stochastic policy explorations become adaptive with respect to the performance of the policy. There are many other exploration strategies that this thesis cannot cover (Ladosz *et al.*, 2022), but similar implementations should be straightforward. Here are two more examples:

- *Noisy network* inserts Gaussian noises into the parameters of a neural network Plappert *et al.* (2020). In this work, the authors propose to perturb the network parameters at the beginning of each episode with noises sampled from a Gaussian distribution. A straightforward way to integrate with the proposed adaptive exploration module is to scale the standard deviation of this distribution with $(1 - \overline{\boldsymbol{S}})$, such that the network is noisier when the performance is not good and vice versa.

- *Intrinsic exploration.* A large body of works focuses on intrinsically motivated exploration Aubret *et al.* (2019). They mostly require a weight factor to control the importance of curiosity reward relative to the original extrinsic reward. Intuitively, it determines whether an agent explores or exploits more. Our adaptive exploration module can also be integrated into such exploration methods to scale the weight factor during training, releasing users from manual fine-tuning.

4.2.5 Summary

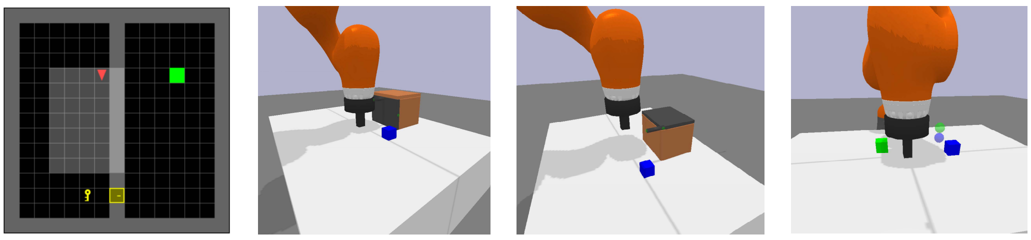
In sum, this section describes the problem assumptions and formally proposes the $\mathbf{A^2}$ method: abstract demonstrations and adaptive exploration, along with how to implement them in detail. In simple words, abstraction demonstrations provide a learner policy with the correct sequence of subtasks, with a goal-relabelling trick to enable the learning of long-horizon trajectories. Adaptive exploration estimates the success rates of a policy and scales its exploration parameters for different subtasks. The Above has shown the implementations for DQN, DDPG and SAC in detail, with a brief discussion of other exploration methods. The pseudo-code of the training process for the goal-augmented DQN agent is summarised in Algorithm 1. The pseudo-codes goal-augmented DDPG and SAC with $\mathbf{A^2}$ can be easily deduced from it. The following section will discuss more on implementations and the results of empirical studies on the effectiveness of $\mathbf{A^2}$.

Algorithm 1 Goal-augmented DQN with $\mathbf{A^2}$

```
Input: maximum epochs, cycles, episodes $M_0, M_1, M_2$
Input: demonstration proportion $\eta$, adaptive exploration update ratio $\tau_S$
for $epoch = 1$ to $M_0$ do
    for $cycle = 1$ to $M_1$ do
        for $episode = 1$ to $M_2$ do
            Reset task, sample a final goal
            $use\_demonstrations \leftarrow False$
            if $episode \leq \eta M_2$
                $use\_demonstrations \leftarrow True$
                Obtain the first correct subgoal as the desired goal
            end if
            for $t = 0$ to $T - 1$ do
                Compute $\boldsymbol{\epsilon}_{dqn}$ with adaptive exploration (Eq. 4.8)
                Sample $a$ with $\epsilon$-greedy for the given subgoal
                Execute $a$ and observe the next state and reward
                Store the transition
                if $desired\_goal\_achieved$
                    Store trajectory
                    if $use\_demonstrations$ (subsection 4.2.3)
                        perform trajectory extrapolation
                        Obtain the correct next subgoal as the new desired goal
                    end if
                end if
            end for
        end for
        Perform $HER$ on the trajectories, perform DQN update (Eq. 4.3)
    end for
    Evaluate $\pi$ for each subtasks,
    Update the adaptive exploration parameters (Eq. 4.10)
end for
```

4.3 Empirical Results

To investigate the effect of the proposed $\mathbf{A^2}$ method, a series of simulation experiments have been conducted with the Mini-Grid (Chevalier-Boisvert *et al.*, 2018) and the Pybullet Multigoal (PMG) environment developed by the author (see publication [2]). All performances were measured by the success rate of achieving the final goal without demonstrations, averaging over five random seeds. This section will first introduce the tasks and some implementation details of the policies, and then discuss the empirical results.



(a) GridDoorKey (b) ChestPush (c) ChestPickAnd-Place (d) BlockStack

Figure 4.3: Experiment tasks. (a) The agent (red) should pick up the key, open the door and reach the goal cell (green). (b-c) The robot should open the grey door of the chest, and push or pick-and-drop the blue lock into the chest. (d) The robot should pick and stack the blocks at a random position in a random order, indicated by the transparent spheres.

4.3.1 Task and implementation details

To test the proposed $\mathbf{A^2}$ methods, six multi-step, sparse reward tasks were chosen, including GridDoorKey (3 sizes), ChestPush, ChestPickAndPlace and BlockStack (see Figure 4.3 for a visualisation). The grid world tasks are for testing the DQN agent with discrete actions, while the robotic tasks are for the DDPG and SAC agents with continuous actions. Source codes are available on https://github.com/IanYangChina/A-2-paper-code. The following will briefly introduce the MDP definitions for these tasks, then the optimisation details and baselines.

GridDoorKey task

Formally, the MDP for the GridDoorKey task can be written as:

$$\begin{aligned}
\mathbf{s} \in \Big\{ &(x_{agent}||y_{agent}||x_{key}||y_{key}||x_{door}||y_{door}||b_{head}||b_{key}||b_{door}) \Big| \\
&x_{agent}, y_{agent}, y_{key}, y_{door} \in \{1 \,..\, grid\_size - 2\}, \\
&x_{key} \in \{1 \,..\, \frac{grid\_size - 1}{2} - 1\}, x_{door} = \frac{grid\_size}{2}, \\
&b_{key} \in \{0, 1\}, b_{door} \in \{0, 1\}, b_{head} \in \{0, 1, 2, 3\} \Big\}
\end{aligned}$$

$$\mathbf{g} \in \Big\{ m(\mathbf{s}) = (x_{agent}||y_{agent}) \Big| x_{agent}, y_{agent} \in \{1 \,..\, grid\_size - 2\} \Big\}$$

$$\mathbf{a} \in \{\text{move\_forward, turn\_left, turn\_right}\}$$

$$r(\mathbf{g}, \mathbf{g}^{+}) = \begin{cases} 0, \text{ if } \mathbf{g} = \mathbf{g}^{+} \\ -1, \text{ otherwise} \end{cases}$$

where $||$ denotes vector concatenation, $\{a \,..\, b\}$ denotes a set of integers between a and b included, b_{head} indicates the four heading directions, $grid\_size$ will be 15, 25 or 35 depending on the size of the grid map. The state is a vector composed of the x and y coordinates of the agent, the key and the door, the heading direction of the agent, and two binary variables indicating whether the agent is carrying the key and whether the door is opened. The goal is represented by the x and y coordinates of the agent. This means that an achieved goal is the coordinate of whichever cell the agent is in, and a desired goal is the coordinate of the cell that the agent needs to reach.

The reward function gives a value of 0 when the achieved goal and the desired goal is identical at a state, and -1 otherwise. The policy has three actions to select at each timestep: turn left, turn right and move one cell forward. When it stamps on the cell with a key, it automatically picks up the key. When it moves forward to the door, the door opens if it has the key, otherwise, nothing happens.

For cases without demonstrations, the agent is given the location of the green cell as the desired goal $\mathbf{g}^+_{final}$. For cases with abstract demonstrations, the task is divided into three subtasks: reach the key $\mathbf{g}^+_1$, reach the door $\mathbf{g}^+_2$ and reach the green cell $\mathbf{g}^+_3 = \mathbf{g}^+_{final}$. The agent is given subsequently the three subgoals. Formally, the set of final goals and the three sets of subgoals are:

$$\mathbf{g}^+_1 \in \big\{(x_{key}||y_{key})\big|x_{key} \in \{1 \;..\; \frac{grid\_size-1}{2} - 1\}, y_{key} \in \{1 \;..\; grid\_size - 2\}\big\}$$
$$\mathbf{g}^+_2 \in \big\{(x_{door}||y_{door})\big|x_{door} = \frac{grid\_size-1}{2}, y_{door} \in \{1 \;..\; grid\_size - 2\}\big\}$$
$$\mathbf{g}^+_3, \mathbf{g}^+_{final} \in \big\{(x_{final}||y_{final})\big|x_{final} \in \{\frac{grid\_size-1}{2} - 1 \;..\; grid\_size - 1\},$$
$$y_{final} \in \{1 \;..\; grid\_size - 2\}\big\}$$

where $(x_{final}||y_{final})$ denotes the coordinate of the final target cell.

At the beginning of each episode, the agent is placed randomly in the left room, the final goal location (the green cell) is generated uniformly in the right room, the key is placed randomly in the left room, the door is generated uniformly randomly on the wall in the middle. Three sizes of the task were run, including a 15×15, a 25×25, and a 35×35 grid. The agent has a total of 40, 50 and 70 timesteps per episode for the task of each size. Instead of the final goal $\mathbf{g}^+$, the agent with demonstrations will be given $\mathbf{g}^+_1$ as the first desired goal. When it reaches the cell where the key is, $\mathbf{g}^+_1$ is achieved and it is given $\mathbf{g}^+_2$ as the desired goal. When it reaches the door, the desired goal is then replaced by $\mathbf{g}_3^+$, i.e., the final goal.

Manipulation tasks

The ChestPush and ChestPickAndPlace tasks with one block and the Block-Stack task with two blocks from the PMG environments are used for empirical studies (see publication [2] for more details). Formally,

The state representation is comprised of the block state, the gripper state and the chest state (if a chest is involved), i.e., $\mathbf{s} \in \{(\mathbf{s_{block}}||\mathbf{s_{grip}}||\mathbf{s_{chest}})\}$. *The block state* consists of the linear positions and Euler orientations of all blocks in the world frame, the relative positions of all blocks with respect to the gripper tip frame, and the relative linear and angular velocities of all blocks with respect to the gripper tip frame. *The gripper state* consists of the linear position, Euler orientation and linear velocity of the gripper tip frame, the velocities of the two fingers, and the width between the two fingers. *The chest state* consists of the opened width and velocity of the door, and the position of the three keypoints of the door in the world frame (shown in Figure 4.4).

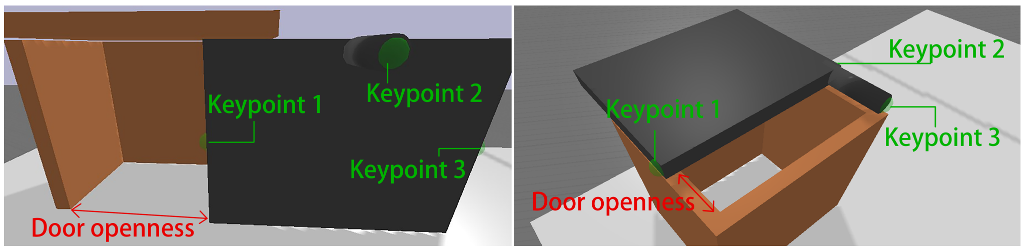


Figure 4.4: The chest state representation for task ChestPush and Chest-PickAndPlace.

At the beginning of each episode, the robot gripper is positioned top-down at the centre of the table. The gripper tip frame is 0.75 m above the table surface. The policy controls the gripper by displacement in the Cartesian space at most 0.002 m per timestep. The actual action output by the policy is a continuous three-dimensional vector in $[-1.0, 1.0]^3$, which is multiplied by 0.002 when applied to the robot controller. For the ChestPickAndPlace and the BlockStack tasks, the action has a fourth dimension, associated with the target width between the gripper fingers, normalised into $[-1.0, 1.0]$ as well.

The goal representation for all tasks consists of the positions of the blocks and the gripper tip in the world frame, the width between the fingers (if grasping is required), and the opened width of the chest (if a chest is involved), i.e., $\mathbf{g}^{+} \in \{(x_{block}||y_{block}||z_{block}||x_{grip}||y_{grip}||z_{grip}||width_{finger}||width_{chest})\}$. For the ChestPush and ChestPickAndPlace tasks, the final goal is the desired opened width of the door and the desired position of the block, which is at the centre of the chest. For the BlockStack task, the final goal includes the positions of the two stacked blocks placed at a random location in a random order, visualised by the transparent spheres in Figure 4.3 (d). Each episode of the ChestPush task has a total of 30 timesteps, and that of the other two tasks has a total of 50 timesteps. The reward function is sparse, giving a value of 0 when a goal is achieved and -1 otherwise. A desired goal is deemed achieved when its Euclidean distance with the achieved goal is less than 0.03 m.

If abstract demonstrations are used in an episode, the agent will be given subgoals according to the demonstrations. The decomposition schemes for the tasks are as follows:

- ChestPush: three subgoals including 1) opening the chest door, 2) reaching the block and 3) pushing the block into the chest;
- ChestPickAndPlace: four subgoals including 1) opening the chest door, 2) grasping the block, 3) moving to the top of the chest and 4) dropping

the block into the chest;

- BlockStack: four subgoals including 1) grasping the base block, 2) moving the base block to the target location, 3) grasping the second block and 4) stacking the second block on top of the base block.

Training details

The tasks are run in the same experiment schedule as (Andrychowicz *et al.*, 2017), organised into *epoch*, *cycle* and *episode*. Each epoch has 50 cycles, each of which has 16 episodes. Different tasks grant the agent with a different number of total timesteps per episode (see the above task specifications). For the three GridDoorKey tasks, the policy is trained for 30, 50 and 70 epochs. For the ChestPush, ChestPickAndPlace and BlockStack tasks, the policy is trained for 30, 50 and 100 epochs. The maximum number of training epochs are selected to be sufficiently large for the algorithm to converge, which in practice means that the evaluation success rates stabilise for at least 5 epochs. HER with the future strategy (see subsection 4.2.1) is applied to the trajectory collected at the end of every episode. If abstract demonstrations are used, HER is applied to all trajectories generated for every subgoal that occurred in the episode (see "trajectory extrapolation" in subsection 4.2.3).

To evaluate the policy, 30 episodes are tested without any exploration after every epoch (the SAC policy uses the output mean as the action). For agents with adaptive exploration, each subgoal is evaluated for 30 episodes to obtain the arithmetic average success rates, after which the exploration parameters are updated according to subsection 4.2.4.

The following training parameters and neural network architectures used in the experiments are again standard for state-based DRL, especially based on the papers of the original algorithms (Lillicrap *et al.*, 2015; Andrychowicz *et al.*, 2017; Plappert *et al.*, 2020; Fujimoto *et al.*, 2018; Haarnoja *et al.*, 2018). Each algorithm uses a replay buffer of size $1e6$. After each episode, the algorithm updates their neural networks 40 gradient steps, each of which with a different mini-batch of size 128 sampled from the replay buffer. Neural network optimisation is done with the Adam optimiser (Kingma and Ba, 2014) with a learning rate of 0.001. For the DDPG and SAC algorithms, the actor networks are updated after the critic networks. For the critic and actor, each takes one gradient step with the same mini-batch. The target networks for DQN and DDPG are updated once with Polyak averaging (Eq. 3.26) with an update rate of 0.05 after the main networks are updated. The target networks for SAC are updated once every two main network updates with the same update rate. All target q value is calculated with a discount factor of 0.98, and clipped within $[-50, 0]$. All observations are normalised using the mean and deviation calculated from all historical observations. The DQN network has three fully-connected layers with sizes 64, 128 and 64. The actor and critic networks of the DDPG and SAC algorithms have three fully-connected layers of sizes 256. The DDPG actor outputs directly a continuous action, while the SAC actor outputs the mean and deviation vectors of the policy distribution. The critic networks output a scalar value. All of them use ReLU activation on each layer, except for the outputs. All actor networks use hyperbolic tangents to activate the final layers and all critic networks (including DQN) do not have output activation functions.

Here, we use the GridDoorKey example to give an illustration of how Algorithm 1 can be implemented. For the task of size 15, we have $M_0 = 30$ epochs, $M_1 = 50$ cycles and $M_2 = 16$ episodes. When the task resets at the beginning of an episode, the initial states are randomly sampled according to the description of the GridDoorKey task earlier in this subsection. Demonstrations are used when the number of episodes is smaller than ηM_2 in a training cycle. If the abstract demonstration is used, the agent is given g_1^+ as the desired goal. At each step, according to $\boldsymbol{\epsilon}_{dqn}$ computed from Eq. 4.8, the agent either takes a random action or the action corresponding to the highest Q value based on $\arg\max_a \tilde{q}_{\boldsymbol{w}^-}(s, a, g^+)$. The transition is then stored in the trajectory. When a desired goal is achieved, if the episode is using abstract demonstrations, the trajectory extrapolation technique (subsection 4.2.3) is performed and a new subgoal is given to the agent. When a training cycle reaches its end, *HER* is performed on the trajectories collected (the future method at the end of subsection 4.2.1). All modified trajectories are pushed into the replay buffer. Then, mini-batch updates are carried out with Eq. 4.3. When an epoch finishes, the agent is evaluated on each subtask for 30 episodes. In order words, it is taking only greedy actions from the Q function and tasked to achieve g_1^+, g_2^+ and g_3^+ respectively for 30 times. The average success rates are used to update the Polyak success rates using Eq. 4.7 and then, the Polyak success rates are used to compute the new exploration parameters for each subtask until the next evaluation run. The training processes of the goal-augmented DDPG and SAC agents in the manipulation tasks are basically the same, except that different subgoals, exploration strategies, and update equations are used.

Baseline and experiment design

To assess the performance gain of the proposed $\mathbf{A}^2$ method, the DQN, DDPG and SAC algorithms are run without abstract demonstrations and adaptive exploration as baselines:

- The baseline DQN uses the exponential decayed ϵ-greedy exploration with $\epsilon_{start} = 1.0$, $\epsilon_{end} = 0.05$ and $\beta = 5e5$. The $\mathbf{A}^2$-aided DQN uses the same start and end probabilities with Eq. 4.8.

- The baseline DDPG uses the non-decaying ϵ-Gaussian strategy (Eq. 4.9) with $\epsilon_{ddpg} = 0.2$ and $\sigma_{ddpg} = 0.05$. The $\mathbf{A}^2$-aided DDPG uses the same values as the upper bounds and adjusts them by Eq. 4.10.
- The baseline SAC explores with actions sampled from the policy distribution computed by the network, while the $\mathbf{A}^2$-aided SAC adjusts the deviations with Eq. 4.11.

All other training details of the baselines remain the same as that for the $\mathbf{A}^2$-aided agents.

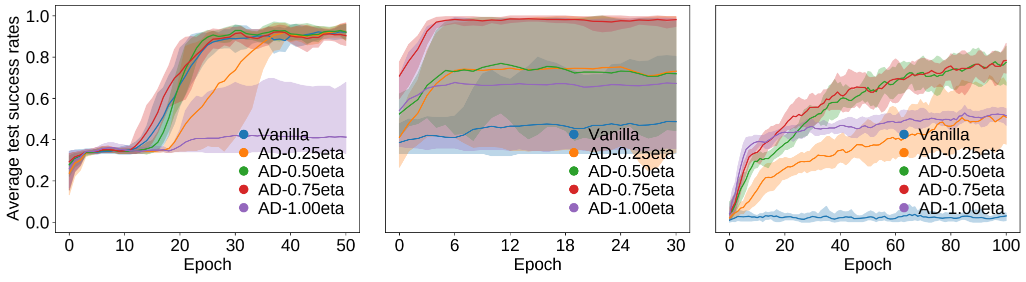
Ablations of the abstract demonstrations and adaptive explorations are conducted first to observe the influence of the parameters η and τ_S. Discrete search for the best values are conducted for $\eta \in [0.25, 0.5, 0.75, 1.0]$ and $\tau_S \in [0.1, 0.3, 0.5, 0.7, 0.9]$. Parameter ablations are run on the GridDoorKey25x25 task (DQN), the ChestPush task (DDPG) and the BlockStack task (SAC) (subsection 4.3.2). The success rate update ratios are evaluated with a fixed demonstration proportion $\eta = 0.75$. To assert the importance of using the Polyak average for estimating the success rates, the comparison is also made between $\mathbf{A}^2$-aided agents with and without the Polyak average in the adaptive exploration update.

The best parameter values are used to compare the baselines without $\mathbf{A}^2$ (tagged by "Vanilla" in the figures below), the ones with abstract demonstrations (tagged by "AD") and the fully $\mathbf{A}^2$-aided agents (tagged by "ADAE"), on all tasks for all agents (4.3.3).

4.3.2 Ablation study

This subsection focuses on examining the parameter values of the $\mathbf{A^2}$ method, including the percentage of episodes η being demonstrated and the adaptive exploration update rate τ_S. Ablations are performed on the GridDoorKey25 tasks with the DQN agent, ChestPush task with the DDPG agent and BlockStack task with the SAC agent.

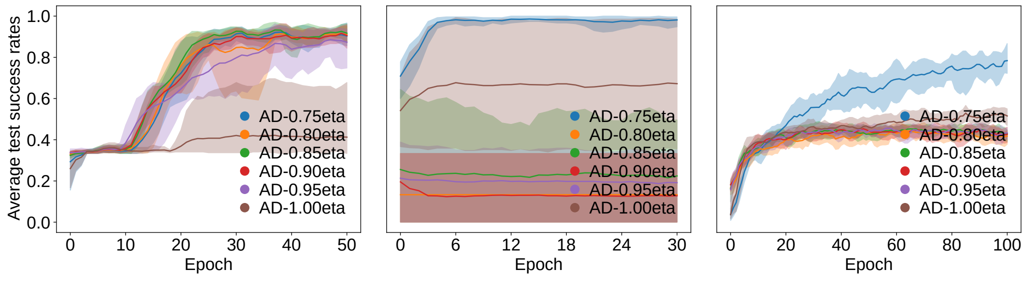
Figure 4.5 shows that, in general, abstract demonstrations improve convergence speeds and performances. Demonstrating 50% or 75% of the training episodes gives the highest performance gains for all cases (green and red lines). Interestingly, giving demonstrations to all episodes does not achieve the best performances or even makes it worse (purple lines). The phenomenon could be caused by a lack of data diversity when all episodes are given demonstrations. The reason behind it is that data with low diversity would cause the neural network to overfit into a narrow distribution other than the real target distribution. In this case, the policy is given too less experiences of exploring from the start given the final goal. This results in low performance in the evaluation of the final task.



(a) GridDoorKey25-DQN (b) ChestPush-DDPG (c) BlockStack-SAC

Figure 4.5: Test success rates with different proportions of demonstrated episodes η. AD: abstract demonstrations.

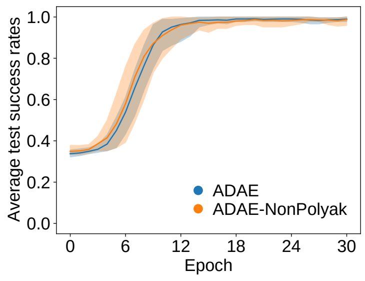
In order to further determine the significance of the value of η, we conducted experiments on a few more values for $\eta \in \{0.80, 0.85, 0.90, 0.95\}$. The results in Figure 4.6 show that for the three agents, $\eta = 0.75$ is the best choice. In addition, the wrong value of η only reduces the average performance of the DQN and SAC agents, but it significantly destabilises the DDPG agent (large variances). This may be due to the brittleness of the DDPG update itself as also reported by other researchers (Zheng12 *et al.*, 2018; Tiong *et al.*, 2020). These results again confirm that too many demonstrations do hurt performances, especially in continuous control tasks, and support that a good value of η tends to locate around 0.75. However, it is recommended to conduct a grid search for the value of η starting from 0.75 for different tasks, as one can see the different effects of this parameter on the three different agents and tasks from Figure 4.5 and 4.6.



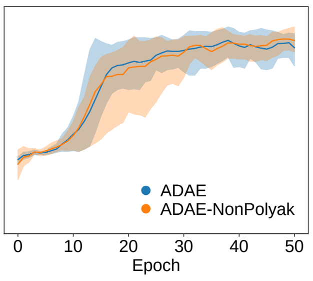
(a) GridDoorKey25-DQN (b) ChestPush-DDPG (c) BlockStack-SAC

Figure 4.6: Test success rates with different proportions of demonstrated episodes η in $\{0.75, 0.80, 0.85, 0.90, 0.95, 1.00\}$. AD: abstract demonstrations.

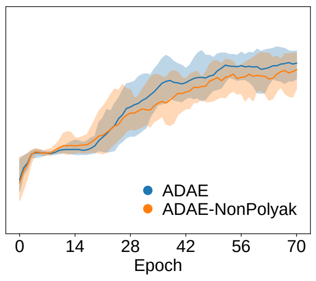
As mentioned in subsection 4.2.4, the Polyak average is used in estimating the policy's success rates over time for the adaptive exploration parameter update. The hypothesis is that the Polyak average will give less jumpy updates of the success rates and the exploration parameters. The comparative results in Figure 4.7 confirm the hypothesis by showing that cases without Polyak averaging gives a performance with higher variances. The reduction of variance is more obvious in longer-horizon continuous control tasks.



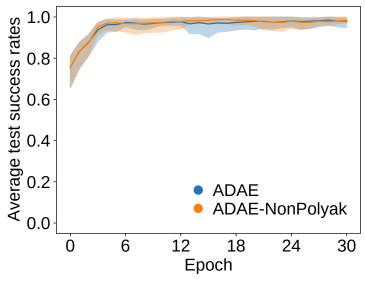
(a) GridDoorKey15



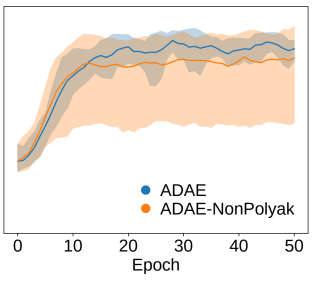
(b) GridDoorKey25



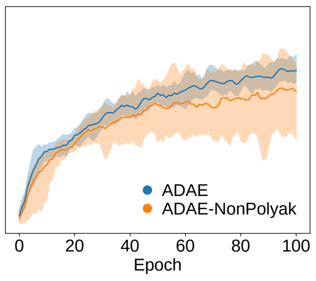
(c) GridDoorKey35



(d) ChestPush



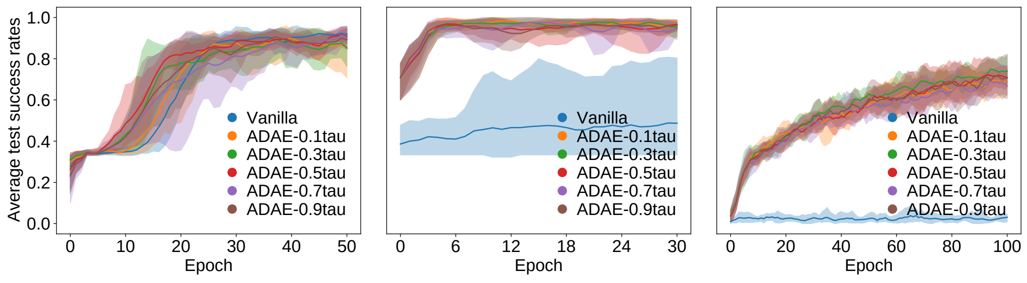
(e) ChestPickAndPlace



(f) BlockStack

Figure 4.7: Test success rates of DQN on gridworld tasks (a-c) and SAC (d-f) on robotic tasks with and without Polyak averaging. All cases are run with 75% demonstrated episodes. *ADAE*: abstract demonstration and adaptive exploration.

Figure 4.8 shows that changing the value of τ_S has trivial effects on the convergence speed as well as the final performance. However, a smaller value may slow down learning as shown by Figure 4.8a. In general, 0.3 will suffice for stabilising learning, but a higher value may be more useful for longer horizon tasks to make the agent progress faster.



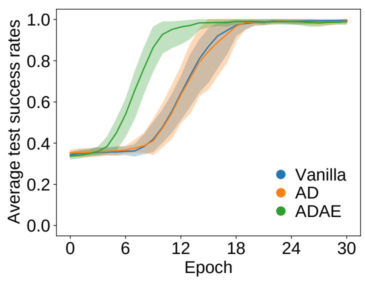
(a) GridDoorKey25-DQN (b) ChestPush-DDPG (c) BlockStack-SAC

Figure 4.8: Test success rates with different success rate update ratio $\tau_{\boldsymbol{S}}$ for adaptive exploration. All cases are run with 75% demonstrated episodes. *ADAE*: abstract demonstrations and adaptive exploration.

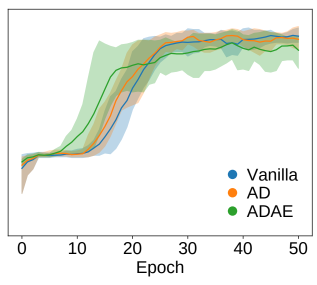
4.3.3 General performance

This subsection focuses on the general improvements improved by the proposed $\mathbf{A^2}$ method on all tasks. According to the ablation results (subsection 4.3.2), it is identified that using abstract demonstrations in 75% of the training episodes and updating the success rates with an update ratio of 0.3 for the adaptive exploration parameter, can achieve better performances over other parameter values. Thus, for all cases compared in this subsection, these two parameter values are used whenever $\mathbf{A}^2$ is applied.

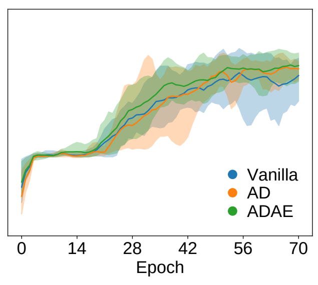
Figure 4.9 shows that, in all experiments, abstract demonstrations help the agent learn faster and achieve higher success rates. This is more obvious in robotic tasks. Notice that as the robotic task becomes more difficult (from subfigures 4.9d to 4.9f), the gap of success rates becomes larger between the algorithms with and without abstract demonstrations (orange and blue lines). This demonstrates that abstract demonstrations can provide vast improvement on multi-step tasks.



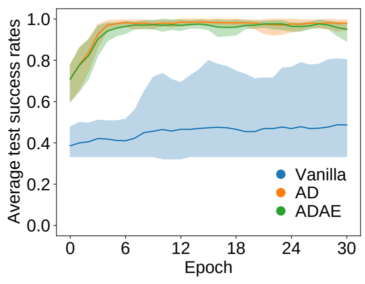
(a) GridDoorKey15



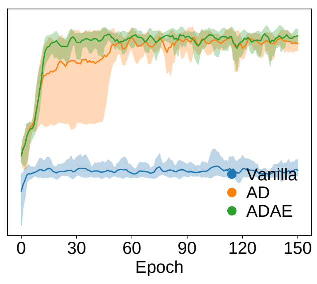
(b) GridDoorKey25



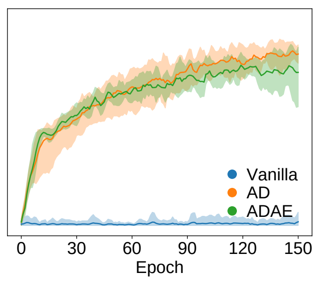
(c) GridDoorKey35



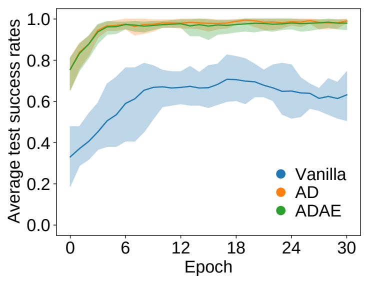
(d) ChestPush



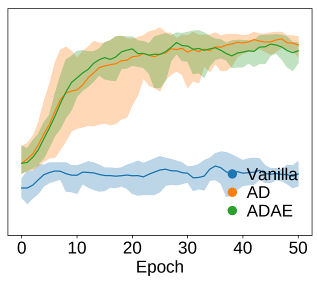
(e) ChestPickAndPlace



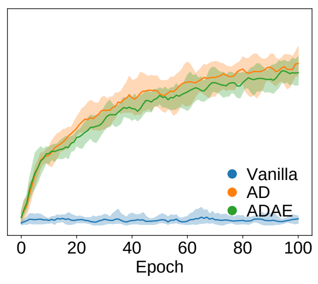
(f) BlockStack



(g) ChestPush



(h) ChestPickAndPlace



(i) BlockStack

Figure 4.9: Test success rates of DQN on gridworld tasks (a-c), DDPG (d-f) and SAC (g-i) on robotic tasks. *AD*: abstract demonstrations; *ADAE*: abstract demonstrations and adaptive exploration.

On the other hand, adaptive exploration only has a slight improvement in the overall success rate or learning speed on top of abstract demonstrations. Nonetheless, it clearly reduces the variance of the learning performances, especially for the robotic tasks (green lines). This result abides with the hypothesis that reducing unnecessary exploration helps the policy act more decisively on well-mastered subtasks and progress/learn faster on later subtasks.

4.4 Summary

To conclude, this chapter proposes two techniques, *abstract demonstrations* and *adaptive exploration*, to accelerate RL algorithms in long-horizon, multi-step and sparse reward robotic manipulation tasks. In short, abstract demonstrations guide an RL policy to achieve subtasks leading to the final goal, while adaptive exploration alters the policy's exploratory behaviour according to its online performances. Named $\mathbf{A}^2$, the proposed method is developed under the framework of GRL, because of its advantages of multigoal learning, knowledge sharing and overcoming the sparse reward issue (Schaul *et al.*, 2015; Andrychowicz *et al.*, 2017). The implementations of $\mathbf{A}^2$ on three popular RL algorithms (DQN, DDPG and SAC) are illustrated in detail in section 4.2. Experiments confirm that abstract demonstrations can improve the convergence speed and overall performance substantially, while adaptive exploration helps in the reduction of performance variance.

The limitations and future developments of $\mathbf{A}^2$ come from its assumptions. First of all, it depends on a well-defined task decomposition scheme to specify the subtasks. This will become a bottleneck when such a task decomposition scheme is difficult to obtain. How the method can be used with learnt subgoals is a valuable future direction. Secondly, the representation of goals in this chapter is based on system states. It remains unclear how it may work when the goals are to be defined on raw sensory observations such as images and point clouds.

Lastly, $\mathbf{A}^2$ does indeed improve the learning of such long-horizon and multi-step tasks, but it is arguable that such end-to-end manipulation policies are less desirable in the real world. As discussed in section 2.4, systems with hierarchies are preferred because the burden of learning a whole task is separated into different modules, and the reusabilities of the submodules are improved. Therefore, the next chapter will take a hierarchical approach towards solving such multi-step manipulation tasks, with the aim of reusing a policy to achieve multiple final outcomes.

Chapter 5

Universal Option Framework for Multi-outcome Multi-step Robotic Manipulation

5.1 Introduction

Hierarchical control systems are highly preferred in solving long-horizon and multi-step manipulation tasks since they can solve such tasks by decomposing them into subtasks whose solutions are easier to compute (Garrett *et al.*, 2021). The advantages of using system hierarchy are supported by the evidence of hierarchical decision-making mechanisms existing in the human brain (Grafton and Hamilton, 2007). Also, the long-lasting research field of hierarchical reinforcement learning (HRL) echoes the same opinion, with various applications on robotics (Pateria *et al.*, 2021b).

The specific tasks investigated in this chapter are the long-horizon, multi-step and multi-outcome manipulation tasks. In particular, a series of block stacking tasks, including an example shown by Figure 5.1, will be used for algorithm evaluation. These tasks require the robot to use a number of subtasks or skills to achieve different outcomes, such as different orders of stacked blocks.

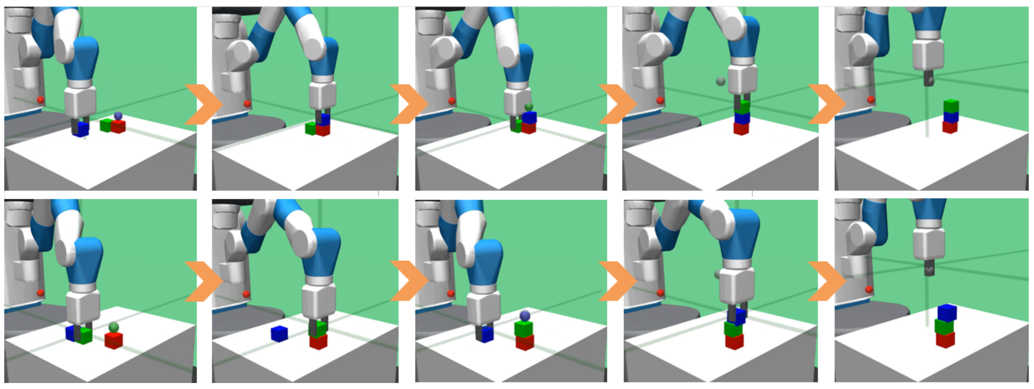


Figure 5.1: A block stack task where the robot needs to stack three blocks in different orders.

In robotics, a real-world manipulation system is normally modularised into two levels, addressing the subtask planning and motion generation problems separately. Among these methods, classic ones rely on a known system dynamic model to plan for a sequence of subtasks and the manipulator motions that solve them (Garrett *et al.*, 2021). However, as elaborated in subsection 2.4.1, it becomes too difficult to obtain an accurate dynamic model when a task involves interaction with many objects. The contact processes are hard to specify and simulate. Therefore, modern learning-based methods instead seek to use data to learn reactive policies at the both task planning and motion generation levels. Nevertheless, the general architecture design of the system to a large extent remains unchanged: a task planning and a motion generation module. It is the contents that are changed from model-based planning to model-free reactive policy learning (Pateria *et al.*, 2021b).

State-of-the-art learning-based HRL systems mostly employ the one-to-many architecture, where a high-level policy uses a number of low-level policies to achieve a final task (Tessler *et al.*, 2017; Barreto *et al.*, 2019; Yang *et al.*, 2020; Hakhamaneshi *et al.*, 2022; Pateria *et al.*, 2021b). As discussed in detail in subsection 2.4.2, it is undeniable that such systems are able to tackle long-horizon and multi-step manipulation tasks. However, there are two limitations that this chapter seeks to improve.

First of all, the low-level policies in these systems normally correspond to a set of subtasks or manipulation skills. For example, picking, moving an object, dropping, pushing, etc. Assigning one skill per policy means that these policies will take up a large amount of storage space in the computer when there are many subtasks or skills, especially when the policies are represented by large neural networks. In addition, many subtasks or skills exhibit similar motion patterns, implying the potential of sharing knowledge and understanding among the tasks or skills. The same logic applies to the high-level planning policy as well. When there are many outcomes that can be achieved by using the same set of skills, state-of-the-art systems would need to retrain the high-level policy. This again will cause a waste of computer resources that may be avoided.

In response, this chapter proposes the *universal option framework (UOF)*, which explores the idea of embedding goal-conditioned policies into hierarchical control systems. In other words, this chapter proposes to make the high-level planning and low-level motion generation policies become goal-conditioned (Schaul *et al.*, 2015; Andrychowicz *et al.*, 2017), such that there is only one policy at each level to achieve multiple goals, improving memory usage and skill reusability. This idea, which may be called hierarchical goal-condition reinforcement learning, is not a completely new idea by the time this research was conducted. Several papers have discussed the possibility (Nachum *et al.*, 2018; Levy *et al.*, 2019; Jiang *et al.*, 2019; Dilokthanakul *et al.*, 2019). There are several limitations or differences that these works exhibit:

- The high-level policy is not goal-conditioned and needed to be retrained for new tasks, incurring unnecessary memory and computation (as shown in subsection 5.5.3).
- Insufficient exploration and sample efficiency in learning long-horizon manipulation tasks (as shown in subsection 5.5.2).
- Do not consider parallel training of both levels (see below).

The second point that this chapter studies is the parallel training process of the both planning and motion control levels. A very common practice in HRL is the use of pre-trained low-level policies. The limitations of using pre-trained low-level policies are twofold. First, separately training both levels induces unnecessary repeat of data collection, as they normally appear within the same task environment. Secondly, separate training is likely to induce incompatibility between the two levels, which demands the low-level policy to be fine-tuned while training the high-level. On the other hand, parallel training trains both levels within the same data collection loop with much less computation and naturally pushes the low-level policy to adapt to the need for the high-level policy. These two phenomena are demonstrated in subsection 5.5.2.

Nonetheless, a good reason to use separate training is that parallel training may result in low sample efficiency and divergence because it puts the high-level policy into non-stationary MDP dynamics. The non-stationarity is caused by a suboptimal and constantly exploring low-level policy. In other words, the training of the high-level planning policy is most stable with an optimal low-level policy that does not explore randomly. Since this is impossible in parallel training, remedies have been proposed (Nachum *et al.*, 2018; Levy *et al.*, 2019). However, this chapter will show that these remedies are insufficient to cope with long-horizon and multi-step manipulation tasks through a theoretic analysis in subsection 5.4.2 and a comparative study in subsection 5.5.2. According to the analysis, the method $\mathbf{A}^2$ developed in chapter 4 is used to eliminate the non-stationarity issue, achieving substantial improvement over previous methods.

5.1.1 Summary and chapter organisation

To sum up, this chapter focuses on solving long-horizon, multi-step and multi-outcome manipulation tasks with HRL. The two specific research questions studied in this chapter regarding HRL are:

- *How to improve memory usage and reusability?*

- *How to eliminate parallel training non-stationarity for long-horizon tasks?*

The rest of the chapter is organised as follows. Section 5.2 introduces the basics of HRL frameworks, especially focused on the classic option framework (OF)(Sutton *et al.*, 1999b). For the first question, section 5.3 explains how to replace the standard policies in the classic OF with goal-conditioned policies, giving birth to the UOF. Implementation for a series of multi-step, multi-outcome block stacking tasks are also illustrated. For the second question, section 5.4 introduces the specific training algorithms, then takes a closer look into the root of the non-stationarity and explains how to use the $\mathbf{A}^2$ method developed in chapter 4 to eliminate the problem for long-horizon multi-step tasks. In section 5.5, the experiment design and the results on a series of block stacking tasks are introduced. Finally, section 5.6 concludes this chapter.

5.2 Hierarchical Reinforcement Learning

As the name implies, HRL extends standard RL formulation with a hierarchy of policies. In some HRL papers, standard RL policies are referred to as "flat" policies, because they do not have a hierarchy. The central idea of HRL is on the learning and design of temporal abstraction in behaviour learning (Sutton *et al.*, 1999b). In plain words, an agent with temporal abstraction can reason across multiple timesteps in an MDP, whereas a flat policy is only concerned about what actions to take at every timestep. Therefore, HRL is advantageous over standard RL when it comes to long-horizon tasks that require planning with diverse skills (Pateria *et al.*, 2021b).

There are usually two levels in the hierarchy, referred to sometimes as the high and low levels, sometimes as the manager and worker, sometimes the planning and control levels, and so on.[1] In the OF, they are referred to as the inter-option policy and the option. The following content will use these terms interchangeably in consideration of clarity. The general idea is that the high-level policy is to select motion primitives, subtasks, subgoals, sub-policies or options, while the low-level policy is to actually interact with the environment according to what is selected by the high-level policy.

There is a large body of papers in the field of HRL studying a diverse set of sub-problems, hence the diverse terms used in the literature (Pateria *et al.*, 2021b). Among them, the OF is of particular interest to this chapter because it provides a sound mathematical foundation extended from the standard RL framework (Sutton *et al.*, 1999b) described in section 3.1. This is attractive because the extension of mathematical formulation naturally brings about the extension of their solutions. In addition, there are many HRL algorithms and applications that are not mathematically based on the OF, but can be interpreted as a variant of the OF. Nonetheless, this section will start by introducing the OF and basic principles of learning algorithms. Then, a short survey and discussion on the construction of the hierarchy are provided to help draw a connection with various research works.

5.2.1 The option framework

As mentioned, the core concept of HRL is a temporal abstraction, which enables reasoning across multiple timesteps. The OF realises temporal abstraction by introducing options into the MDP. Options are called *macro-actions* and the single-step actions are called *micro-actions* in some literature. For-

[1]Exceptions that deal with more than two levels exist (Levy *et al.*, 2019), but they can be generalised from two-level methods and are less relevant to the focus on this thesis.

mally, an option, $o \in \mathcal{O}$, is a temporally extended action that takes a number of single-step actions in an MDP. It is defined as a tuple $(\pi^o, \mathcal{I}^o, \beta^o)$, where $\pi^o : \mathcal{S} \times \mathcal{A} \to [0, 1]$ is a standard RL policy distribution, $\mathcal{I}^o \in \mathcal{S}$ is a set of state where the option policy π^o can be initiated, and $\beta^o : \mathcal{S} \to [0, 1]$ is the termination distribution that determines whether the option should terminate given a state. In the deterministic cases, it becomes a binary termination function denoted as $b^o : \mathcal{S} \to \{0, 1\}$.

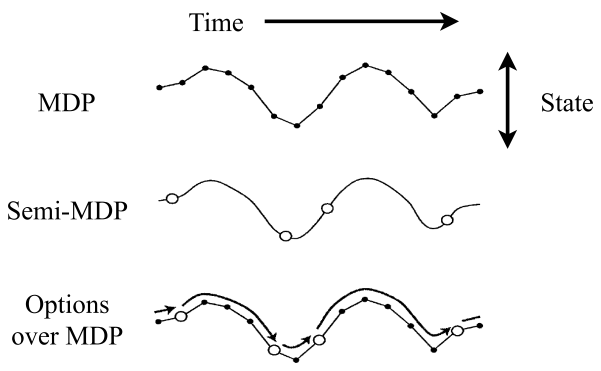


Figure 5.2: Figure 1 from (Sutton *et al.*, 1999b): The state trajectory of an MDP is made up of small, discrete-time transitions, whereas that of an SMDP comprises larger, continuous-time transitions. Options enable an MDP trajectory to be analyzed in either way.

In standard MDPs, an action is taken at each discrete timestep, while an option takes several actions until it is terminated according to β^o. An agent is expected to select a new option and follow its option policy when the current option terminates. One may view the framework as a team where the options are a number of team members that specialised in different subtasks, the actions are what actually these members are doing to finish the subtasks, and the team leader is the option-selection (high-level) policy that chooses who in the team to start working. Unlike reality where multiple teammates can work simultaneously, in the OF typically there is only one option (low-level policy) that is activated at the same time.

The concept of semi-MDP (SMDP) may help understand options as well. An MDP is known to become a semi-MDP with a fixed set of options (see Figure. 5.2) (Sutton *et al.*, 1999b). This is because the Markov property is only retained at the level of option transition. In other words, the state-action-state transition is non-Markovian because it is dependent also on the current option and the state-action history that happened since the current option was initiated. The option-state-option transition, on the other hand, is Markovian if the option policies are themselves Markovian. This insight is important because it means the solutions for MDPs can be applied to the OF when given a set of Markovian options. In practice, Markov property is a common assumption for options (Barreto *et al.*, 2019; Jiang *et al.*, 2019), and is also adopted in this chapter.

A classic example of a block stacking task may help to understand options, actions and their relationship. Assume that there are three options named *Pick*, *MoveTo* and *Release*. At the beginning of an interaction episode, the inter-option (high-level) policy may select the *MoveTo* option policy, $\pi^o_{MoveTo}(a|s)$, which produces 5 actions to reach a position where a block can be grasped. Then, the *Pick* option, $\pi^o_{Pick}(a|s)$ may be chosen, which spends 3 actions to close the fingers and firmly grasp the block. Then the *MoveTo* option can be used again to move the block to a target location in another 4 timesteps. Finally, the *Release* option can be activated and produces 3 actions to let go of the block. Notice that in this example, the three options can be activated in any state, i.e., $\mathcal{I}^o = \mathcal{S}$; the termination function stops an option when a state matches the target state of the corresponding subtask; only one option can be activated at one time; the numbers of actions produced by each option are different; and a new option is only activated when the last option is terminated.

Similar to standard RL, the objective for an OF agent is to find a set of options that select actions as well as an inter-option policy that selects options to maximise return. Therefore, there are two problems to be tackled: constructing the options and learning to plan over them. The following will first discuss common practices for constructing the options, and then introduce methods that learn to select options.

5.2.2 How options have been constructed

By definition, an option has three components: the initiation set, the policy and the termination condition. The easiest choice to construct them would be through manual programming, in which case all the components of an option are user-defined. What is left to be done is to learn the inter-option policy. However, manual programmes are mostly inflexible. A slightly more autonomous choice is to assign each option to a specific subtask and use off-the-shell motion planners to solve them. For example, the recent idea of HRL with parameterised primitives (Dalal *et al.*, 2021). In this case, the HRL problem becomes very similar to many hierarchical control problems in the robotic community, such as the popular TAMP methods as discussed in subsection 2.4.1.

On the other extreme, options can be learnt end-to-end along with the inter-option policy. Learning these components end-to-end is non-trivial and remains largely underdeveloped. The central question is how to define the learning objective such that meaningful, distinct and interpretable options (or skills, subgoals, subtasks) emerge naturally out of the optimisation process. There are various terms referring to this particular problem: option discovery, subpolicy discovery, subgoal discovery, skill discovery, etc., depending on the perspective of the works. Recent advances in end-to-end methods include the option-critic (Bacon *et al.*, 2017), deep continuous option discovery (Krishnan *et al.*, 2017), inferred option policy gradient (Smith *et al.*, 2018), and double actor-critic (Zhang and Whiteson, 2019). Studies made specifically for the termination condition (Harutyunyan *et al.*, 2019) and the initiation set (Khetarpal *et al.*, 2020b) are also conducted. Nonetheless, all these works showed that learning the options and the planner altogether from a single reward function is a very difficult problem, the algorithms are brittle and the resultant options are difficult to interpret (Pateria *et al.*, 2021b).

Many researchers then sought to learn only the option policies, leaving the initiation set and termination condition user-defined. The most common assumption for the initiation set would be that an option is available everywhere: $\mathcal{I}^o = \mathcal{S}$. The agent can select any option in any state. Defined assumptions of the termination condition differ. A few common ones include: 1) when the option has been executed after a fixed number of timesteps; 2) when a subgoal or subtask designated for the option is achieved; 3) task-specific terminations such as a robot reaches an invalid position or pose. These conditions are commonly used together.

Given the defined initiation set and termination conditions, there are a diverse set of methods for constructing the option policies. Commonly, each option is constructed with a specific subtask, skill or subgoal in mind. In other words, these methods require human priors to define the purpose or function of each option before it is actually learnt. This assumption releases the algorithms from the difficulty of automatic task decomposition of the full end-to-end HRL problem. There are several directions regarding how the learning is conducted.

First and the most common choice is to train each option policy with respect to a specific reward function, a task or a skill, such as in (Yang *et al.*, 2020). However, most methods use pre-trained option policies without further fine-tuning because they attempt to avoid the non-stationarity that appears when training both levels simultaneously (Sutton *et al.*, 1999b). The problem occurs because, with suboptimal, exploratory and updating option policies, the MDP of the inter-option policy is ever-changing. The second contribution of this thesis utilises the method proposed in the first contribution to deal with such non-stationarity. There are also methods that attempt to fine-tune the pre-trained option policies while training the inter-option policy, which is crucial when the sub-policies are to be reused in different high-level tasks (Li *et al.*, 2019). Secondly, instead of training each option towards its own reward function, many researchers started to use goal-conditioned options. The reason behind this is that it is easier to represent the action space of the inter-option policy as a subgoal space instead of a space of reward functions. Therefore, the inter-option policy effectively becomes a subgoal generator rather than an option selector (Peng *et al.*, 2017; Jiang *et al.*, 2019; Levy *et al.*, 2019; Staroverov *et al.*, 2020). The difficulty of option discovery is thus transformed to the difficulty of subgoal discovery (Dilokthanakul *et al.*, 2019; Pateria *et al.*, 2021a). The second contribution of this thesis uses such goal-conditioned policies for not only options, but also the inter-option policy to solve multi-outcome, multi-step and long-horizon manipulation tasks. Lastly, researchers also seek to extract skills from demonstrations (Hakhamaneshi *et al.*, 2022) or imitate human motion priors (Peng *et al.*, 2019) before reusing them on high-level tasks that require a composition of these skills.

In summary, there is a diversity of learning methods that one can use to construct options given a known set of target subtasks. They provide different benefits and suit different needs. Perhaps more interesting and difficult is still the study of how subtasks or skills could emerge automatically without human priors, such as the fully end-to-end HRL methods. However, deeper details are beyond the scope of this thesis.

5.2.3 Inter-option policy learning

The simplest way to consider learning to plan over a set of known options is to treat the options as actions in the standard MDP framework. Essentially, this neglects the influence of the number of micro-actions differed from option to option and reduces options into single-step actions. Therefore, the standard RL algorithms for discrete action space, such as Q learning, can be applied straightforwardly by simply replacing the actions with options (Eq. 3.7). This is the most common way to perform HRL (Pateria *et al.*, 2021b). Typically, the reward function is computed based only on the system states, irrelevant to the option that is taken (Pateria *et al.*, 2021b). This assumption is adopted by this thesis as well, although it could be problematic in certain cases. For instance, in parallel training, an option may accidentally achieve a rewarding state due to random exploration, causing the algorithm to assign that option greater expected values when it should not do so. However, such cases are statistically rare and tend to be resolved by reducing exploration over time and increasing training time. More details can be found in credit assignment research (Sutton, 1984; Zhou *et al.*, 2020).

The problem with this way of learning is that every execution of an option will only result in a single transition $\xi_n = \{s_n, o_n, r_n, s'_n, o'_n\}$, while the transitions between the two options are discarded. The learning algorithm can only learn about the value of one option each time when a selected option is terminated. Let's call these methods *wait-and-see*. However, if the option is Markovian, which means that the option policy selects actions depending only on the current state, then each transition produced during the execution of the option policy is valid to learn about the option and other options. There is an *off-policy* manner to learn about the value of different options more efficiently. The following will introduce such an algorithm, Intra-option learning (IOL), that learns an inter-option policy with data in between two consecutive options. IOL can also learn about other options when they are not even selected (Sutton *et al.*, 1998).

With the *wait-and-see* type of update, the value of an option is evaluated only at the point of termination. The reward for selecting that option can either be the cumulated option return or a high-level reward function that only depends on the resultant state. For example, it can be learnt with the following Q learning minimisation objective:

$$\begin{aligned} J(\boldsymbol{w}) &\approx \mathbb{E}_{\xi_n \sim \mathcal{D}}\left[\frac{1}{2}(\hat{q} - \tilde{q}_{\boldsymbol{w}}(s_n, o_n))^2\right] \\ &\approx \mathbb{E}_{\xi_n \sim \mathcal{D}}\left[\frac{1}{2}((r_n + \gamma \max_{o'} \tilde{q}_{\boldsymbol{w}^-}(s'_n, o')) - \tilde{q}_{\boldsymbol{w}}(s_n, o_n))^2\right] \end{aligned} \tag{5.1}$$

where s_n is the terminal state for option o_n. In order to learn about options using data generated during the execution of an option, the estimation of the true (target) state-option value $\hat{q}$ should consider the possibility that other options could be selected in the next timestep. In other words, given a collected transition generated by an option, it is no longer certain to the learning agent whether it will terminate or not. There are thus two cases for the q value of the next state: it equals the maximum option value for the next state if the current one terminates, or the option value of the current option as it continues. Therefore, the state-option value for the next state can be redefined and estimated as follows:

$$\tilde{u}_{\boldsymbol{w}^-}(s'_n, o') = (1 - \beta^{o_n}(s'_n))\tilde{q}_{\boldsymbol{w}^-}(s'_n, o_n) + \beta^{o}(s) \max_{o'} \tilde{q}_{\boldsymbol{w}^-}(s'_n, o') \tag{5.2}$$

Eq. 5.2 is called the *option value upon arrival*, which is not restricted to the terminal transition of an option. Thus, we can write the cost function to be maximised for the policy parameters given a transition $\xi_n = \{s_n, o_n, \beta^{o_n}(s'_n), s'_n\}$ as:

$$\begin{aligned} J(\boldsymbol{w}) &\approx \mathbb{E}_{\xi_n \sim \mathcal{D}}\left[\frac{1}{2}(\hat{q} - \tilde{q}_{\boldsymbol{w}}(s_n, o_n))^2\right] \\ &\approx \mathbb{E}_{\xi_n \sim \mathcal{D}}\left[\frac{1}{2}((r_n + \gamma\tilde{u}_{\boldsymbol{w}^-}(s'_n, o')) - \tilde{q}_{\boldsymbol{w}}(s_n, o_n))^2\right] \end{aligned} \tag{5.3}$$

where, $\tilde{u}_{\boldsymbol{w}^-}(s'_n, o') = [(1 - \beta^{o_n}(s'_n))\tilde{q}_{\boldsymbol{w}^-}(s'_n, o_n) + \beta^{o_n}(s'_n) \max_{o'} \tilde{q}_{\boldsymbol{w}^-}(s'_n, o')]$. In comparison with Eq. 5.1, Eq. 5.3 is now able to use every transition that is produced when executing a particular option, without waiting for an option to be terminated. Also, it has the benefits of learning from more diverse samples as well as learning about different options with each sample. The algorithm is therefore named Intra-option learning (Sutton *et al.*, 1998). Moreover, this algorithm is to be extended in the subsection 5.4.1 to incorporate goal-conditioned options and inter-option policies. More advanced topics and possible variations regarding learning inter-option policies can be found in (Sutton *et al.*, 1998, 1999b).

5.2.4 Summary

In sum, HRL seeks to solve long-horizon tasks in a typically two-level structure. The two levels deal with planning at a coarse time scale and low-level control at a finer time scale. The classic option framework (OF) is introduced in this section to form the basis for understanding the state-of-the-art HRL methods as well as the proposed UOF in the next section. The IOL algorithm will be extended to be compatible with a goal-conditioned option and a goal-conditioned inter-option policy.

5.3 The Universal Option Framework

With the foundations introduced in the last section, this section will formally describe the proposed UOF, which integrates goal-conditioned policies (Schaul *et al.*, 2015; Andrychowicz *et al.*, 2017) into the classic option framework (Sutton *et al.*, 1999b). The reader is referred to section 4.2.1 for the preliminary of GRL.

The first subsection below will describe formally the universal option and high-level policy. The second subsection will explain the specific implementation of the block stacking tasks for a better understanding of the concepts and experiments.

5.3.1 Universal option and high-level policy

The original option framework, as introduced in subsection 4.2.1, consists of an inter-option policy at the higher level and a set of options at the lower level. An option is comprised of an initiation set, a termination function and an intra-option policy (Sutton *et al.*, 1999b). For clarity, the term "high-level policy" will be used to refer to the inter-option policy that selects options at the planning level, while "low-level policy" will be used to refer to the policies that interact directly with the environment at the lower level. The following will formally describe the goal-conditioned option and high-level policy, and the link between them. The term "universal" is inherited from (Schaul *et al.*, 2015; Andrychowicz *et al.*, 2017) to emphasise that the proposed framework is able to represent and learn knowledge of the tasks not just for states but also for different goals.

Universal option

To briefly recall, an option, $o \in \mathcal{O}$, consists of three components: a low-level policy $\pi^o : \mathcal{S} \times \mathcal{A}^L \to [0, 1]$, the initiation set $\mathcal{I}^o \in \mathcal{S}$ and the termination distribution $\beta^o : \mathcal{S} \to [0, 1]$. The option in the original framework is typically defined over states. This means that the low-level policy is single-objective, the initiation set indicates whether an option can be selected given a state, and the termination function indicates whether the option should terminate given a state.

Similarly, a universal option, o_g, also consists of three components: a universal policy $\pi_g^L : \mathcal{S} \times \mathcal{G}^L \times \mathcal{A}^L \to [0, 1]$, a universal initiation set $\mathcal{I}_g^o \in \mathcal{S}$ and a universal termination distribution $\beta_g : \mathcal{S} \times \mathcal{G}^L \to [0, 1]$. However, the universal option is now defined over states and goals. This means that the universal policy is now capable of achieving multiple objectives in terms of goals. In addition, it is natural to define that the initiation set should indicate a subset of states from which a desired goal is achievable, and the termination function should indicate whether a desired goal is achieved given a state. In this thesis, the termination condition is assumed to be deterministic such that the universal option terminates when the given desired goal is achieved, denoted as $b_g : \mathcal{S} \times \mathcal{G}^L \to \{0, 1\}$.

Universal high-level policy

In the standard option framework, a high-level policy is defined as a mapping from states to options, denoted as $\pi^H : \mathcal{S} \times \mathcal{A}^H \to [0, 1]$[2] Implementation-wise, the high-level action space is discrete, commonly represented as a set

[2]For the ease of illustration, the following will use $\mathcal{A}^H$ to replace $\mathcal{O}$.

of integers. The high-level policy may be learnt by semi-MDP q-learning or intra-option learning (see subsection 5.2.3) to select options to maximise the reward function.

When there is only one universal option, it is not difficult to notice that the high-level policy can no longer be a distribution over options (or high-level actions). A common practice is to define it as a distribution over low-level goals: $\pi_g^H(g|s) : \mathcal{S} \times \mathcal{G}^L \rightarrow [0, 1]$, (Nachum *et al.*, 2018; Levy *et al.*, 2019; Jiang *et al.*, 2019; Dilokthanakul *et al.*, 2019). In deterministic cases, it is then a mapping $\pi_g^H(s) : \mathcal{S} \rightarrow \mathcal{G}^L$. In such cases, the high-level policy is responsible for selecting a sequence of subgoals to be achieved by the universal option such that the future return is maximised. In other words, $\mathcal{A}^H = \mathcal{G}^L$. Although this definition is very natural and involves the least human effort to design the high-level action space, it is very brittle and inefficient to learn when the goal space is continuous and high-dimensional. In such difficult cases, learning the high-level policy effectively equals the subgoal generation problem, which itself has drawn interest from researchers in recent years (Dilokthanakul *et al.*, 2019; Pateria *et al.*, 2021a).

However, this chapter proposes to retain a discrete high-level action space, in which an action is an integer that corresponds to a subtask or skill. In the goal-conditioned cases, ac action corresponds to a set of desired subgoals. This alternative asks for a task decomposition scheme, which is precisely what has been described in the $\mathbf{A}^2$ methods. The assumptions made in subsection 4.2.3 are employed again with a slight change to emphasise that the goal space is at the low level:

- A task decomposition scheme is available to divide the task into a total of I subtasks.
- For each subtask, indexed by $i \in \mathbb{N}, i \leq I$, there exists a non-empty

subset of the low-level goal space covering all the desired low-level goals for that subtask: $\mathcal{G}_i^{L+} \neq \emptyset$ and $\mathcal{G}_i^{L+} \in \mathcal{G}^L$.

- The high-level action space is comprised of the indexes of the subtasks: $a^H \in \mathcal{A}^H = \{i \mid i \in \mathbb{N}, i \leq I\}$.

Finally, the high-level policy itself can be made goal-conditioned, such that it may learn to achieve different task outcomes using the same set of subtasks or skills. Formally, the universal high-level policy is a mapping $\pi_g^H : \mathcal{S} \times \mathcal{G}^H \times \mathcal{A}^H \rightarrow [0, 1]$. Per practice for GRL, the reward function for learning the high-level policy is also based on goals, typically associated with a sparse indicator function of whether a goal is achieved.

Summary and discussion

In sum, the UOF operates as the following describes. At the beginning of an episode, a desired final task goal is sampled and given to the high-level policy. With the current observation and the final goal, the high-level policy selects a high-level action, $a^H \sim \pi_g^H(a^H|s, g^{H+})$, which corresponds to a subtask and a desired subgoal, $g_i^{L+} \sim \mathcal{G}_i^{L+}$, where $i = a^H$. The universal option then takes low-level actions, $a^L \sim \pi_g(a^L|s, g_i^{L+})$, to interact with the environment to reach a state where the desired subgoal is achieved. When the subgoal is achieved, the high-level policy is queried again for the next subgoal based on the new observation. The episode resets when a maximum number of environment interaction steps is reached (notice it is NOT the number of high-level actions).

Decomposing the continuous low-level goal space into a number of subspaces associated with subtasks or skills is in fact a very important architectural choice for the proposed hierarchical learning framework. It first reduces the dimensionality of the high-level action space, then more importantly enables the application of abstract demonstrations and adaptive exploration (developed in chapter 4). The latter is vital in eliminating the non-stationary training issues (subsection 5.4.2) and improving sample efficiency of parallel training for long-horizon multi-step multi-outcome tasks (subsection 5.5.2).

Implementation-wise, the experiments in this chapter will rely on a manually designed scheme for task decomposition. However, similar to what has been discussed in section 4.4, the learning of task decomposition is an interesting direction, closely related to the active field of subgoal discovery (Dilokthanakul *et al.*, 2019; Pateria *et al.*, 2021a). In addition, this chapter advocates for the integration with GRL, representing subtasks in the goal space. As demonstrated by the empirical results in section 5.5, this has the potential of improving memory usage and policy reusability.

Finally, before the discussion of how to improve the training process of UOF, the next subsection will describe the detailed implementation of the proposed framework for the manipulation tasks of interests, so that the reader can build up a more grounded understanding.

5.3.2 Implementation

This subsection illustrates the implementation details of the MDP and the UOF for a set of block-stacking tasks simulated by the Mujoco engine. An example of the simulation environment is shown by Figure 5.1, where a robot is tasked to stack the three blocks as a tower in two different orders (green→blue→red, or blue→green→red). In this example, the UOF agent needs to learn both stacking outcomes. Both levels of the UOF will share the same state representation and initial state distribution, but they will have different action spaces, goal representations and reward functions. A total of eight tasks are to be used to evaluate the framework and training methods introduced in the next section (see subsection 5.5.1 for task variations).

States & initial state distribution

For a block-stacking task with M blocks, its state $\mathbf{s} = (\mathbf{s}_{gr}||\mathbf{s}_1^b||\mathbf{s}_2^b||...||\mathbf{s}_M^b)$ is comprised of the states of the gripper $\mathbf{s}_{gr}$ and the blocks $(\mathbf{s}_1^b||\mathbf{s}_2^b||...||\mathbf{s}_M^b)$, where $||$ denotes vector concatenation. The gripper state consists of the absolute Cartesian coordinates and the linear velocity of the gripper, the linear velocity of the gripper fingers (symmetric), and the finger width: $\mathbf{s}_{gr} = (\mathbf{x}_{grip}^{abs}||\mathbf{v}_{grip}^{abs}||v_{finger}^{abs}||w_{finger})$. The state of the m-th block consists of its relative Cartesian coordinates, linear and angular velocities with respect to the gripper tip frame: $\mathbf{s}_m^b = (\mathbf{x}_m^r||\mathbf{v}_m^r||\mathbf{w}_m^r), \forall m \in \{1, ..., M\}$.

The initial distribution of the state $p_0(\mathbf{s})$ is described as follows. At the beginning of an episode, the robot gripper is positioned above the centre of the table. The blocks are randomly placed on the table with their initial orientations aligned with the world frame. For the m-th block, its initial x-y position, (x_0^m, y_0^m), is uniformly sampled on the planar workspace within a square, centred at the gripper's x-y position (x_0^{gr}, y_0^{gr}), i.e., $x_0^m \sim \mathcal{U}(x_0^{gr} - \delta, x_0^{gr} + \delta)$ and $y_0^m \sim \mathcal{U}(y_0^{gr} - \delta, y_0^{gr} + \delta)$, where δ is half of the square edge length. In this work, $\delta = 15\ cm$.

Actions

In this chapter, we are dealing with deterministic policies. For all tasks, the low-level policy, $\pi_g^L : \mathcal{S} \times \mathcal{G}^L \rightarrow \mathcal{A}^L$, is responsible for controlling the gripper in the 3D Cartesian space and controls its finger width. A low-level action, $\mathbf{a}^L \in \mathcal{A}^L$, therefore, has four elements, including the displacement of the gripper tip frame (range in $[-0.05 \text{ m}, 0.05 \text{ m}]$) and the target finger width (range in $[0, 0.1 \text{ m}]$), denoted as $\mathbf{a}^L = (\Delta x^{gr}||\Delta y^{gr}||\Delta z^{gr}||w_{finger})$. An exception is the Rotation Task (see subsection. 5.5.1), where the agent is additionally allowed to rotate the gripper around its Z-axis ($\Delta yaw \in [-\frac{\pi}{2}, \frac{\pi}{2}]$). All action dimensions are normalised into $[-1, 1]$, as per RL common practice aligned with the continuous RL control literature (Lillicrap *et al.*, 2015).

For the high-level action space, there are I discrete actions related to the I subtasks,i.e., $a^H \in \mathcal{A}^H = \{i \mid i \in \mathbb{N}, i \leq I\}$. The value of I differs for different tasks (see subsection. 5.5.1). For instance, there will be 6 high-level actions for the second task in Table 5.1: $a^H \in \mathcal{A}^H = \{0, 1, 2, 3, 4, 5\}$.

Low-level goals & reward function

Following the mathematical convention in section 4.2, denote a low-level goal from a representation mapping of the state as, $\mathbf{g}^L = m^L(\mathbf{s})$. In the following experiments, a low-level goal is represented as a vector consisting of the absolute Cartesian coordinates of all blocks and the gripper tip frame, and the gripper finger width. Formally, given M blocks, a low-level goal is represented as a vector:

$$\mathbf{g}^L = (\mathbf{x}^{abs}_{grip} || w_{finger} || \mathbf{x}^{abs}_1 || \mathbf{x}^{abs}_2 || ... || \mathbf{x}^{abs}_M)$$

It is not difficult to verify that the core assumption of GRL is satisfied in all tasks: given a state $\mathbf{s}$, there is always a goal that can be found (is achieved) at that state. Formally: $\forall \mathbf{s} \in \mathcal{S}, \exists \mathbf{g}^L \in \mathcal{G}^L, \text{s.t. } \mathbf{g}^L = m^L(\mathbf{s})$.

As mentioned, this chapter assumes access to a task decomposition scheme that generates a number of subtasks for a given final task. Therefore, the definition of the desired low-level goals, $\mathbf{g}^{\mathcal{L}+}$, differs from subtask to subtask. They are calculated according to the following principles:

- For a grasping subgoal of the m-the block, $\mathbf{g}^{H+}_{grasp\_m}$, the block positions remain the same as the current system state, while the gripper position is set to the position of the target block and the finger width is equal to the size of the block. For instance, in a task with M blocks, denote l_1 as the size of the first block, then the subgoal of grasping the first block is given as $\mathbf{g}^{H+}_{grasp\_1} = (\mathbf{x}^{abs}_1||l_1||\mathbf{x}^{abs}_1||\mathbf{x}^{abs}_2||...||\mathbf{x}^{abs}_M)$.

- For a placing subtask of the m-th block, $\mathbf{g}^{H+}_{place\_m}$, the block positions remain the same as the current system state, except for the block to be placed, whose position equals the target location. The gripper position is also set to the target location and the finger width is equal to the size of the block. For a task with M blocks, with the target position for the first block denoted as $\mathbf{x}^{abs+}_1$, the subgoal for placing the first block is written as $\mathbf{g}^{H+}_{place\_1} = (\mathbf{x}^{abs^+}_1||l_1||\mathbf{x}^{abs^+}_1||\mathbf{x}^{abs}_2||...||\mathbf{x}^{abs}_M)$.
- For both grasping and placing subtasks, other blocks can be set to a certain location as well, such that the desired low-level goal can represent situations where some other blocks are already correctly stacked. For example, following the same context, the subgoal of placing the second block while the first block is at position $\mathbf{x}^{abs^+}_1$ can be written as $\mathbf{g}^{H+}_{place\_2} = (\mathbf{x}^{abs^+}_2||l_2||\mathbf{x}^{abs^+}_1||\mathbf{x}^{abs+}_2||...||\mathbf{x}^{abs}_M)$.
- For all subtasks, the desired low-level goals are updated at every timestep, because some components are equal to the value of the current state. For example, the positions of blocks that are irrelevant to the concerned subtask.
- In all tasks, the final goal requires all the blocks to be put in a certain configuration and the gripper to stay back to its initial position x^{abs}_{start} with fingers closed $w_{finger} = 0$. For example, the last subgoal of stacking two blocks can be written as $\mathbf{g}^{H+}_{end} = (\mathbf{x}^{abs}_{start}||l_2||\mathbf{x}^{abs^+}_1||\mathbf{x}^{abs+}_2)$.

Following these principles, one can write down the subgoals for the block stacking tasks that will be experimented with in this chapter (Table 5.1 and 5.2). The following will give a small example for the task of stacking two out of three blocks in two different orders (two final outcomes) as an example (task 2 in Table 5.1). Denote the positions of the three blocks by their colours red, green and blue, the subgoals can be written down in the correct order as follows.

$$\begin{aligned}
\mathbf{g}^{H+}_{grasp\_B} &= (\mathbf{x}^{abs}_B || l_B || \mathbf{x}^{abs}_R || \mathbf{x}^{abs}_B || \mathbf{x}^{abs}_G) \\
\mathbf{g}^{H+}_{place\_BR} &= (\mathbf{x}^{abs^+}_B || l_B || \mathbf{x}^{abs}_R || \mathbf{x}^{abs^+}_B || \mathbf{x}^{abs}_G) \\
\mathbf{g}^{H+}_{end\_BR} &= (\mathbf{x}^{abs}_{start} || 0 || \mathbf{x}^{abs}_R || \mathbf{x}^{abs^+}_B || \mathbf{x}^{abs}_G) \\
\mathbf{g}^{H+}_{grasp\_G} &= (\mathbf{x}^{abs}_G || l_G || \mathbf{x}^{abs}_R || \mathbf{x}^{abs}_B || \mathbf{x}^{abs}_G) \\
\mathbf{g}^{H+}_{place\_GR} &= (\mathbf{x}^{abs^+}_G || l_G || \mathbf{x}^{abs}_R || \mathbf{x}^{abs}_B || \mathbf{x}^{abs^+}_G) \\
\mathbf{g}^{H+}_{end\_GR} &= (\mathbf{x}^{abs}_{start} || 0 || \mathbf{x}^{abs}_R || \mathbf{x}^{abs}_B || \mathbf{x}^{abs^+}_G)
\end{aligned}$$

The reward function for the low-level policy is a standard goal-conditioned reward function described in subsection 4.2.1, which gives a reward of 0 when a desired goal is achieved and -1 otherwise (Andrychowicz *et al.*, 2017). Whether a desired goal is achieved is here measured by the L2 distance. Given the current state $\mathbf{s}$, a desired low-level goal $\mathbf{g}^{L+}$ is said to be achieved if the actual achieved low-level goal $\mathbf{g}^L = m^L(\mathbf{s})$ is closed enough to the desired one in terms of the L2 distance with a threshold. Denote $\mathbb{1}\left[c(\cdot)\right]$ as the indicator function that gives 1 when the condition $c(\cdot)$ is satisfied and 0 otherwise, the reward function for the low-level policy is

$$r^L(\mathbf{s}_t, \mathbf{a}^L_t, \mathbf{g}^{L+}) = \mathbb{1}\left[||m^L(\mathbf{s}_{t+1}) - \mathbf{g}^{L+}||_2 \le \delta_d\right] - 1$$

where, $\mathbf{s}_{t+1}$ is the next state according to the system dynamic and the threshold $\delta_d = 0.02$ m in all experiments.

High-level goals & reward function

The high-level goal space employs a N-dimensional binary representation, where N is the number of subtasks/steps for a task. The value of each dimension is determined according to whether the associated subtask is achieved or not. That is: given a state $\mathbf{s}$,

$$\mathbf{g}^H = (f_{\mathbf{g}_1^{L+}}, f_{\mathbf{g}_2^{L+}}, ..., f_{\mathbf{g}_N^{L+}}),$$

where $f_{\mathbf{g}_n^{L+}} = \mathbb{1}\left[||m^L(\mathbf{s}) - \mathbf{g}_n^{L+}||_2 \leq \delta_d\right]$ is the binary indicator function that indicates whether the n-th subtask is achieved at the given state.

Similar to the low-level case, the desired high-level goal, $\mathbf{g}^{H+}$, differs from subtask to subtask. Again, taking the second task in Table 5.1 as an example, the high-level goal will be a 6-D vector as there are 6 subgoals. In total, there will be 6 different outcomes that the high-level policy may be asked to achieve. They can be written as follows.

$$\begin{aligned}
\mathbf{g}^{H+}_{grasp\_B} &= (100000) \\
\mathbf{g}^{H+}_{palce\_BR} &= (110000) \text{ placing requires the grasping subgoal remain achieved} \\
\mathbf{g}^{H+}_{end\_BR} &= (001000) \\
\mathbf{g}^{H+}_{grasp\_G} &= (000100) \\
\mathbf{g}^{H+}_{place\_GR} &= (000110) \\
\mathbf{g}^{H+}_{end\_GR} &= (000001)
\end{aligned}$$

Similar to the low-level case, the high-level reward function is calculated by checking whether the desired high-level goal is achieved. Whereas, this time it uses element-wise equality as the measurement:

$$r^H(\mathbf{s}_t, \mathbf{a}_t^L, \mathbf{g}^{H+}) = \mathbb{1}\left[m^H(\mathbf{s}_{t+1}) = \mathbf{g}^{H+}\right] - 1$$

where $\mathbf{s}_{t+1}$ is the state after a low-level action $\mathbf{a}^L$ is executed at the last state s. This reward function gives a reward of 0 when the desired high-level goal is achieved and -1 otherwise. Here, notice that the reward function is irrelevant to which high-level action is selected by the high-level policy. This assumption can be problematic in some cases as discussed in subsection 5.2.3, but in this chapter, it does not affect the result. For more research in this regard, please refer to (Sutton, 1984; Zhou *et al.*, 2020).

Summary

In sum, this subsection illustrates how the block stacking tasks are implemented in detail, including the goal-augmented Markov decision process definitions for both levels in the UOF. An example of an episode of the second task may help to see the overall idea. At the beginning of an episode, the high-level policy is given a high-level goal to stack the green block on the red one $g^{H+}_{place\_GR}$. According to $\pi^H_g(\mathbf{s}_0, g^{H+}_{place\_GR})$, the high-level policy may select $a^H_0 = 3$ which corresponds to the fourth low-level goal $g^{L+}_{grasp\_G}$. Then, the low-level policy may achieve the goal at $t = 4$, receiving 3 rewards of -1 and one of 0. The high-level policy is then given a reward of -1 as it has not yet achieved the high-level goal. The high-level policy may now select the second action $a^H_4 = 4$ which gives the fifth low-level goal $g^{L+}_{place\_GR}$ to the low-level policy. Assume that, at $t = 7$, the low-level policy achieves the goal according to $\pi^L_g(\mathbf{s}_7, g^{L+}_{place\_GR})$, both policies will be given a reward of 0. The objective for both levels is the same as any standard RL algorithm: maximising the expected future discounted return. In the next section, the detailed training method for the UOF will be introduced.

5.4 Training Methods

This section will introduce how both levels of the UOF are updated. The training process is organised into *epoch, cycle* and *episode*, exactly the same as the experiments in chapter 4 following (Andrychowicz *et al.*, 2017). Again, each epoch has 50 cycles, each of which has 16 episodes. The different task allows a different number of interaction steps. Note that, in the HRL setting, the number of interaction steps that occurred corresponds to the number of low-level actions taken, which varies for different tasks (see subsection 5.5.1).

5.4.1 Learning algorithms

Low-level policy learning

The low-level policy is learnt using the DDPG algorithm with HER goal-relabelling using the episode sampling strategy with $k = 4$ (details in subsection 4.2.1). The learning process of the low-level policy benefits from the $\mathbf{A}^2$ method proposed in chapter 4. The implementation is exactly the same as that conducted in chapter 4, except that the *trajectory extrapolation* trick is discarded. Instead, HER is applied to segments of a trajectory from a new subgoal till it is achieved or the episode ends. For example, if two subgoals are involved in an episode, the trajectory is divided into two for the low-level policy and HER is applied twice. This is because the low-level policy is no longer required to master the whole manipulation task from the beginning state. It is now specialised in completing each subtask individually by achieving a sequence of desired low-level goals, while how the low-level goals are ordered is concerned by the high-level policy.

High-level policy learning

As explained in subsection 5.2.3, there are normally two ways to train a high-level policy: SMDP-Q learning or IOL (Sutton *et al.*, 1999b). Simply put, SMDP-Q only uses the transition collected when an option is terminated, which in this chapter, refers to when a low-level goal is achieved. On the other hand, IOL collects every transition even if the option is not terminating, and uses them altogether to update the high-level policy. This makes IOL more sample efficient than SMDP-Q learning (Sutton *et al.*, 1999b). Therefore, the following will introduce how to adapt the IOL algorithm to train the universal high-level policy with batch optimisation. The new algorithm is called goal-conditioned intra-option learning (GIOL).

The core of IOL is to calculate the *option value upon arrival* (Eq. 5.2), which is extended to the UOF high-level policy as follows. Given a high-level transition, $\xi_n^H = \{s_n, g_n^{H+}, a_n^H, r_n^H, b_{n,g_{a^H}^{L+}}, s_n'\}$, the estimated *goal-conditioned option value upon arrival*, $\tilde{u}$, for the next state s_n' under the desired high-level goal g_n^{H+} can be obtained by modifying the terms in Eq. 5.2 to be goal-augmented components:

$$\begin{aligned}&\tilde{u}_{\boldsymbol{w}^-}(s_n', g_n^{H+}, a_n^H)\\&\quad= (1 - b_{n,g_{a^H}^{L+}})\, \tilde{q}_{\boldsymbol{w}^-}(s_n', g_n^{H+}, a_n^H) + b_{n,g_{a^H}^{L+}} \max_{a^{H\prime}} \tilde{q}_{\boldsymbol{w}^-}(s_n', g_n^{H+}, a^{H\prime})\end{aligned} \tag{5.4}$$

where, $b_{n,g_{a^H}^{L+}}$ is the deterministic termination condition (a binary value) indicates whether the low-level goal, $g_{a^H}^{L+}$, associated with the high-level action is achieved, and $\boldsymbol{w}^-$ indicates the q value is calculated by a target q network. Accordingly, the minimisation objective for updating the universal high-level q network is estimated by modifying the components in Eq. 5.3 to be goal-augmented:

$$\begin{aligned} J(\boldsymbol{w}) &\approx \mathbb{E}_{\xi_n^H \sim \mathcal{D}} \left[\frac{1}{2} \hat{q}(s_n, g_n^{H+}, a_n^H) - \tilde{q}_{\boldsymbol{w}}(s_n, g_n^{H+}, a_n^H))^2 \right] \\ &\approx \mathbb{E}_{\xi_n^H \sim \mathcal{D}} \left[\frac{1}{2} (r^H + \gamma \tilde{u}_{\boldsymbol{w}^-}(s'_n, g_n^{H+}, a_n^H) - \tilde{q}_{\boldsymbol{w}}(s_n, g_n^{H+}, a_n^H))^2 \right] \end{aligned} \tag{5.5}$$

where, $\boldsymbol{w}$ indicates the parameters of the main q network. Similar to other RL algorithms, the GIOL algorithm also iterates between data collection, policy evaluation and improvement. The universal q functions for the state, high-level goal and high-level action are updated using Eq. 5.5. The universal high-level policy is obtained by taking the argmax operation over the estimated q values given a state and a desired goal:

$$a^H = \pi_g^H(s, g^{H+}) = arg \max_{a^H} \tilde{q}_{\boldsymbol{w}}(s, g^{H+}, a^H)$$

Exploration

The low-level policy will use the $\mathbf{A}^2$ implementation for the DDPG agent, described in subsection 4.2.4 with the same parameter setting. In the next subsection, analyses will be given to justify why applying adaptive exploration to the low-level policy not only accelerates the learning of the universal option, but also is essential for parallel training and important for the universal high-level policy.

The high-level policy will use the episode-wise exponential decaying ϵ-greedy exploration: $\epsilon = \epsilon_{end} + (\epsilon_{start} - \epsilon_{end}) \times e^{\frac{-k}{\beta_\epsilon}}$, where k is the number of passed episodes. It decays ϵ from 1.0 to 0.02, with different values of β_ϵ for different tasks, and keeps the same ϵ within an episode.

Networks and training details

With the above algorithms, both levels are to be updated after every *episode*. For the low-level policy, its trajectories are processed by HER before sampling after every episode. The high-level policy does not use HER. Both levels use a second critic to reduce overestimation. Both levels apply mean-deviation input normalisation with historical statistics. Both levels apply value clip. The target action values of the low-level policy are clipped in $[-25, 0]$, while that of the high-level policy are clipped in $[-T, 0]$, where T is the maximal number of interaction steps of an episode (see subsection 5.5.1). Both levels are updated 40 times after each cycle with a batch size of 128, a learning rate of $1e-3$, by the Adam optimiser (Kingma and Ba, 2014). A discount factor of 0.98 is used for all tasks. All target networks are updated after each optimisation step using Eq. 3.26 with $\tau = 0.1$. All networks are composed of three fully-connected layers of size 256 activated by ReLU. The final layers of all q networks are not activated, while the final layers of all actor networks are activated by Hyperbolic tangent (Tanh).

5.4.2 Tackling non-stationarity

As mentioned in the introduction, one of the aims of this chapter is to investigate the non-stationarity that occurred to the high-level policy during the parallel training processes of HRL agents, including the proposed UOF agent. The issue can be revealed by taking a close look at the transition function of the high-level policy.

In the UOF agent, or any HRL system, the next system state is determined by both the high-level and the low-level policy. Specifically, the action that interacts with the environment is selected by the low-level policy, which is selected by the high-level policy. This phenomenon is irrelevant to how the hierarchical architecture is designed and how both levels are represented and connected.

Denote the policies as $\pi^H(a^H|s)$ and $\pi^L(a^L|s, a^H)$, the system dynamic transition distribution according to the high-level and low-level actions as $p(s'|s, a^H)$ and $p(s'|s, a^L)$, then the probability of the next state of the system is written as:

$$\begin{aligned} p(s') &= p(s)\ \pi^H(a^H|s)\ p(s'|s, a^H) \\ &= p(s)\ \pi^H(a^H|s)\ \pi^L(a^L|s, a^H)\ p(s'|s, a^L) \end{aligned} \tag{5.6}$$

According to the recurrent relationship, the value function distribution accounted for the state distribution for the high-level policy can be written as:

$$\begin{aligned} v^{\pi^H}(s) &= \sum_{a^H} \pi^H(a^H|s) \sum_{s',r} p(s'|s, a^H)\,[r + \gamma v(s')] \\ &= \sum_{a^H} \pi^H(a^H|s) \sum_{a^L} \pi^L(a^L|s, a^H) \sum_{s',r} p(s'|s, a^L)\,[r + \gamma v(s')] \end{aligned} \tag{5.7}$$

Now consider Eq 5.6 and 5.7 in cases with pre-trained low-level policies and in parallel training scenarios. First of all, when the low-level policy is pre-trained, no exploration is performed at the low level when training the high-level policy. Looking at Eq. 5.6, the low-level policy can in fact be regarded as a part of the system dynamics, and the high-level policy can be trained just as any non-hierarchical RL systems. The value $v^{\pi^H}(s)$ can be estimated from the data collected, either on- or off-policy, as all distributions involved are unchanging, except the high-level policy that is to be improved.

However, when training in parallel, the value distribution becomes non-stationary as the transition distribution becomes non-stationary, because of a changing low-level policy. This impedes the learning of the high-level value function in two ways. First, as the low-level policy is being updated, the value distribution is no longer stable. The estimation target is moving in the space of distributions, making the optimisation problem ill-defined. This does not happen in cases with a pre-trained low-level policy. In fact, before the low-level policy has been well-trained, the high-level policy will struggle to learn at all. This is precisely why previous works propose to modify the collected trajectories so that they are corrected to the situation with an optimal low-level policy (Nachum *et al.*, 2018; Levy *et al.*, 2019). Secondly, the low-level policy requires a certain amount of exploration to learn about the task and the environment, as any other RL algorithm. This means that, in parallel training, there is always a chance the low-level policy will deviate from the correct trajectory even if it has been well-trained. Considering the multi-step tasks concerned in this chapter, the compounded probability of deviation from the correct trajectory of several subtasks keeps the high-level policy away from any meaningful learning experiences.

With these analyses, this chapter proposes to apply the $\mathbf{A}^2$ method developed in chapter 4 to the low-level policy. The main reason is that the adaptive exploration strategy will reduce exploration as much and as soon according to the performance that it achieves at different subtasks. In other words, in parallel training, the low-level policy will nearly stop exploration and stay as a stable distribution when it performs well. Thus, the high-level transition dynamic is stabilised as soon as possible to enable the learning of the high-level policy. In the meantime, the use of abstract demonstrations also benefits the high-level policy because the correct sequences of subtasks are in effect the correct high-level actions to be taken. Along with adapted low-level exploration, it will enable the high-level policy to proceed to later subtasks as fast as possible.

5.4.3 Summary

To sum up, this section first explains what algorithms are used to update the parameters of the policies and value functions for both levels in the UOF. Specifically, DDPG is used to learn the universal option policy, and the intra-option learning algorithm is extended to goal-conditioned cases in this section, named GIOL, for learning the universal high-level policy. Secondly, the dynamic process of training the high-level policy in parallel with the low-level policy is mathematically analysed. By looking at the root of the non-stationarity issue, this chapter proposes to use the $\mathbf{A}^2$ methods to stabilise and accelerate learning for the high-level policy. In principle, this idea can be applied to HRL architectures other than the proposed UOF, as the non-stationarity issue is not unique to UOF. To help understand, the pseudo-code of the training process is summarised in Algorithm 2.

Algorithm 2 Parallel training pseudo-code for UOF

```
Input: maximum epochs, cycles and episodes $M_0, M_1, M_2$
Initialise GIOL , DDPG and $\mathbf{A}^2$
for epoch = 1 to $M_0$ do
    for cycle = 1 to $M_1$ do
        for episode = 1 to $M_2$ do
            Sample a high-level goal
            for $t = 0$ to $T - 1$ do
                if use_demonstrations (subsection 4.2.3)
                    Obtain the correct next low-level goal
                else
                    Sample $a^H$ related to a low-level goal from $\pi_g^H$
                end if
                while not low_level_goal_achieved
                    Sample $a^L$ from $\pi_g^L$ with adaptive exploration
                    Execute $a^L$ and observe the next state and rewards
                    Store the transition
                end while
            end for
        end for
        Perform HER on the low-level trajectory
        Update the universal option policy with DDPG
        Update the universal high-level policy with GIOL
    end for
    Evaluate $\pi_g^L$ for each task
    Update the adaptive exploration parameters
end for
```

5.5 Empirical Results

This section will provide details of experiment tasks, and ablative and comparative studies. Recall that the tasks of interest are multi-step and multi-outcome manipulation and how to improve HRL for these tasks. All performances shown in this section are averaged over three random seeds. The main focus of the following experiments is then to answer the following questions:

(1) Can parallel training be accelerated by applying the $\mathbf{A}^2$ method?

(2) Is parallel training comparable to separate training?

(3) How does $\mathbf{A}^2$ stabilise parallel learning compared to hierarchical actor-critic (HAC) (Levy *et al.*, 2019)?

(4) Does UOF perform comparably with specialised high-level policies?

5.5.1 Task design

This subsection will introduce the tasks used to evaluate the framework and algorithms, eight block stacking tasks with a 7-DOF Fetch robot based on the Mujoco engine are used. In particular, there are four basic tasks for ablation studies and four additional tasks to test the limit and generalisation ability of the method.

Table 5.1 includes the configurations of four basic block-stacking tasks. Task 2 is used in particular to evaluate questions (1) and (2). Tasks 1 and 2 are used to evaluate question (3). Tasks 2, 3 and 4 are used to evaluate question (4).

In these tasks, the robot is asked to pick and place some blocks to build towers. Three blocks of different colours (Red, Blue, and Green) are involved. For each task, there is a different number of final outcomes. To achieve each outcome, different subtasks are required to be executed in different orders.

Task	Blocks	No. of steps	Desired outcomes	Training epoch	Training timestep	Testing timestep
1	R, B	3	B→R	150	25	50
2	R, B, G	6	B→R; G→R	800	25	50
3	R, B, G	15	B→R; B→G; R→B; R→G; G→R; G→B	1000	25	50
4	R, B, G	10	B→G→R; G→B→R	1500	40	60

Table 5.1: Basic Block-stacking Tasks.

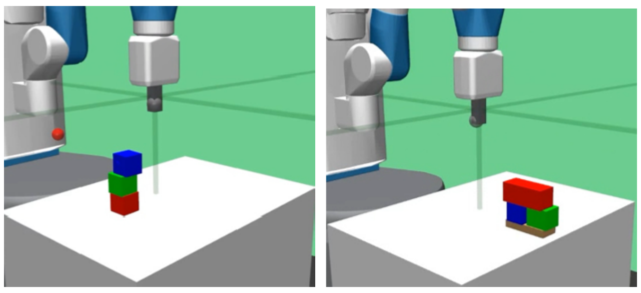

(a) (b)

Figure 5.3: Example task visualisation. (a) The 'B→G→R' outcome of the fourth basic task; (b) the 'R→BG' outcome of the pyramid task.

For example, 'B→G→R' is an outcome that denotes the desired top-down order of three blocks of task 4 (Figure 5.3a). This 'B→G→R' outcome will require five subtasks: 1) grasp the block G, 2) place G on the top of R, 3) grasp the block B, 4) place B on top of G and R, and 5) move the gripper back. Because the agent has to learn another outcome 'G→B→R', there are in total 10 subtasks in task 4. 'Training timestep' and 'Testing timestep' represent the number of total interaction steps that the agent is allowed to perform in a training or testing episode.

Table 5.2 lists the four additional tasks that are more complex than the basic tasks described above. The first additional task, *Pyramid*, is to showcase that the proposed method can achieve a different stacking type (pyramid). The second one, *Rotation*, is a variant of the basic task 1 that allows the algorithm to rotate the gripper about the z-axis. This makes the problem more difficult with an extra degree of control freedom. The last two additional tasks, *Random block size (RBS)* 1 and 2, are for testing the generalisation ability of the trained agents by randomising the sizes of the blocks. In particular, the agent is trained on the basic task 1 and 2, then evaluated on the additional task RBS 1 and 2 without further training.

Task	Blocks	No. of steps	Desired outcomes	Training epoch	Training timestep	Testing timestep
Pyramid	R, B, G	14	BG→R; R→BG	2000	60	80
Rotation	R, B	3	B→R	300	25	50
RBS 1	R, B	3	B→R	-	-	50
RBS 2	R, B, G	6	B→R; G→R	-	-	50

Table 5.2: Additional Block-stacking Task. RBS: random block size.

5.5.2 Parallel training improvement

The effects of A^2 in parallel training

As discussed in sectioon 5.1, the benefit of parallel training is twofold: it reduces repetitive computation as both levels of policies are trained in the data collection loop, and it avoids unnecessary fine-tuning of the low-level policy that is required by separate training scheme. Although it tends to be non-stationary due to a constantly exploring low-level policy, in subsection 5.4.2, the root of this issue is examined and the $\mathbf{A}^2$ method is proposed as a solution. This subsection then seeks to empirically answer the question (1): can parallel training be accelerated by the $\mathbf{A}^2$ method?

First of all, similar to the experiments discussed in chapter 4, the performances of the high-level policy are evaluated with different proportions of the episodes being demonstrated. In this study, the low-level policy is trained in parallel with the high-level policy using full $\mathbf{A}^2$ support, as described in subsection 5.4.1. The aim is to investigate how abstract demonstrations benefit the high-level policy. Note that the policy has to learn both outcomes of the basic task 2.

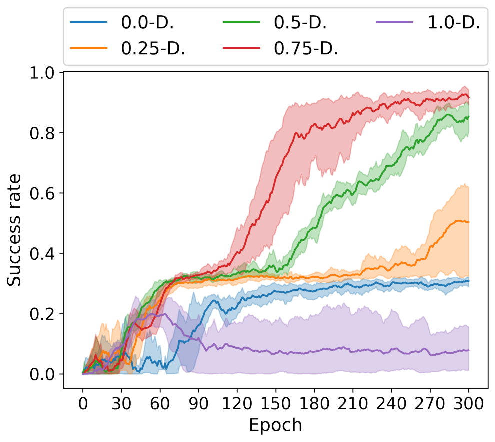


Figure 5.4: Average success rates of the universal high-level policy with different proportions of demonstrated episodes in the basic block stacking task 2. 0.0-D, 0.25-D, 0.5-D, 0.75-D, and 1.0-D denote the respective proportions of demonstrations added in the episodes.

From Figure 5.4, it is shown that the high-level policy has a hard time learning without abstract demonstrations (blue line), while with 75% of the episodes being demonstrated, the high-level policy can achieve near-optimal performance within 300 epochs. However, it also shows that with all episodes being demonstrated, the high-level policy actually performs worse (purple line). This is likely caused by the lack of exploration needed to maintain the diversity of the collected data. Similar to what happened in the experiments in subsection 4.3.2, the neural network may overfit to a narrow distribution rather than the one that it is supposed to learn. However, abstract demonstrations are indeed essential to the learning of the high-level policy.

Secondly, by giving the high-level policy demonstrations in 75% of the episodes, a comparison is made to see how fast the high-level is learning with and without the low-level policy using adaptive exploration. In this experiment, the baseline only differs from the full agent in that it uses the original ϵ-Gaussian exploration (Eq. 4.9), without adapting the exploration parameters.

Figure 5.5 shows the overall success rates of the two agents. Clearly, the agent with adaptive exploration at the low-level policy outperforms the baseline in terms of convergence speed, performance and variance. This improvement in fact comes from the improvement of the low-level policy, which can be proved by looking at the performance of the universal option policy. Figure 5.6 (a) and (b) show the three subtasks and the performance of the low-level policy in terms of how many low-level actions are spent to achieve them. A clear reduction of the number of required low-level actions is achieved by the agent using adaptive exploration. It also achieves faster convergence. These results altogether demonstrate that adaptive exploration indeed accelerates the learning of low-level policy in such multistep tasks, and it then helps to stabilise and accelerate the learning of the high-level policy.

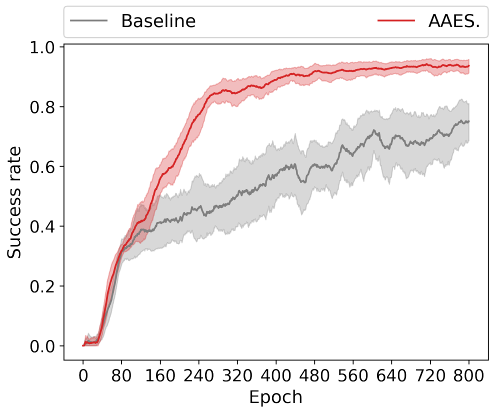
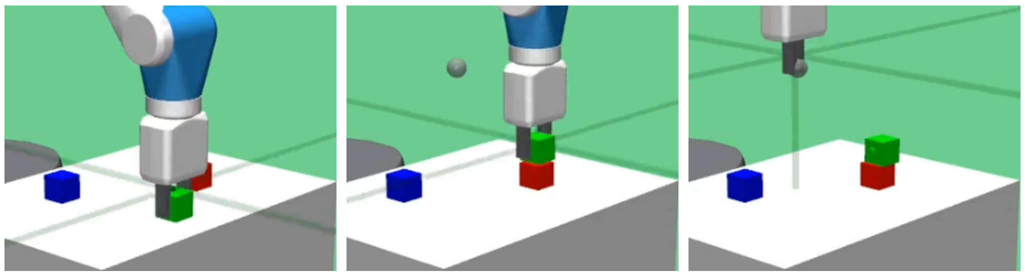
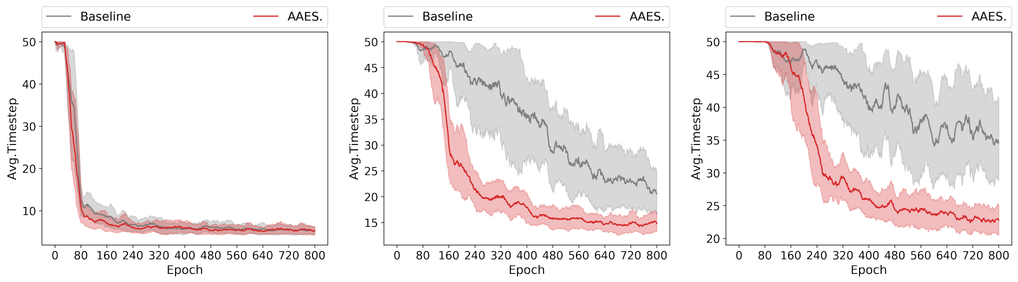


Figure 5.5: Average success rate of high-level policy performance for task 2. AAES: the agent with adaptive exploration strategy applied to the low-level.



(a)



(b)

Figure 5.6: Visualisation and the averaged number of required actions for the three consecutive subtasks of task 2. (a): Visualisation; (b) The average number of low-level actions needed to achieve the subtasks as training proceeds.

In sum, these two ablation studies provide an answer to the first question. Indeed, the results altogether have proved that parallel training at both levels can be stabilised and accelerated by the $\mathbf{A}^2$ method introduced in chapter 4. In particular, it proves the importance of adjusting low-level policy exploration when training the framework in parallel. As the motivation for applying the methods is not the subject architectural design of the HRL agent, this idea has the potential to be extended to the parallel training of other HRL frameworks in the future.

Compared to separate training

For question (2), the experiments are conducted to compare the high-level learning performances in parallel training and the learning with a pre-trained universal option. For the separate training baseline, the universal option is pre-trained with abstract demonstrations for 300 epochs, after which it reaches near-optimal success rates for all subtasks. The separate training baseline then starts training the high-level policy with the universal option being fixed. All other training details are the same.

Figure 5.7 displays the performances of this experiment. It shows that separate training (grey line) learns faster at the beginning yet fails to further improve its performances. Separate training also exhibits a larger variance in performance as the grey-shaded area is bigger. On the contrary, parallel training (red line) starts slower as it learns from zero, but is able to achieve near-optimal performance with a much smaller variance. These demonstrate the benefits of parallel training over separate training in terms of performance. In addition, the separate training agent requires the universal option to be pre-trained in advance for another 300 episodes, causing unnecessary data collection and computation in this case.

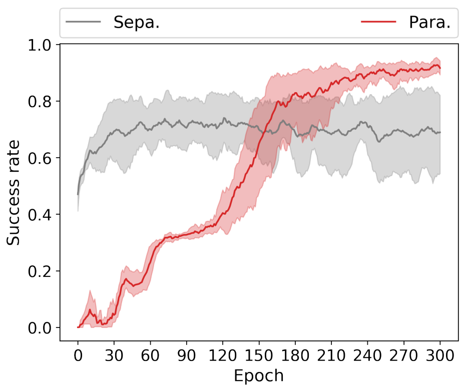
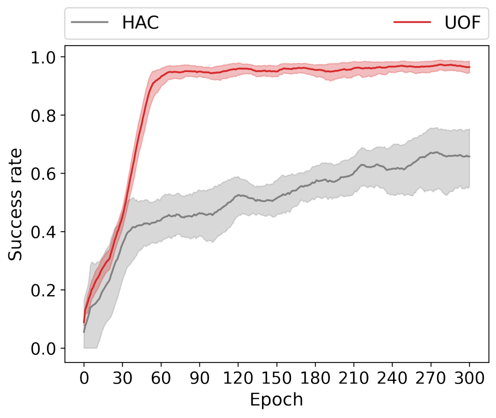


Figure 5.7: Average success rates of the universal high-level policy. **Sepa.**: Trained with a pre-trained universal option; **Para.**: Trained in parallel with the universal option.

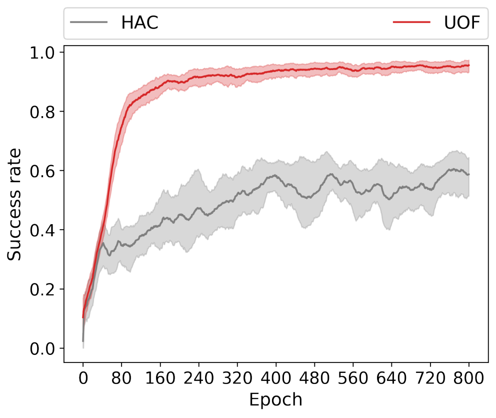
In sum, the answer to question (2) is that parallel training does outperform training with pre-trained low-level policies in terms of higher performances and lower computation costs. This is however under the condition that the non-stationary MDP dynamics of the high-level policy can be stabilised. Once stabilised, parallel training allows the low-level policy to adapt to the need of the high-level policy and enables the high-level policy to start learning as soon as possible. Thus, when the non-stationary problem can not be resolved or the pre-trained low-level policies are to be reused for many tasks, separate training may still be preferred. Nevertheless, the experiment and the literature (Li *et al.*, 2019) suggest that fine-tuning is important, in which cases adapting the low-level policy's exploration may again need to be adjusted to eliminate the non-stationarity issue.

Compared to hierarchical actor critic

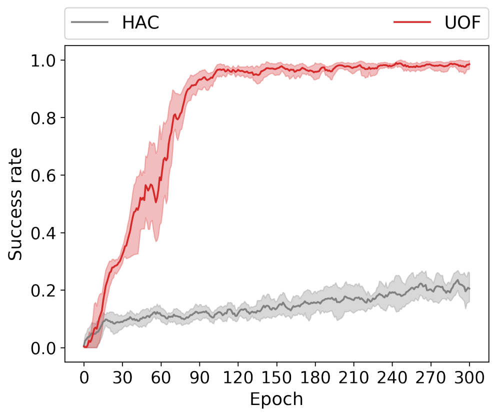
To answer the question (3), the $\mathbf{A}^2$-aided UOF is compared to the HAC agent (Levy *et al.*, 2019). HAC is also a goal-conditioned HRL framework, for which the authors propose three modifications to the collected experiences to deal with the non-stationary transition problem. To make the comparison as fair as possible, abstract demonstrations are also provided to the HAC agent.



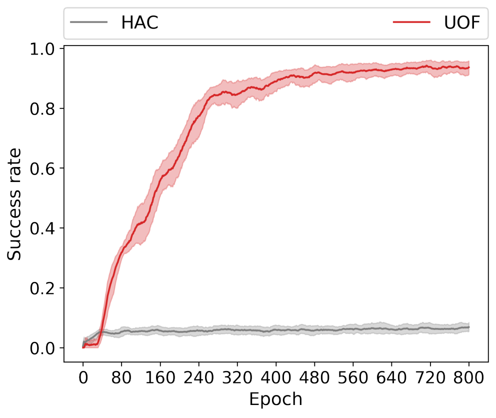
(a) Task 1 - Low-level



(b) Task 2 - Low-level



(c) Task 1 - High-level



(d) Task 2 - High-level

Figure 5.8: Average success rates of HAC and UOF.

Figure 5.8 displays the averaged success rates of both levels of the agents in tasks 1 and 2. Overall, UOF has achieved substantially better performances than HAC in all aspects. This implies a least three points. First, as

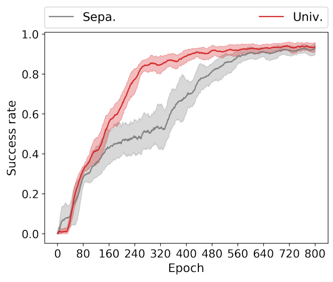
HAC uses a continuous action space for the high-level policy, it is too difficult to learn to generate subgoals in the continuous space. This demonstrates the significance of task decomposition. Secondly, the low-level policy of the HAC agent also learns much slower than UOF. This suggests that the adaptive exploration method is essential in long-horizon multi-step task learning. Thirdly, the slow learning progress of the high-level policy also implies the insufficiency of the solutions proposed in (Levy *et al.*, 2019) to deal with transition non-stationarity. On the other hand, the $\mathbf{A}^2$ method cuts straight to the root of the non-stationarity issue and improves high-level learning significantly.

Summary

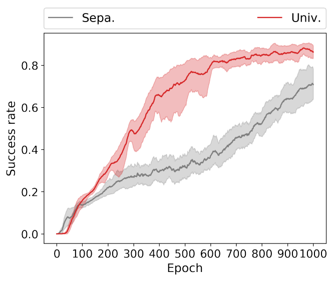
To sum up, this subsection focuses on the parallel training results of the UOF aided by the proposed $\mathbf{A}^2$ method. Experiments show that the non-stationarity problem discussed in subsection 5.4.2 can be substantially removed. Additionally, with the help of $\mathbf{A}^2$, the high-level policy actually outperforms separate training and the HAC baseline. Overall, these results demonstrate that parallel training is possible and can even be more efficient when performed with the right training techniques.

5.5.3 Learning multiple outcomes

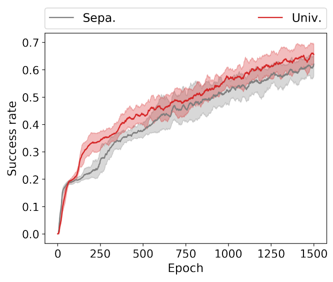
As parallel training is resolved, this subsection looks into the performance of universal policies and separated policies. As the aim of the chapter, the UOF agent is tasked to learn multiple outcomes with only one universal high-level policy. In the following, it is compared to a set of separated non-goal-conditioned policies, each of which uses the same universal option to achieve only one outcome of a task. In terms of training, each separated policy is an individual neural network with its own replay buffer. Note that this is large memory consumption. The performances of achieving all outcomes are averaged and compared.



(a) Task 2



(b) Task 3



(c) Task 4

Figure 5.9: Average success rates of achieving the final outcomes with universal and separated policies for the basic task 2, 3 and 4. **Univ**: universal high-level policy; **Sepa**: separated high-level policies.

Figure 5.9 shows that the universal policy can perform similarly or better than the combination of a set of separated policies. In particular, they achieve similar performances on task 4, while UOF learns faster in tasks 2 and 3. It is interesting to see that the universal policy performs better in tasks with more outcomes to learn. This may be due to the fact that a universal policy has the potential to share knowledge among different outcomes, making it learn faster to achieve similar tasks. However, when the task horizon becomes longer, separated policies can be advantageous because it has fewer final tasks to learn about, therefore requiring a smaller amount of data, as shown by Figure 5.9c.

These results answer the last question: the universal high-level policy actually performs comparably or even better than separated policies, especially in tasks with many outcomes.

Task	No. Outcomes	Policy type	Network size	Buffer size
2	2	Universal	$\sim$1.12 Mb	$\sim$2.42 Gb
		Separate	$\sim$2.24 Mb	$\sim$4.84 Gb
3	6	Universal	$\sim$1.12 Mb	$\sim$2.42 Gb
		Separate	$\sim$6.72 Mb	$\sim$14.52 Gb
4	3	Universal	$\sim$1.31 Mb	$\sim$2.60 Gb
		Separate	$\sim$2.26 Mb	$\sim$5.20 Gb

Table 5.3: Approximate memory requirements for training the high-level policies for task 2, 3 and 4.

Lastly, to demonstrate how much memory and computation can be saved by using universal policies, Table 5.3 displays the neural network parameter sizes and the replay buffer sizes for training a universal policy and a set of separated policies for the same number of outcomes. From the table one can see the memory requirement for separated policies grows as the number of outcomes grows. This may be acceptable in the state-based observation experiments conducted in this chapter. However, the requirement will become prohibitive very soon when the observations are high-dimensional, such as point clouds or images.

In sum, this subsection demonstrates that universal policies can achieve similar or better performances over individually trained policies, with the advantages of potential knowledge sharing and memory usage reduction.

5.5.4 Additional tasks

This subsection gives a brief report on the performances of the proposed UOF and its training methods on the four additional tasks. The aim is to briefly show that the proposed framework and methods have the ability to cover more complex tasks and also to discover its limitations.

First of all, the UOF agent achieves an average success rate of 0.6 at the low level and 0.4 at the high level for the Pyramid task. The performances are not optimal because of the longer task horizon with more consecutive subtasks. Each outcome of the task requires 7 subtasks to be achieved in the correct order, which poses a huge search space. This result exhibits the limitation of the framework in the face of longer task horizons and more consecutive task steps. Future works may focus on a better representation of the high-level action space that does not grow combinatorially as the number of blocks and outcomes grows.

Secondly, the agent is able to achieve near-optimal performances on the rotation task similar to the basic task 1. Note the rotation task is a variant of task 1 with an extra degree of gripper control freedom. However, it can only achieve it with a doubled training time. This suggests the importance of action space design for continuous RL algorithms, as an extra degree of control freedom may introduce much difficulty in the learning processes.

Lastly, the agent trained on the basic tasks 1 and 2 is evaluated without further training on the same tasks with randomised block sizes. The agent is able to achieve a success rate of 0.66 for both tasks with 30 testing episodes. However, failures occur mostly at the extremes of the range of the block sizes. In other words, when the block is too large or too small compared to the ones used in training, the agent is more likely to fail. Nonetheless, this experiment shows that the trained agent has a certain degree of generalisability. For better performances on unseen tasks, a fine-tuning process is recommended.

5.6 Summary

To conclude, this chapter develops a new HRL framework, named *universal option framework (UOF)*, to address long-horizon, multi-step, multi-outcome and sparse reward manipulation tasks. In short, UOF is created by combining the GRL paradigm and the classic option framework for HRL. In order to accelerate and stabilise the parallel learning of the UOF for such difficult tasks, this chapter looks into the theoretical root of the parallel training non-stationarity problem and proposes to leverage the $\mathbf{A}^2$ method as a fix.

Experimental results of a series of simulated block stacking tasks demonstrate that 1) it is possible to use universal policies to learn multiple steps and multiple outcomes, saving much memory and computation resources, and 2) it is possible to eliminate the non-stationary transition issue in parallel training and obtain substantial learning speed and performance improvements.

There are a few insights that could be drawn from the work in this chapter. First of all, it is possible to learn multiple subtasks and tasks with only universal policies in the HRL scenario, however, this may sacrifice the learning efficiency. Therefore, it is up to the engineer to determine whether it is worth the risk to trade learning efficiency for a smaller memory consumption. Secondly, the task decomposition assumption plays a vital role in the design of the HRL architecture and the acceleration methods for its training. Similar to what is discussed in chapter 4, it is very demanding to develop learning-based task decomposition methods for cases where manual decomposition is difficult. Lastly, the framework is only evaluated on state-based tasks in this chapter. Future works are required to validate and improve the proposed methods on tasks with image or point cloud observations. The insight here is that a good abstraction of the action and observation space can lift burdens from the learning algorithm.

Chapter 6

Contact-Rich Grasping and Manipulation with General Affordance

6.1 Introduction

The previous chapters have put much effort into developing and accelerating RL-based algorithms for long-horizon manipulation tasks. In those two chapters and many other works, an important assumption is that the algorithm is given a known grasp pose for the target object, and the algorithm is only concerned with how to generate the robot motions. This assumption may suffice for simple manipulation tasks without rich contacts, such as simple pick-and-place. However, tasks such as tool use, disentangling objects, hanging objects or some assembly tasks would impose stricter constraints on grasp pose selection. They require the robot to select grasps that are not only stable but also enable better manipulation performances. Such tasks require the manipulation system to understand and leverage the relationships between the environment, grasp generation and the manipulation tasks. In other words, this chapter is particularly interested in the following challenge:

How to select grasp poses that enable better manipulation performances?

The problem is closely related to the problem of *task-oriented grasps* (TOGs) generation, which seeks to generate grasps that are suitable for the following manipulation tasks. As discussed in detail in subsection 2.2.4, existing methods either seek to classify task-agnostic grasps (TAGs), or try to predict TOGs directly. This chapter also proposes to rely on off-the-shell TAG generators for TAG planning. However, the aim of the research is taken one step further: to filter and select TAGs that can improve downstream manipulation, beyond enabling.

There is a major difference compared to existing works on TOG planning – the information used to filter and select TAGs. Specifically, previous methods only provide binary labels for the planned grasps, indicating whether they are suitable for certain manipulation tasks or not (Detry *et al.*, 2017; Kokic *et al.*, 2020; Murali *et al.*, 2021; Sun and Gao, 2021). These methods focus on generating grasps that enable manipulation. Unlike them, the aim of this research is taken beyond binary manipulation possibility prediction, seeking to generate grasps that improve manipulation performances.

To address the problem, this chapter draws a connection to the concept of affordances, something that is hypothesised to be one of the essential abilities behind human perception and action (McClelland, 2019). As introduced in the psychology research community, affordance is the knowledge about the interaction relationship between the actions of an agent and the environment (Gibson and Collins, 1982). Notice that, affordance is subject to the environment and an agent. In the robotic community, researchers tend to regard affordances as environment-dependent action possibilities and consequences (McClelland, 2019; Khetarpal *et al.*, 2020a; Ardón *et al.*, 2021). It naturally follows that a robot can use such knowledge to reduce the search space of actions into only affordable actions for planning or exploration (Khetarpal *et al.*, 2020a, 2021).

Recently, a promising affordance theory based on the RL paradigm has been proposed, leading to improved planning and value learning performances for RL algorithms (Khetarpal *et al.*, 2020a). However, the definition of *action consequences* is originally based on the state changes in the MDP. In this chapter, we first contribute to the field by proposing the concept of *general affordances* that describes the action-environment relationship beyond system state prediction. For example, the consequences of a grasp action may be defined as the probability of dropping the grasped object during the following manipulation, and the affordance then could be the subset of state and grasp actions that achieve a satisfactory chance of dropping an object.

In short, a robot would be able to use *general affordances* to select actions according to arbitrary user-defined consequences, leading to potentially more flexible and reliable action selections.

Following the theoretic development, this chapter develop a manipulation system that uses general affordances to select grasps with better-predicted manipulation performances. The system is named *general affordance-aware manipulation* (GAM). In GAM, the robot is trained by RL to manipulate the objects given task-agnostic grasps (TAGs). After the manipulation policy has been trained, a manipulation affordance prediction (MAP) module will be trained to predict the performances of the manipulation policy. Finally, the learnt MAP module will be used to filter away TAGs with undesired predicted manipulation performances. To demonstrate the concepts and potentials of the GAM framework, it will be implemented with three hook disentangling manipulation tasks in simulation. The tasks are typical examples of contact-rich manipulation, and GAM can be applied to many other manipulation tasks. The following will briefly review the literature regarding object disentangling tasks, and then illustrate the section organisation of this chapter.

6.1.1 Related works on object disentangling

The specific manipulation scenario that this chapter uses to test the proposed GAM framework is the task of disentangling entangled objects. For example, picking up an entangled hook-shaped object, as shown in Figure 6.1. Such tasks occur in many aspects of day-to-day life and industries, yet remain an underdeveloped area. The complexity is twofold: 1) different grasp point and position will result in different manipulation difficulty and result; 2) the rich contact dynamics makes it difficult to conduct planning, and therefore demands model-free learning algorithms to generate manipulation trajectories.

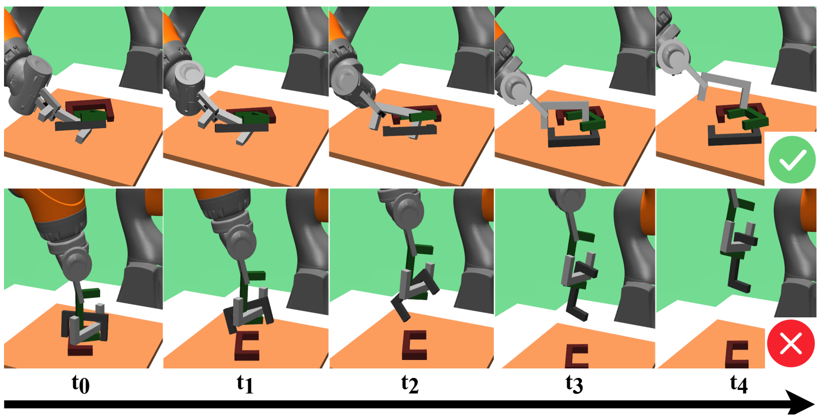


Figure 6.1: Examples of picking an entangled hook. In the first row, the robot manages to rotate and lift up only the white hook. While in the second row, the robot fails to separate the grasped green hook.

Previously, Matsumura *et al.* (2019) developed the first solution to pick up one object from a pile of potentially tangled objects. They proposed a DNN to predict whether or not a top-down grasp pose will result in picking up several tangled objects and use it as a filter to avoid picking from such grasp poses (Matsumura *et al.*, 2019). Similarly, Moosmann *et al.* (2020) trained a DNN to predict whether an object is free from entanglement during a straight lifting-up motion and avoid grasping entangled objects (Moosmann *et al.*, 2020). The same team further developed a supervised (Moosmann *et al.*, 2021b) and RL (Moosmann *et al.*, 2021a) approach to manipulate and separate entangled objects given task-agnostic grasp poses. Another recent work proposed a topological solution to compute entanglement score from a depth image and thus find top-down grasp poses that are free from entanglement (Zhang *et al.*, 2021). Another work proposes a sophisticated set of designed rules to recognise and model entangled tubes, detect entanglement and find a disentangling solution (Leão *et al.*, 2020).

In short, these methods exhibit two major limitations. First, they mostly use lifting-up motion trajectories that are not always sufficient to separate the entangled objects in all situations. Secondly, they mostly rely on a TAG generator for grasp planning, which is agnostic to the downstream separation manipulation task. To achieve higher disentangling success rates and performances, a manipulation system is required to incorporate 6 DoF TOG generation and model-free motion generation. Therefore, given its difficulties and practical values, such a contact-rich grasping and manipulation task serves as a good testbed for the proposed GAM framework.

6.1.2 Summary and chapter organisation

To sum up, this chapter studies the problem of filtering and selecting grasp poses to achieve better contact-rich manipulation performances. The proposed idea is to learn manipulation affordances with respect to task-agnostic grasps, after which the learned affordances can be used to select grasps with better-predicted manipulation performances and improve the actual manipulation results. In order to evaluate the proposed framework, experiments are conducted on a series of difficult hook disentangling manipulation tasks in simulation. The contributions/novelties of this chapter include:

- Extend the RL-based affordance theory to include the prediction of arbitrary action consequences, called *general affordances*.
- Based on the general affordance concept, develop a new manipulation framework that selects task-agnostic grasps according to predicted manipulation performances.
- Design and implement the training processes of the general affordance-aware manipulation (GAM) framework in a series of hook-disentangling

tasks in simulation.

- Demonstrate the effectiveness and substantial improvements over existing methods with the use of an affordance-based grasp filter.

The rest of the chapter is organised as follows. Section 6.2 will introduce the affordance theory in RL and its extension to include the predictions of arbitrary action consequences. Section 6.3 will illustrate the GAM framework and its implementation on the object disentangling task in detail, covering TAG generation, RL-based manipulation, manipulation affordance prediction and grasp filtering. Section 6.4 will describe the experiment design and discuss experimental results. Finally, section 6.5 concludes this chapter.

6.2 Affordance Theory in RL

As already mentioned in section 2.4.3, the concept of affordance refers to the knowledge of what actions are possible and what the consequences of the actions are with respect to (a part of) an object or environment as well as an agent (Gibson and Collins, 1982). For decades the study of the affordance of objects has been focused on modelling and understanding human perception and cognition (Pezzulo and Cisek, 2016; Masoudi *et al.*, 2019), designing affordable products (Masoudi *et al.*, 2019), the effects of social behaviours (Orban *et al.*, 2021), etc. The concept has also drawn the attention of the robotic and machine learning community (Ardón *et al.*, 2021). However, as the usages and references of the concept of affordance increase over time, there is a demand of unifying them in the robot learning community with a mathematical framework. Unsurprisingly, RL, given its great success and generality, was chosen by some researchers as a foundation to formalise a theory of affordance. There are several reasons why it may be preferred to base the framework on RL:

- RL has a general and developed mathematical formulation for decision-making and behaviour learning.
- The RL community has bore a set of algorithms that proved to be promising when aided by DL and ready to be extended.
- Empirical evidence has shown that the use of affordance could accelerate learning and planning, and help in high-level task reasoning, which is the bottleneck of most algorithms.

Part of the third contribution of this thesis is a short survey on the topic of deep robotic affordance learning (DRAL) through the lens of RL (publication [4]). This section will first introduce the theory of affordance in RL (ARL), provide a few remarks on the applications of the ARL theory, and finally proposes an extension to the concept called *general affordances*.

6.2.1 Definition

The theory is based on the general decision-making framework, MDP, which has been introduced at the beginning of this chapter. Before formulating affordances, an example would be helpful. Originally, Gibson and Collins (1982) saw it as a relationship between an agent and the environment, the action possibility and its consequences. For example, the handle of a hammer affords the grasping action of a teenager, and the consequence of conducting the grasping action is the hammer being grasped within the palm of the teenager. Notice that the grasping action is regarded as affordable only because its *desired consequence* can be achieved after applying it. This suggests that the affordance (possibility) of an action is better defined by whether the desired consequence of that action could be realised given a circumstance. To formalise this, the concept of *intent* is introduced to represent *the desired consequences of an action given a state* (Khetarpal *et al.*, 2020a).

According to Khetarpal *et al.* (2020a), an intent of an action a is a mapping from states to a distribution of states that is desired to be achieved by applying the action: $I_a : \mathcal{S} \to \mathcal{S}_a^+$. Consider a moving left action in a grid world, its intent always maps to the grid cell at the left. However, the moving left intent can only be achieved when there is an actual cell at the left, not when there is a wall on the left. In other words, the desired consequence of an action, i.e. its intent, is not always satisfied. It can only be satisfied when the system dynamic permits so. To generalise this notion a little: an intent of an action is satisfied according to distribution distance metric d and a threshold ϵ_I, if and only if

$$d(I_a(s), p(\cdot|s, a)) \leq \epsilon_I \tag{6.1}$$

where p represents the system transition distribution. Eq. 6.1 captures the affordance of an action in the sense of dynamic transition. It defines that an action is affordable at a state if the desired consequence of the action is aligned with the system dynamic (to a certain degree ϵ_I). The definition of affordances for an agent is straightforward. Given a set of intents for all actions and states in an MDP $\mathcal{I} = \cup_{a \in \mathcal{A}} I_a$, a distance metric and a threshold, the affordances for an agent is a subset of state-action pairs, $\mathcal{AF}_{\mathcal{I}} \subseteq \mathcal{S} \times \mathcal{A}$, such that $\forall(s, a) \in \mathcal{AF}_{\mathcal{I}}$, its intent is satisfied, i.e., Eq. 6.1 is satisfied.

6.2.2 Remarks

In practice, there are three interesting directions concerning the application and learning of this definition. The following remarks summarise them, which were partly included in the survey paper (publication [4]).

Remark 1: It is very useful if an algorithm can predict or is given prior knowledge of whether an action is affordable at a state. In other words, given a state-action pair, $(s, a)\ \forall s \in \mathcal{S}, a \in \mathcal{A}$ and a set of intents, the distribution of whether a state-action pair is within the affordance set: $p^{\mathcal{AF}_{\mathcal{I}}}(s, a) = p((s, a) \in \mathcal{AF}_{\mathcal{I}})$ is known. This is particularly preferred because knowing what actions are affordable accelerates the planning process and reduces unnecessary exploration on effectless actions (Khetarpal *et al.*, 2020a; Xu *et al.*, 2021; Mandikal and Grauman, 2021a) (see subsection 2.4.3 as well).

Remark 2: The definition gives a natural way to use affordance: estimate the action possibility $p^{\mathcal{AF}_{\mathcal{I}}}$ and use it to infer afforded actions. This then serves as the foundation to classify recent papers on the topic of DRAL:

- works that learn $p^{\mathcal{AF}_{\mathcal{I}}}$ from data and use it as a criterion to select actions.
- works that learn a mapping to afforded actions represented as object keypoints, i.e., find a set of keypoints for an object such that actions performed on these keypoints are affordable.
- works that learn $p^{\mathcal{AF}_{\mathcal{I}}}$ from data and use it to learn a partial dynamic model for only affordable actions.

Remark 3: There is a large space for further study of learning and using affordances. The potential benefits revealed by the definition above remain at the level of intermediate or one-step action possibility (in the sense of MDP). However, using the affordances of skills at a coarser time scale has a higher practical value as it contributes to the abstract level of planning for complex and long-horizon tasks. Application opportunities are many, such as combining affordances with HRL, planning with options, partial dynamic model with temporal abstraction (Xu *et al.*, 2021; Khetarpal *et al.*, 2021; Nica *et al.*, 2022).

Remark 4: There is a prerequisite for learning $p^{\mathcal{AF}_\mathcal{I}}$ and using it to infer affordable actions: a well-defined set of actions and their desired consequences. State-of-the-art research mentioned above assumes a given set of actions and their effects, based on which the action possibilities are learnt and used. However, it would be more intriguing if actions and their effects can be represented in a way that new actions and effects are allowed to be discovered. In practice, this is more valuable to think of the "actions" here in terms of skills. In other words, this brings us back to the interesting problem discussed in subsection 5.2.2: skill discovery in HRL. From the perspective of affordances, another interesting research direction is the evolving skill possibility. For example, a robot may slowly become incapable of performing certain motions as its hardware decays, or the other way around, new skills may be discovered when the environment changes.

6.2.3 General affordance

In this subsection, we develop the definition of *general intent* and *general affordance* in the context of MDP and RL, as part of the **third contribution of this thesis** (paper 5).

The definition above builds upon intent that captures the desired consequences of actions in terms of the desired next state distribution in an MDP (Khetarpal *et al.*, 2020a). The set of affordances is therefore induced by the comparison between the desired and true dynamic distributions of the next state. We now call this kind of affordance *dynamical affordances* and denote it as $\mathcal{AF}^p_\mathcal{I}$, where p refers to the system dynamic transition. Also, denote the intent of an action in terms of the desired next state distribution as the *dynamical intents* $I_a^p : \mathcal{S} \to \mathcal{S}_a^+$. Now this can be readily extended to multistep prediction. In other words, action consequences as the distribution of the desired state after multiple timesteps, which corresponds to temporal abstraction and HRL. It is the multistep generalisation of the one-step dynamical intents and affordances, or so-called *temporally extended intent* and *option affordances* (Khetarpal *et al.*, 2021).

Now we consider examples of non-dynamical action consequences, which tend to be long-term and delayed consequences. The first choice would be the distribution of the desired return, which corresponds to the q value prediction. More generally, it could be the desired values according to some certain measurement. Examples include the possibility of dropping an object, the amount of water poured into a cup, the fuel consumption of an autonomous driving car, etc. Notice that a policy π is now needed to be included as the long-term action consequence is always induced by some policy. Following this thought, the *general intent* can be defined with respect to a given measurement y.

Definition 1 (General intent $I_{a,\pi}^y$): *Given a measurement* $y : \mathcal{S} \times \mathcal{A} \to \mathcal{Y} \in \mathbb{R}^n$, *where* $\mathcal{Y}$ *is the space of all possible measurement values and* $\mathbb{R}^n$ *is the* n*-dimensional real number space, the general intent w.r.t. measurement* y *for a policy* π *is defined as a mapping to a subset of measurement values that are desired (or intended) to be achieved by taking action* a *and following the policy* π *thereafter, denoted as* $I_{a,\pi}^y(s) = y^{\pi+}(s,a) : \mathcal{S} \to \mathcal{Y}_{a,\pi}^+ \in \mathcal{Y}$.

Notice that, let y be the system transition function and $\mathcal{Y}$ be the state space, general intent degrades to the concept of dynamical intent. For the dynamical intent, there is a true system transition distribution to be compared. However, an arbitrary measurement may not have a true value distribution. Therefore, it is required to define a target distribution of the measurement value for comparison. In other words, the general affordances can come out from the comparison between the desired action consequences for a policy π and the baseline action consequences according to some other policy $\hat{\pi}$. Note that, it could be the optimal policy, or not.

Definition 2 (Intent satisfaction): *Denote $y^{\hat{\pi}}$ as the action consequences w.r.t. the measurement y of taking an action a and following the baseline policy $\hat{\pi}$ thereafter, given a distribution distance metric $d(\cdot)$ and a threshold ϵ_I, the general intent $I^y_{a,\pi}$ is satisfied according to the standard of $\hat{\pi}$ iif.:*

$$d(I^y_{a,\pi}(s), y^{\hat{\pi}}(s,a)) \leq \epsilon_I \tag{6.2}$$

In plain words, Def. 2 states that for an action at a state, given a desired action consequence for a policy π and the action consequence following a baseline policy $\hat{\pi}$ thereafter, that desired consequence can be satisfied to a certain degree if Eq. 6.2 holds. It could help determine to what degree a policy π can afford some action consequences: given the fuel cost $(y^{\hat{\pi}}(s,a))$ from Cardiff to London by a taxi driver (the baseline policy $\hat{\pi}$) leaving (the action) at 9 am (the state), would taking the same action of leaving at the same time by another driver (π) afford the same fuel cost? It could also help find the state where an action can afford certain consequences following the policy π according to the standard of $\hat{\pi}$: when (state) to leave (action) Cardiff and following π will afford the fuel cost induced by $\hat{\pi}$. Here, the action consequence is the measure of fuel consumption, but it could very well be travelling time, CO_2 emission, etc. Accordingly, the general affordance definition for a policy π comes out naturally as follows.

Definition 3 (General affordances $\mathcal{AF}^y_{\mathcal{I},\pi}$): *Given a set of general intents for a policy π for all actions and states in an MDP, $\mathcal{I}^y_\pi = \cup_{a\in\mathcal{A}} I^y_{a,\pi}$, a distribution distance metric and a threshold, the general affordances for an agent is a subset of state-action pairs, $\mathcal{AF}^y_{\mathcal{I},\pi} \subseteq \mathcal{S} \times \mathcal{A}$, such that $\forall(s,a) \in \mathcal{AF}^y_{\mathcal{I},\pi}$,* *its general intent (Eq. 6.2) is satisfied according to the standard of a baseline policy $\hat{\pi}$.*

It is not difficult to see that the one-step and multi-step dynamical affordances are special cases of the general affordances when the action consequences and the measurement are defined as dynamic transitions, and the baseline policy $\hat{\pi}$ is just the physic laws. Similar to these special cases, knowing the general affordances for an agent in terms of certain measurement metrics will help in planning and learning. For example, by learning to predict the travel time of a number of transports for a specific driver, one can rule out a subset of them that are not affordable compared to the baseline travel time with a threshold. In fact, multiple measurements may be learnt simultaneously to support diverse action selection strategies. Chapter 6 will demonstrate an example usage of general affordances in a robotic manipulation task.

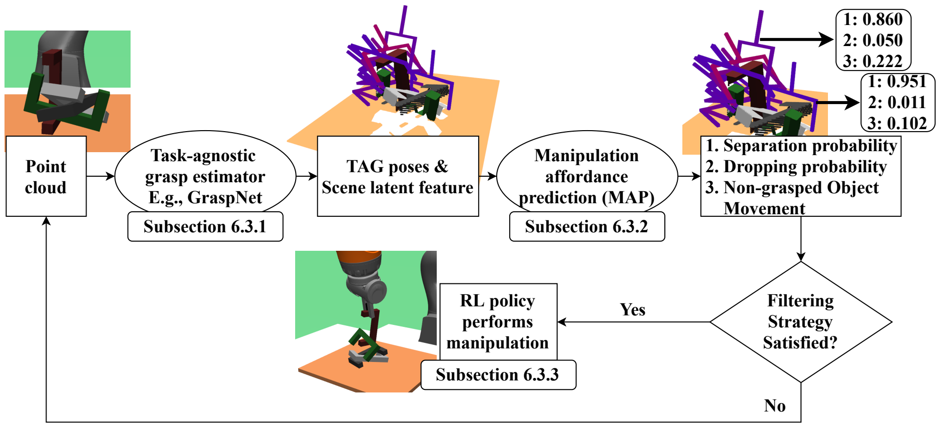
6.2.4 Summary

To sum up, this section introduces the foundation of the affordance theory in the RL framework. In order to describe affordances in a computationally useful way, Khetarpal *et al.* (2020a) proposes the notion of intent as the desired action consequences in terms of desired next state distribution. The affordance definition then follows as a subset of state-action pairs that satisfy the intents with respect to the ground true system dynamic given a threshold. Remarks are given regarding the learning and application of such affordances.

In addition, following this definition, this thesis then proposes to define *general intents, affodances* based on action consequences in terms of an arbitrary measure function of a state and action pair. The general affordance definition includes the special cases proposed for dynamical action consequences in (Khetarpal *et al.*, 2020a, 2021). In the next section, the proposed concept is applied to develop a manipulation framework.

6.3 General Affordance-aware Manipulation

This section will introduce the proposed GAM framework in detail. Recall that, the objective of this chapter is to improve complex and contact-rich manipulation performances through task-oriented grasp selection. Overall, the GAM framework is comprised of three main components: 1) a task-agnostic grasp generator (subsection 6.3.1), 2) the manipulation affordance-based grasp filtering module (subsection 6.3.2 and 3) the manipulation policy (subsection 6.3.3).



Point cloud
Task-agnostic grasp estimator E.g., GraspNet
Subsection 6.3.1
TAG poses & Scene latent feature
Manipulation affordance prediction (MAP)
Subsection 6.3.2
1: 0.860
2: 0.050
3: 0.222
1: 0.951
2: 0.011
3: 0.102
1. Separation probability
2. Dropping probability
3. Non-grasped Object Movement
Filtering Strategy Satisfied?
Yes
No
RL policy performs manipulation
Subsection 6.3.3

Figure 6.2: The overall workflow of the proposed general affordance-aware manipulation (GAM) framework. There are three components: 1) a task-agnostic grasp generator (subsection 6.3.1) that proposes stable TAG poses; 2) the manipulation affordance-based grasp filter (MAGF) that uses predicted manipulation performances to filter the TAG poses (subsection 6.3.2); 3) an RL manipulation policy that controls the robot arm to separate the grasped hook (subsection 6.3.3).

The workflow is depicted by Figure 6.2 and summarised as follows. The GAM framework first predicts a set of stable but task-agnostic grasps, along with the latent features of the points around the grasps. The grasps and their latent features are then used to estimate the manipulation performances that the manipulation policy may achieve. Finally, the manipulation policy performs the task using the grasps selected according to filtering strategies based on the predicted performances. The following will elaborate in more detail.

6.3.1 Task-agnostic grasp generation

Every manipulation system requires a grasp planning module, which predicts grasps given an observation of the task scene. In principle, any grasp planner can be integrated into the GAM framework. As 6 DOF grasp poses are largely preferred in complex manipulation tasks, this chapter employs a recently popular TAG generator called GraspNet (Fang *et al.*, 2020a).

In short, the GraspNet model takes into a partial point cloud of the task scene and predicts a set of 6 DOF stable TAGs (see Figure 6.3). It evaluates a downsampled set of 3D grasp points of the scene with a number of approaching vectors, approaching distances, in-plane rotations along the approaching vectors and finger widths. The prediction results consist of the transformation matrices of the grasps with respect to the camera frame along with their stability and robustness scores. In addition, to grasp planning, the pre-trained GraspNet model is used to obtain the latent features of the task scene associated with the grasps. The latent features are the outputs of the point auto-encoder network of the GraspNet model, based on the PointNet++ backbone network (Qi *et al.*, 2017). Implementation-wise, the experiments in this chapter use the pre-trained model open-sourced by the authors.

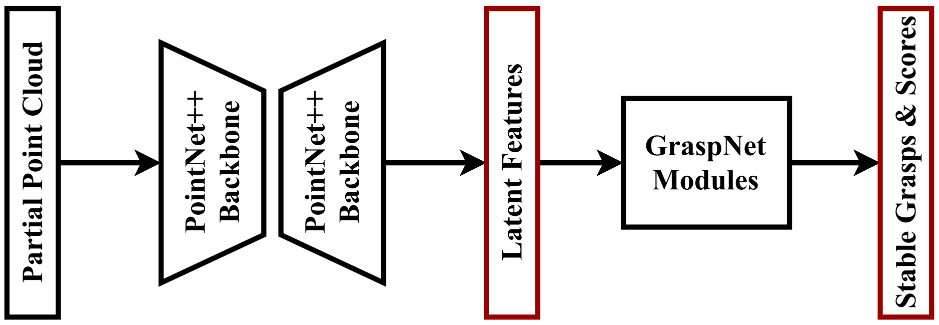


Figure 6.3: The (simplified) GraspNet architecture (Fang *et al.*, 2020a). It takes into the partial point cloud observation of the scene and processes the points with the PointNet++ backbone auto-encoder (Qi *et al.*, 2017). The latent features are used in the GraspNet to predict grasp poses and their stability and robustness scores. In this chapter, the latent features and the grasp poses are used together to predict the manipulation performances.

Notice an assumption made here is that the TAG generator is able to produce stable grasps, although in practice this is commonly not true. The GraspNet model adopted in this chapter is used directly without fine-tuning, therefore also achieves degraded performances compared to that reported in the original paper. Without further fine-tuning, the robot can grasp the objects with only $\sim 10\%$ of the generated grasps. This becomes a stepping stone for training the following manipulation policy and the affordance prediction model, because much time and memory are required to compute the forward pass of the GraspNet model. In order to facilitate faster training of the following modules, the experiments use the simulator and the GraspNet model to collect a dataset of pre-recorded scenes, stable grasp poses and their latent features for each task. Though fine-tuning is an alternative to improve the TAG generator performance, it is not necessary for proving the concepts of GAM in this chapter.

6.3.2 Manipulation affordance-based grasp filtering

This subsection illustrates the main contributive component of this section: the manipulation affordance prediction module and the associated grasp filtering strategies. This chapter proposes to understand and use the proposed general affordances concept developed in the last section.

To recall, the idea of *general affordances* extends the definition given by (Khetarpal *et al.*, 2020a) to include the knowledge about the consequences of actions in terms of any measurement of the user's concerns according to a baseline policy $\hat{\pi}$. Denote a measurement of action consequences of taking an action at a state and following a policy thereafter as $y^{\pi}(s, a)$, the *general intent* for the policy π is defined as the desired action consequence in terms of the given measurement $I^{y}_{a,\pi}(s) = y^{\pi+}(s, a)$. The general affordances with respect to a specific measurement is a subset of state-action pairs, in which the general intents of the actions can be satisfied at these states, compared to a baseline action consequences $y^{\hat{\pi}}(s, a)$. A natural baseline would be the true action consequences that can be achieved by a certain policy.

Implementation on grasping and manipulation

Before describing the learning and usage of the general affordances, the following contents first illustrate how to apply these concepts in the grasping and manipulation context. In this application, the manipulation system is regarded as a two-level system, where the high-level policy π^{H} seeks to select grasps and the low-level policy π^{L} seeks to manipulate the objects based on the given TAGs. This subsection looks at the high-level action selection and regards the low-level policy as a black box, which may be learnt through reward maximisation or a classic motion planner (see next subsection). The affordances to be learnt are named *manipulation affordances*, associated with a set of *manipulation intents*.

For the hook disentangling task, the measurement of the action consequences can be defined as some manipulation performance metric of the downstream manipulation task. We propose three measurements as follows:

(1) y_1: The probability of the grasped hook being separated.

(2) y_2: The probability of the grasped hook being dropped in manipulation.

(3) y_3: The averaged movements of the non-grasped hooks.

Define the baseline action consequences as the true manipulation performances that can be achieved by the low-level manipulation policy π^L, denoted as $y_1^{\pi^L}$, $y_2^{\pi^L}$, and $y_3^{\pi^L}$. The manipulation intents are then defined by specifying the desired values of the measurements, denoted as $y_1^{\pi^L+}$, $y_2^{\pi^L+}$, and $y_3^{\pi^L+}$. For the i-th measurement y_i, the manipulation affordances of its intents $\mathcal{AF}^{y_i}_{\mathcal{I},\pi^L}$ cover those state-action pairs whose true action consequences $y_i^{\pi^L}$ are close enough in arithmetic difference to the manipulation intents given a threshold ϵ_i. In order words, for the i-th measurement:

$$\forall\ (s^L, a^L) \in \mathcal{AF}^{y_i}_{\mathcal{I},\pi^L},\ y_i^{\pi^L+}(s^L, a^L) - y_i^{\pi^L}(s^L, a^L) \leq \epsilon_i.$$

where the manipulation performance is defined over the low-level state and action pairs, which are not readily applicable for the purpose of grasp selection at the higher level in our problem. In order to link the high-level actions to the affordable manipulation performances, we assume that a function exists to map the high-level actions to the initial low-level states, denoted as $\Phi : \mathcal{A}^H \rightarrow \mathcal{S}^L$. Such a function does exist in this case where a grasp pose selected by π^H will determine the starting configuration of the downstream manipulation task, i.e. s_0^L. In addition, we can omit the low-level actions in the equation as we focus only on grasp selection (low-level state selection). Hence, we can define the grasp (high-level action) is said to afford a certain manipulation performance iif. $y_i^{\pi^L+}(s_0^L,\cdot) - y_i^{\pi^L}(s_0^L,\cdot) \leq \epsilon_i$ where $s_0^L = \Phi(a^H)$ and $a^H \sim \pi^H(a^H|s^H)$. Subsequently, given a threshold, filtering strategies can be readily applied to select high-level states and grasps such that the desired manipulation performances are satisfied.

Estimate the true action consequences

For many real-world problems, the true action consequences are unknown. Thus, it is often necessary to learn and estimate them, for instance, using a neural network. Denote the i-th estimated manipulation performance of a grasp as $\tilde{y}_{i,\boldsymbol{\vartheta}}^{\pi^L}(\Phi(a^H),\cdot)$, whose weights $\boldsymbol{\vartheta}$ are optimised via supervised learning in this chapter. For convenience, the model is named *general action-consequence prediction* (GAP) model. For the hook disentangling tasks in this chapter, the true action consequences, i.e., downstream manipulation performances, are associated with the low-level manipulation policy. Therefore, the manipulation policy (see next subsection) is evaluated for every pre-recorded scene and grasp pose to collect the labels for training. After training, $\tilde{y}_{i,\boldsymbol{\vartheta}}^{\pi^L}$ is used to infer manipulation-affordable grasp actions.

Network: For the experiments of this chapter, the GAP model $\tilde{y}_{i,\boldsymbol{\vartheta}}^{\pi^L}$ is represented by a three-layer (512-512-256) MLP, which outputs a three-dimensional vector corresponding to the three manipulation performance measurements. The input of the network is a concatenated vector of the (3D) coordinates of a grasp, the flattened rotation matrix (3×3) of the grasp and the point cloud latent feature (256-dim) provided by the Grasp-Net module. All layers are activated by ReLU, while the probability outputs are activated by the Sigmoid function and the averaged movement prediction head has no activation. The probability prediction heads are trained with the Binary Cross Entropy loss while the averaged movement prediction head is trained with the smooth L1 loss. Adam (Kingma and Ba, 2014) is used with a learning rate of 0.001 and a batch size of 1024. The network is updated for 10000 optimisation steps.

Affordable action inference

Action inference can be conducted if the true (or estimated) GAP model $\tilde{y}^{\pi^L}_{i,\boldsymbol{\vartheta}} i$ is given. According to Definition 2, a grasp action can afford the i-th manipulation performance iif. $y^{\pi^L+}_i \leq \tilde{y}^{\pi^L}_{i,\boldsymbol{\vartheta}}(\Phi(a^H), \cdot) + \epsilon_i$. One can then sample from the set of all affordable grasps, simply execute the manipulation policy with every affordable grasp, or filter away certain grasps that do not afford certain manipulation performances.

In addition, one can also use it to restrict the search space of grasp actions during planning within the set of affordable actions. However, this would require the access to a dynamic model. For example, one could plan over the affordable grasp space for a few grasp poses to clear up a number of entangled hooks. Although this is an interesting and valuable method, it is difficult to implement due to the difficulty of obtaining an accurate dynamic model, especially for such contact-rich manipulation tasks. Therefore, this chapter only uses the learnt affordances by filtering away unaffordable actions and executing every affordable one.

Lastly, it is worth noting that, by varying the *desired precision*, ϵ_i, the range of intents that can be satisfied can be changed (Khetarpal *et al.*, 2020a). For example, if $\epsilon_i = +\infty$ then any intent can be satisfied, which is however unrealistic. In this chapter, we use $\epsilon_i = 0$, meaning that any intent is considered satisfiable strictly according to the true manipulation policy performance.

Action filtering strategy

In this chapter, we consider and implement action filtering strategies with different manipulation intents. For example, ignoring all grasp actions that will result in a probability of successful separation that is smaller than different values. In fact, one can filter away grasp actions based on multiple action consequences. In other words, one specific action filtering strategy corresponds to a specific set of affordances (high-level state-action pairs). This module is named *manipulation affordance-based grasp filter* (MAGF) In particular, this chapter applies three strategies to filter the TAGs generated by the GraspNet model:

- I: Filter away grasp actions according to the estimated separation success probability: $y_1^{\pi^L+} \leq \tilde{y}_{1,\boldsymbol{\vartheta}}^{\pi^L+}(\Phi(a^H), \cdot)$.
- II: Filter away grasp actions according to the averaged movements of the non-grasped hooks: $y_3^{\pi^L+} \geq \tilde{y}_{3,\boldsymbol{\vartheta}}^{\pi^L+}(\Phi(a^H), \cdot)$.
- III: Combine strategies I and II.

Summary

To sum up, this subsection describes an implementation example for the concept of *general affordances* proposed in subsection 6.2.3 in the hook disentangling scenario. In simple words, the GAM framework will use the estimated manipulation performances and a set of pre-defined manipulation intents to describe the manipulation affordances for the task-agnostic grasp actions generated by the GraspNet model. Moreover, three filtering strategies are proposed to identify different affordance sets that improve the downstream manipulation performances in different ways. To give an overview, Algorithm 3 provides an example algorithm for applying the filtering strategy I to select grasp poses and execute manipulation. Experiments on these strategies in subsection 6.4.3 will confirm the effectiveness of the proposed concepts and framework.

Algorithm 3 General-affordance manipulation with strategy I

```
Input: TAG generation policy $\pi^H$, manipulation policy $\pi^L$
Input: Number of episodes $M$
Input: Manipulation intent $y_1^{\pi^L+}$, GAP model $\tilde{y}_{1,\boldsymbol{\vartheta}}^{\pi^L}(\Phi(a^H), \cdot)$
for $episode = 1$ to $M$ do
    Sample a set of TAG poses $A^H$ from $\pi^H$
    for $a^H \in A^H$ do
        if $y_1^{\pi^L+} \leq \tilde{y}_{1,\boldsymbol{\vartheta}}^{\pi^L+}(\Phi(a^H), \cdot)$ (strategy I)
            Set simulation to $s_0^L = \Phi(a^H)$
            for $t = 0$ to $T - 1$ do
                Sample and execute $a^L \sim \pi^L$
        end if
    end for
end for
```

6.3.3 Reinforcement learning-based manipulation

This subsection is concerned with the learning of the low-level manipulation policy π^L, which controls the robot arm to separate the grasped hook. In particular, the episodic RL paradigm is used to formalise the task and the deep Q learning algorithm (see subsection 3.2.2) is employed to learn the policy. Note that, other motion generation methods are not excluded, and RL is selected due to the model-free benefit.

Overall, the manipulation process always starts with a hook being grasped by the gripper. For example, the two subfigures at step t_0 in Figure 6.1. Each episode corresponds to a pre-recorded grasp pose in a specific simulation scene. At each timestep, the RL agent selects an action to move the gripper in the Cartesian space, including translational and rotational movements at all axes. An episode is considered terminated with three conditions:

- The grasped object is dropped.
- The grasped object is moved out of the workspace.
- The maximum number of steps, T, is reached.

After the RL agent performs T actions ($T = 3$ in this chapter), a manually defined motion is executed to move the gripper straight up for 0.15 meters. If the grasped object is lifted up alone, the episode is considered successful. Otherwise, it is considered failed (e.g., when the object is dropped while being lifted up, when there are other hooks still entangled with it, etc.). The following will elaborate on the definition of the state, action, reward function and the training process.

State

For the manipulation policy, the observation is comprised of the states of the hooks and the gripper. For each hook, the state is comprised of the Cartesian coordinates of a number of keypoints (see Figure 6.4 left) and the quaternion of the centre of the hook, with respect to the world frame. For the gripper, the state consists of the Cartesian coordinates and the quaternion of the gripper tip frame, the finger width and the index of the hook being grasped.

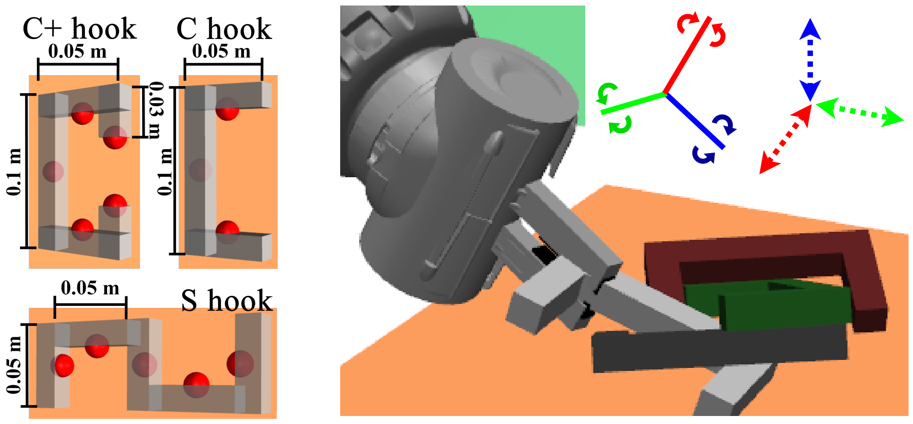


Figure 6.4: Left: hook size and its three keypoints (red). Right: action space visualisation (rotations about the gripper tip axes and translations along the world frame axes).

Action

The action space consists of a total of 12 discrete actions, each of which corresponds to a small translational or rotational movement on the axes of the gripper tip frame. As shown by the right subfigure of Figure 6.4, the first 6 actions translate the gripper tip frame along the axes' positive and negative directions for a small distance d_a, while the last 6 actions rotate the gripper tip frame about the three axes in the clockwise and counter-clockwise directions for a small angle δ_a. In all the following experiments, $d_a = 0.05$ m and $\delta_a = 30$ degree. One recent previous work proposes an action space that moves the gripper to a set of locations that form a hemisphere around the gripper tip (Moosmann *et al.*, 2021a). Experiments are conducted to compare the different action space designs. To differentiate, our agent is named Cartesian movement (CM) policy and the baseline of (Moosmann *et al.*, 2021a) is named hemisphere movement (HM) policy.

Reward

At each timestep of the episode, the policy is given the weighted negative value of the averaged movement of all non-grasped hooks as a reward signal. The reason is that minimising the movements of non-grasped objects is practically the most important requirement on top of the separation success. Additionally, three terminal rewards are given for the three termination conditions. When the grasped object is dropped or moved out of the workspace, a negative reward is given as a punishment. When the maximum number of timestep is reached, successful lifting up the object results in a positive reward and a failed case results in a zero reward. Formally, the reward function at timestep t can be written as:

$$r_t = \kappa \times d_t^{obj} + r_t^{step} \tag{6.3}$$

where, κ is the coefficient for the averaged non-grasped object movement d_t^{obj} and:

$$r_t^{step} = \begin{cases} 0, & \text{non-terminal or fail to lift up object} \\ a, & \text{successfully lift up object at the end} \\ b, & \text{grasped object dropped or out of workspace} \end{cases}$$

In all the following experiments, $\kappa = -1.0$, $a = 10$ and $b = -10$.

Training and network

For the proposed action space and the baseline one (Moosmann *et al.*, 2021a), the manipulation policy is updated exactly once after each timestep for a total of 2000000 timesteps. The q networks for the baseline and the proposed method are both represented by three fully-connected layers of size 256. Each layer is activated by ReLU and no activation is applied to the output. Each agent uses a replay buffer of size 1000000, which is filled with $2e3$ warm-up transitions collected by taking random actions at the beginning of the training process. Adam (Kingma and Ba, 2014) is used to update the networks with a learning rate of 0.0001, a batch size of 128 and a discount factor of 0.99. The target networks are updated by copying the main networks exactly at every 1000 optimisation steps. For exploration, the linearly decaying ϵ-greedy strategy is used. The random action probability is decayed from 1.0 to 0.05 in 50000 timesteps linearly. For evaluation, the policy performs 30 episodes without exploration to calculate the averaged manipulation performances every 10000 timesteps.

6.4 Empirical Results

This section will introduce the details of the task and experiment design, then provide experiment results as well as discussions on the RL-based manipulation policy and the general affordance-aware manipulation framework. In particular, this section seeks to answer the following research questions:

(1) Given TAGs, how do the proposed CM policy, the HM policy and the straight-up lifting motion (SLM) baseline perform?

(2) Does the MAGF module improve manipulation results?

6.4.1 Experiment Design

Five variations of tasks are included in the experiments: two, three, four C shape hooks, three C+ shape hooks and three S shape hooks, shown by Figure 6.5 right. As mentioned, due to the generalisation ability of the GraspNet model, a number of task scene simulation states, point cloud latent features and TAGs are pre-recorded. The collection process starts with randomising the orientations of the hooks one by one to form a tangled state, and then dropping them together from 0.1 m above the centre of the workspace (the orange square in Figure 6.5 left). After the hooks stabilise on the table, the GraspNet model is run to obtain 10 TAGs with the highest scores. For each TAG, the robot is moved to the pose and then the fingers are closed. Then, a basic manipulation stability test is conducted by executing each of the 12 CM RL actions once. If the object remains grasped after the test, the simulation state of the robot grasping the object, the latent feature and the grasp itself are recorded as a data point. For each task variation, there are 250000 pre-recorded data points.

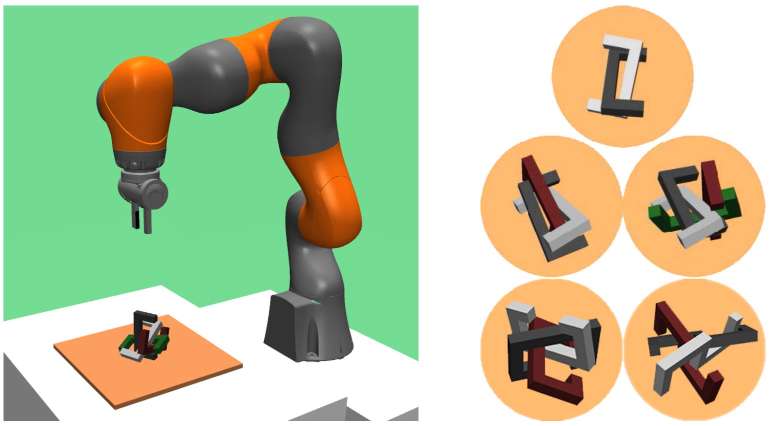


Figure 6.5: Left: manipulation workplace setting. Right: five task variations).

To answer the question (1), both RL policies are trained to separate the grasped hooks given all recorded TAGs. The training loops over all TAGs, each of which constitutes an RL episode. Training stops when the global timestep limit (2000000) is reached. The performance of each RL agent for each task is averaged over three random seeds. For evaluation, the SLM baseline is run over 50000 pre-recorded TAGs, while each RL agent is run over 45000 TAGs distributed to each random seed (15000 each). This experiment is primarily to discover the performances of the three manipulation policies with TAGs.

To answer the question (2), the GAP module $\tilde{y}^{\pi^L+}_{i,\boldsymbol{\vartheta}}$ is trained and used in the MAGF module to select TAGs before any manipulation. As mentioned, the action consequences are based on the policy that produces the actions. Therefore, the best CM policy among the three random seeds for each task is evaluated for all the pre-recorded TAGs to collect the ground-truth labels: $y^{\pi^L}_1$, $y^{\pi^L}_2$, and $y^{\pi^L}_3$. The GAP model is then trained to predict these action consequences given the recorded TAGs and latent features. After training, the GAP model is used to generate the estimated manipulation performances for TAGs in evaluation. Specifically, for each task with each filtering strategy (I, II or III), a total of 45000 TAGs are evaluated. The CM policy is executed with TAGs that satisfy the filtering strategy, and the performance is averaged over all satisfactory TAGs.

6.4.2 Performances with task-agnostic grasps

This subsection presents and discusses experiment results regarding question (1). Figure 6.6 displays the testing performances of both RL policies during policy training, as well as the performances of the SLM baseline.

First of all, Figure 6.6 shows that, even though the SLM baseline (green lines) achieves lower success rates in general, the RL policies (blue and orange lines) achieve much higher object dropping rates and non-grasped object movements, especially the one with HM actions (Moosmann *et al.*, 2021a). In addition, the RL policies may be able to separate an entangled hook with more grasping poses compared to the SLM baseline, but the gain of the success rates becomes less obvious as the task becomes more difficult. This result reveals the problem of conducting manipulation tasks with a TAG generator: most of the grasp poses will be unsuitable for manipulation.

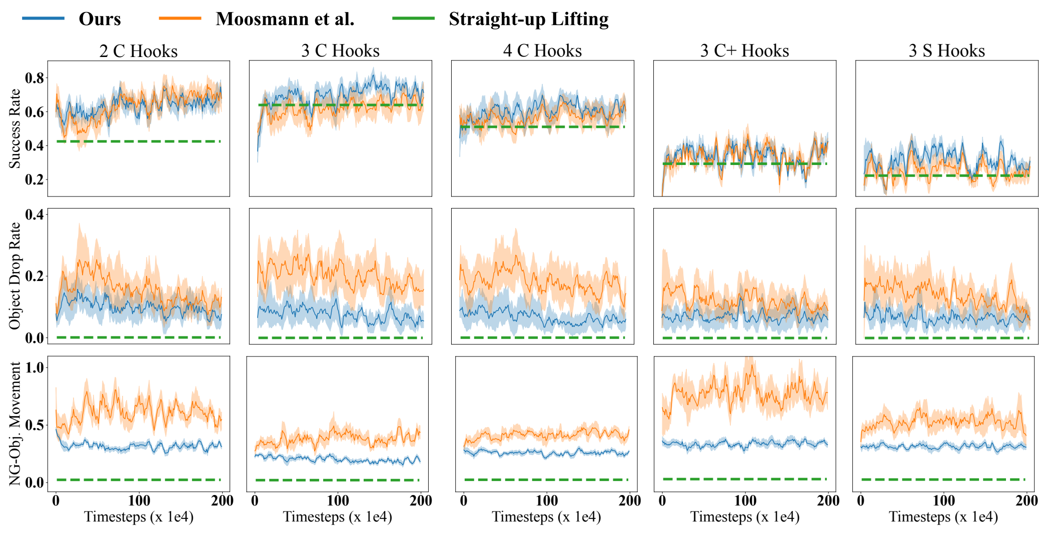


Figure 6.6: Means and standard deviations of the testing results of both RL agents (Blue and orange), and the performance of the straight-up lifting motion (green). **The performances reveal that a large proportion of the TAGs is not suitable for the downstream manipulation**. The situation exacerbates as the task becomes more difficult. The first row shows the success rates, the second row shows the rate of the object being dropped during manipulation, and the third row shows the average non-grasped object movements (in meters). From left to right, the columns correspond to the tasks with 2, 3, and 4 C hooks, 3 C+ hooks and 3 S hooks.

Secondly, comparing the two RL policies, the one with the proposed CM action space (blue lines) achieves smaller object-dropping rates and non-grasped object movements. This observation implies that the HM action space (Moosmann *et al.*, 2021a) is disadvantageous as it likely moves the gripper too drastically. However, the low separation success rates of both RL policies indicate that the design of the RL training processes could be over-simplified. For example, increasing the number of permitted actions per episode could bring an increase in the performances. On the other hand, this may also result in a higher object-dropping rate and non-grasped object movement, as a longer episode means more movements are to be made.

In general, the results in this subsection reveal the limitation of conducting manipulation tasks using only TAG poses. With TAGs, the SLM and both RL policies achieve unsatisfactory performances, with the CM policy slightly outperforming the other two in terms of success rates, and the SLM baseline achieving the least object-dropping rates and non-grasped object movements. In the next subsection, experiment results will demonstrate that this problem is solvable.

6.4.3 Performances with grasp filtering

This subsection presents and discusses the experimental results regarding question (2). Figure 6.7 displaces the performances achieved by the baselines without grasp filtering (GF) and the CM policy with different GF strategies. Notice that the performances of the SLM baseline and both RL policies without GF (the three blue bars of each comparative case) are consistent with the observations and discussions in the last subsection. They in general perform substantially worse than the CM policy with any GF strategy.

First of all, comparing the green, orange and pink bars with the blue bars, the performance gains of applying the MAGF module with the CM policy with any of the three strategies are clearly displayed. This empirically proves two points: 1) it is possible to learn to predict such manipulation performances based on a grasp pose and the scene latent feature as the threshold values match the actual manipulation results, and 2) the proposed MAGF module is effective in filtering away unsatisfactory TAGs by using individual (I and II) or combined (III) performance thresholds.

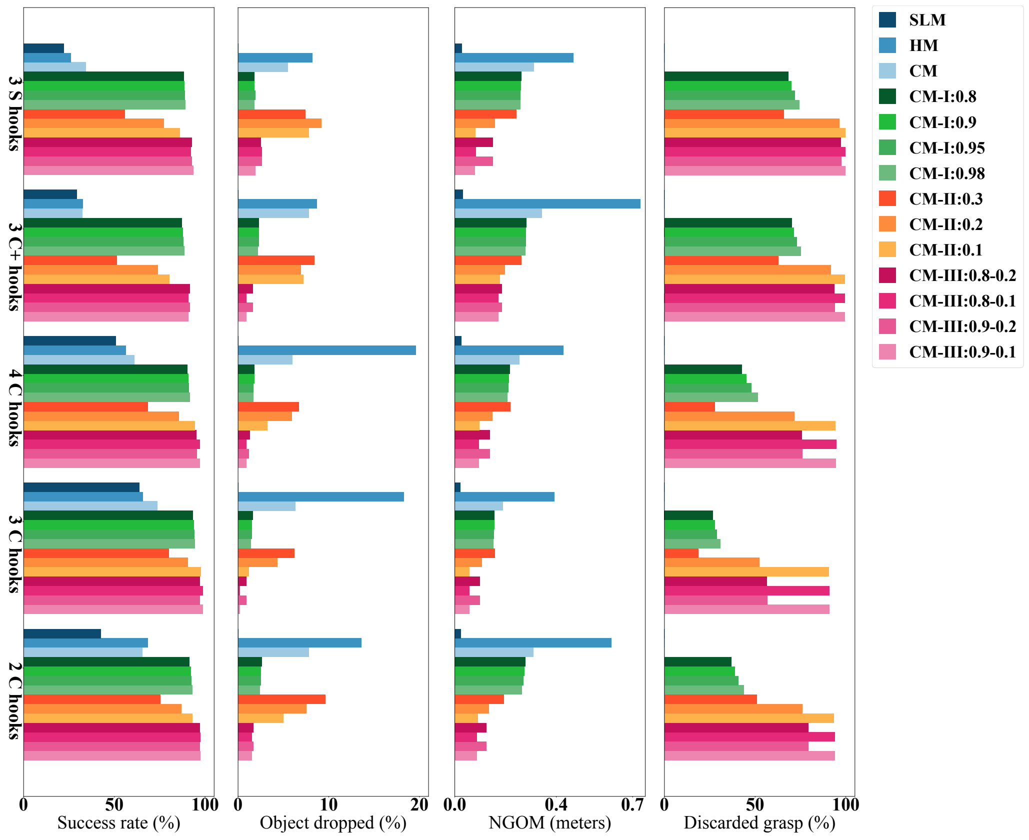


Figure 6.7: Comparison of different filtering strategies. SLM: straight-up lifting motion; HM: hemisphere movements; CM: Cartesian axes movements; NGOM: non-grasped objects movements. From left to right: success rates, objects dropping rates, non-grasped object movements, percentage of discarded grasps and figure legend. For each histogram, from top to bottom: the task with 3 S, 3 C+, and 2, 3, and 4 C hooks.

Secondly, the data reveals an interesting connection between the non-grasped object movements and separation success rate. The results of the first strategy (CM+I: green bars) show that those TAGs that are more likely to succeed are statistically associated with smaller object dropping rates and non-grasped object movements. Conversely, the results of the second strategy (CM+II: the orange bars) show that those TAGs that are more likely to cause smaller non-grasped object movements are statistically associated with higher success rates and low object dropping rates.

A possible cause of this phenomenon could be related to the contacts established during the manipulation processes. As the non-grasped object movements are the consequences of contacts and forces established between the grasped object and others, its increase indicates an increase in the number of contacts among objects. Therefore, it is plausible to believe that making less contact with other objects could lead to a smaller object-dropping rate and a higher success rate.

In general, the following conclusions can be drawn from the results:

- State-of-the-art TAG generators such as GraspNet tend to generate a large proportion of grasp poses that do not afford satisfactory manipulation performances.
- The proposed manipulation affordance prediction module based on the general affordance theory can be used to select TAGs according to different manipulation constraints and strategies.
- The GAM framework indeed substantially improves the manipulation performances of the difficult hook separation tasks.
- The results also reveal that reducing contact with other objects is important to achieve successful contact-rich manipulation tasks.

6.5 Summary

In sum, this chapter draws inspiration from the theory of affordances to interpret the connection between selecting a grasp pose and the downstream manipulation that happens to the grasped object. Specifically, section 6.2

recalls a recent theory of affordances based on the RL paradigm and extends it to include arbitrary action consequences that the robot is supposed to be concerned about. The proposed concept, *general affordances*, is then employed to develop a new manipulation system that can select task-agnostic grasp poses according to predicted manipulation performances, such as success rates, object dropping rates, etc. The proposed framework is named *general affordance-based manipulation* (GAM) framework. Its concept and effectiveness are then demonstrated by challenging experiments of hook disentangling tasks in simulation, showing substantial improvements over baselines without affordance-based grasp filtering.

However, a number of limitations need to be considered in the future. First of all, the quality of the simulation may be sufficient to serve the purpose of proof-of-concept, it is not realistic enough for deploying the simulation-validated methods directly onto real-world platforms. This is mainly due to the inaccurate modelling of the physic dynamics involving rich contacts and the difficulty of real-world object state estimation (the keypoints' poses of the simulated hooks). Much more effort is in demand to develop more realistic simulation software and improve the safety and sample efficiency of learning algorithms so they may learn in the real world. Secondly, the GAM framework assumes access to a TAG generator that gives reliably stable grasp poses, which is hardly true in practice. An interesting direction to consider may be using the information of a bad grasp poses for manipulation to fine-tune the TAG generator. Lastly, the concept of general affordance and its applications remain largely unexplored. Some mathematical relationships involving policy entropy, value function, Bellman equation, learning, etc. are to be further established and refined.

Chapter 7

Conclusion and future work

7.1 Conclusion

In this era of information, the field of robotics is becoming more and more involved with the field of artificial intelligence (AI). As recognised by many researchers, the current stage of robotics is in the middle of the big shift from classic model-based solutions to modern learning-based solutions. The research conducted in this thesis has also been carried by this big trend. What is primarily motivating to roboticists is the promising potential to leverage data to solve previously unsolvable tasks due to the difficulty of explicitly embedding human priors into the system design processes. This is also the main driver of the research in this thesis.

There are many unsolved challenges in robotics. This thesis was specifically concerned with improving the learning-based robotic arm grasping and manipulation processes. The problem of grasping and manipulation has been deemed a key challenge in empowering robots to step out of the comfort zone of the industry. Many of the real-world manipulation tasks exhibit vastly different characteristics compared to those that occur in structured and mostly certain environments like the industry. After reviewing the classic and modern approaches to robotic grasping and manipulation (chapters 2 and 3), the thesis identified three interesting and practically valuable tasks regarding learning-based long-horizon grasping and manipulation. Based on the proposed research objectives, the following summarises each research project. Section 7.2 will discuss the limitations of this thesis and some future important research directions.

Accelerate end-to-end RL for multi-step manipulation

In pursuit of objectives 1 to 3, chapter 4 focused on the difficulty of the end-to-end learning of long-horizon manipulation tasks that involved multiple subtasks under sparse reward signals, such as stacking several blocks, placing objects back in a drawer, etc.

In conclusion, we confirmed that end-to-end RL methods perform poorly in the face of long task horizons, delayed and sparse reward signals and subtask dependencies without acceleration techniques. To improve convergence rates and performances, we proposed $\mathbf{A}^2$, a new acceleration method for long-horizon end-to-end RL with sparse rewards that used abstract demonstrations and adaptive exploration. Abstract demonstrations were simply the correct sequences of a number of subtasks that led the learning agent towards the completion of the manipulation. Adaptive exploration helped to reduce random exploration for subtasks that were well-mastered and vice versa. This accelerated the learning agent to approach later subtasks by decreasing the probability of drifting away due to unnecessary exploration.

In order to facilitate experiments, a simulation software named "Pybullet Multigoal" (PMG) was developed. To examine the effectiveness of $\mathbf{A}^2$, it was implemented on the three most used RL algorithms: DQN, DDPG, and SAC, which were then experimented with a series of multi-step sparse reward tasks. The results confirmed that abstract demonstrations improved the convergence speeds and performances significantly in all tasks, while adaptive exploration improved performances in terms of reduced variances.

Improved HRL for multi-outcome multi-step manipulation

Secondly, to accomplish objectives 4 to 7, chapter 5 sought to develop a better hierarchical reinforcement learning-based manipulation system that can learn multiple task outcomes using the same set of skills. It was found that current HRL systems need to learn and maintain multiple policies for different skills and final tasks, and parallel training in HRL methods is non-stationary and inefficient due to a constantly exploring low-level policy.

In response, we developed a new HRL framework, named *universal option framework* (UOF), in which there is only one goal-augmented policy at each level. The low-level policy is able to act differently in the same system states given different subtasks, enabling knowledge-sharing between subtasks. Similarly, the high-level policy is able to reorganise the sequence of subtasks to achieve different task outcomes. The UOF is designed to learn different tasks at much lower data collection costs and with less memory consumption.

In addition, we applied parallel training for HRL to avoid repetitive or unnecessary data collection. The root cause of the non-stationarity of the parallel training process is then identified and the $\mathbf{A}^2$ method is proposed as a better solution to stabilise parallel training.

The simulation results of implementing UOF on a series of block-stacking tasks showed that it is possible to learn multiple outcomes and multiple subtasks with universal policies. Moreover, the stabilising effect of $\mathbf{A}^2$ also assented to the analysis of the root cause of the non-stationarity of parallel training processes, leading to significant learning acceleration and performance improvement.

General affordance-based grasping and manipulation

Last, in order to achieve objectives 8 to 10, chapter 6 focused on improving the grasp selection process for better downstream manipulation. To this end, we chose to use open-source TAG generators to obtain stable grasping poses. However, we hypothesised and empirically confirmed that not all stable grasp poses are suitable for downstream manipulation tasks. Even for the grasp poses that can support a manipulation task, they would induce different levels of manipulation difficulty and thus performances. The information about how well a grasp pose will support the downstream manipulation is required to improve manipulations that are sensitive to grasp selection.

Therefore, we drew inspiration from the concept of affordances and built upon the computational formulation of RL-based affordance to develop the theory of *general intent* and *general affordance*. In order to do so, a thorough review was conducted and summarised into a paper (Yang *et al.*, 2023). The general affordance was designed to capture the set of state-action pairs in which the actions are afforded to achieve some consequences in terms of any user-defined measurements. Based on these concepts, a number of ways to integrate them into action inference processes were also discussed.

Then, a manipulation framework, named *general affordance-based manipulation* (GAM), was developed based on the general affordance theory. GAM was designed to filter TAG grasps based on the estimated manipulation performances that these grasps may afford. It was then implemented with a point-cloud-based grasp generator and an RL-based manipulation policy. It was evaluated on a series of contact-rich hook disentangling tasks in simulation, in which the grasp poses were filtered based on the estimated manipulation success rate and the estimated average movement of surrounding objects that occurred during manipulation. Experiments confirmed the effectiveness of the GAM framework and thus demonstrated the validity of the concepts of general intent and general affordance. Interestingly, the experiments also revealed that the performances of contact-rich manipulation tasks are largely affected by how much contact is made between the grasped object and surrounding objects.

7.2 Limitations and future work

Finally, this section will discuss the limitations of the studies conducted in this thesis, followed by the proposals for future research topics.

The first limitation is rooted in the perception part of the proposed systems. In all experiments, the observations provided to the algorithms are the states of the system. Such as the coordinates and orientations of objects. Although the concepts and effectiveness of the approaches are demonstrated in simulations where such states are readily available, it remains unknown how they can perform in the real world where sensory observations are noisy, uncertain and redundant, and many system states are difficult or even impossible to be estimated, let alone accuracy requirements. In other words, the proposed approaches in this thesis are limited in terms of their generalisability across the types of sensory input. With this regard, a few future research questions could be invested:

- How to accelerate long-horizon multi-step, multi-outcome and sparse reward manipulation tasks when the provided inputs are not system states, but noisy, uncertain and redundant sensory readings?
- Training GRL policies require access to a goal generator, which can only be efficient in simulation given the amount of data needed for training RL agents. Then, what representation of goals is compact and effective for transferring the policy from simulation to the real world? Languages, images, point clouds or others?
- If only high-dimensional and noisy observations are available, is it possible to construct or learn a latent system state representation that enables faster learning and real-world deployment for robotic manipulation tasks?

The second limitation concerns the GAM framework. As presented, the core of the GAM framework is to consider what manipulation outcomes could be achieved during the generation and selection of the grasping poses. The experiments in chapter 6 indeed demonstrate the benefits of doing so. However, the grasp generator employed in the experiments is assumed to be fixed and unchangeable. As reported, only $\sim 10\%$ of the generated grasp poses can reach and maintain a firm grasp of the object, and a large proportion of these task-agnostic stable grasps is still not suitable for the downstream manipulation task. This is very far away from the basic requirements of a real-world manipulation system as the computation cost is too high.

However, the interplay between grasp planning and manipulation can be leveraged in the reverse direction as well. In order words, for systems that are specialised in a particular kind of manipulation task, the experienced manipulation data can be used to fine-tune the grasp generator, such that it tends to generate more grasps suitable for the manipulation task over time. Therefore, a practically valuable research direction is to study how manipulation experiences can be used as feedback to improve the grasp generator, such that the computation efficiency can meet the standard of real-world deployments.

Thirdly, the proposed systems in this thesis neglect a constraint that is critical to real-world systems: safety. On one side, the exploration methods employed in the training of the RL agents in this thesis have no consideration of safety issues whatsoever. This is obviously not realistic if training is to happen in the real world, even if it is only a fine-tuning process (Liu *et al.*, 2020). On another, the design of the action spaces for an RL agent should also consider safety constraints (Brunke *et al.*, 2022). A vivid example is shown by chapter 6 where RL policy with the hemisphere movements tends to cause much more influence to the surrounding objects. Therefore, a number of research topics can be pursued in this regard:

- For manipulation motion learning, especially for contact-rich tasks, how to define an action space that is not only efficient for learning but also safer and more flexible? Alternatively, is it possible to employ a set of safety-aware motion skills? Does imitation learning help?
- For exploration, safety constraints are always case-sensitive. A particular constraint that works well in a particular manipulation task may require total redesign for another task. What are the common principles of safety constraint design for contact-rich manipulation tasks? Are they generalisable and easy to implement across various tasks?
- Humans possess the prediction ability to anticipate potential dangers or hazards. Could such abilities be interpreted as a form of general affordance prediction? What are good representations? Is it always necessary to learn an explicit dynamics model for evaluating safety as proposed by recent research?

Last but not least, another major limitation is that the experiments in this thesis are all based on computer simulation software. Robotic simulation, on the one hand, is an important substitution for real-world platforms mainly because it is cheaper, safer and easier. On the other hand, however, the methods, especially data-driven methods, developed and validated in the simulation environment often suffer from the difficulty of real-world deployment. In particular, the reasons for the proposed projects failing in real-world tests are most likely to be 1) the inaccurate or even impossible estimation of the states of objects and 2) the mismatch of physic dynamics between simulation and the real world due to inaccurate and oversimplified physics models. In the future, new research is needed to deploy and test these methods on real-world platforms. In particular, the following directions may be considered:

- Examine the proposed methods with different observation modalities such as images and point clouds that are easier to obtain in the real world. Research in fast and realistic rendering (Schwarz and Behnke, 2020), representation learning and pre-training (De Bruin *et al.*, 2018; Lesort *et al.*, 2018), and image-based DRL (Lazaridis *et al.*, 2020) would be helpful as references.
- Transferring simulation-based policies into the real world is also a promising direction. Incorporating sim-to-real techniques such as domain randomisation (Alghonaim and Johns, 2021) into the proposed methods could be fruitful. Fine-tuning the simulation-based policy in the real world may also be carried out if the mismatch between simulation and the real world is not too large.
- Developing better robotic simulation software is also necessary. One of the big assumptions of robotic simulation is the rigid body assumption which has a great impact on the modelling of object contacts. More advanced physics simulators that try to discard this assumption have been developed and demonstrated to support more accurate and precise contact simulation (Collins *et al.*, 2021), but they are not yet ready for robot learning experiments at scale. Incorporating topics such as continuum physics simulation, deformable objects, and differentiable simulations could be valuable for developing new robotic simulators (Hu *et al.*, 2018, 2019a,b; Huang *et al.*, 2021).

Bibliography

Abbeel, P., Coates, A. and Ng, A. Y. 2010. Autonomous helicopter aerobatics through apprenticeship learning. *The International Journal of Robotics Research* 29(13), pp. 1608–1639.

Abbeel, P. and Ng, A. Y. 2004. Apprenticeship learning via inverse reinforcement learning. In: *Proceedings of the twenty-first international conference on Machine learning*. p. 1.

Abdo, N., Kretzschmar, H., Spinello, L. and Stachniss, C. 2013. Learning manipulation actions from a few demonstrations. In: *2013 IEEE International Conference on Robotics and Automation*. IEEE, pp. 1268–1275.

Agostinelli, F., McAleer, S., Shmakov, A. and Baldi, P. 2019. Solving the rubik's cube with deep reinforcement learning and search. *Nature Machine Intelligence* 1(8), pp. 356–363.

Agrawal, P. 2022. The task specification problem. In: *Conference on Robot Learning*. PMLR, pp. 1745–1751.

Akkaya, I., Andrychowicz, M., Chociej, M., Litwin, M., McGrew, B., Petron, A., Paino, A., Plappert, M., Powell, G., Ribas, R. *et al.* 2019. Solving rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113* .

Aleotti, J. and Caselli, S. 2010. Interactive teaching of task-oriented robot grasps. *Robotics and Autonomous Systems* 58(5), pp. 539–550.

Alghonaim, R. and Johns, E. 2021. Benchmarking domain randomisation for visual sim-to-real transfer. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 12802–12808.

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

Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Pieter Abbeel, O. and Zaremba, W. 2017. Hindsight experience replay. *Advances in neural information processing systems* 30.

Ardón, P., Pairet, È., Lohan, K. S., Ramamoorthy, S. and Petrick, R. 2021. Building affordance relations for robotic agents-a review. In: *2021 International Joint Conference on Artificial Intelligence (IJCAI)*. Springer.

Arora, S. and Doshi, P. 2021. A survey of inverse reinforcement learning: Challenges, methods and progress. *Artificial Intelligence* 297, p. 103500.

Arriola-Rios, V. E., Guler, P., Ficuciello, F., Kragic, D., Siciliano, B. and Wyatt, J. L. 2020. Modeling of deformable objects for robotic manipulation: A tutorial and review. *Frontiers in Robotics and AI* p. 82.

Aubret, A., Matignon, L. and Hassas, S. 2019. A survey on intrinsic motivation in reinforcement learning. *arXiv preprint arXiv:1908.06976* .

Bacon, P.-L., Harb, J. and Precup, D. 2017. The option-critic architecture. In: *Proceedings of the AAAI Conference on Artificial Intelligence.*

Barreto, A., Borsa, D., Hou, S., Comanici, G., Aygün, E., Hamel, P., Toyama, D., Mourad, S., Silver, D., Precup, D. *et al.* 2019. The option keyboard: Combining skills in reinforcement learning. *Advances in Neural Information Processing Systems* 32.

Barry, J., Kaelbling, L. P. and Lozano-Pérez, T. 2013. A hierarchical approach to manipulation with diverse actions. In: *2013 IEEE International Conference on Robotics and Automation.* IEEE, pp. 1799–1806.

Bicchi, A. 1995. On the closure properties of robotic grasping. *The International Journal of Robotics Research* 14(4), pp. 319–334.

Bicchi, A. and Kumar, V. 2000. Robotic grasping and contact: A review. In: *Proceedings 2000 ICRA. Millennium conference. IEEE international conference on robotics and automation. Symposia proceedings (Cat. No. 00CH37065).* IEEE, vol. 1, pp. 348–353.

Billard, A. and Kragic, D. 2019. Trends and challenges in robot manipulation. *Science* 364(6446), p. eaat8414.

Borrego, J., Figueiredo, R., Dehban, A., Moreno, P., Bernardino, A. and Santos-Victor, J. 2018. A generic visual perception domain randomisation framework for gazebo. In: *2018 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC).* IEEE, pp. 237–242.

Breyer, M., Furrer, F., Novkovic, T., Siegwart, R. and Nieto, J. 2019. Comparing task simplifications to learn closed-loop object picking using deep reinforcement learning. *IEEE Robotics and Automation Letters* 4(2), pp. 1549–1556.

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

Brunke, L., Greeff, M., Hall, A. W., Yuan, Z., Zhou, S., Panerati, J. and Schoellig, A. P. 2022. Safe learning in robotics: From learning-based control to safe reinforcement learning. *Annual Review of Control, Robotics, and Autonomous Systems* 5, pp. 411–444.

Burda, Y., Edwards, H., Pathak, D., Storkey, A., Darrell, T. and Efros, A. A. 2019. Large-scale study of curiosity-driven learning. In: *International Conference on Learning Representations*. Available at: https://openreview.net/forum?id=rJNwDjAqYX.

Cai, J., Cheng, H., Zhang, Z. and Su, J. 2019. Metagrasp: Data efficient grasping by affordance interpreter network. In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 4960–4966.

Caldera, S., Rassau, A. and Chai, D. 2018. Review of deep learning methods in robotic grasp detection. *Multimodal Technologies and Interaction* 2(3), p. 57.

Chebotar, Y., Handa, A., Makoviychuk, V., Macklin, M., Issac, J., Ratliff, N. and Fox, D. 2019. Closing the sim-to-real loop: Adapting simulation randomization with real world experience. In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 8973–8979.

Chen, X., Zhou, Z., Wang, Z., Wang, C., Wu, Y. and Ross, K. 2020. Bail:

Best-action imitation learning for batch deep reinforcement learning. *Advances in Neural Information Processing Systems* 33, pp. 18353–18363.

Chevalier-Boisvert, M., Willems, L. and Pal, S. 2018. *Minimalistic gridworld environment for openai gym*, [Online]. https://github.com/maximecb/gym-minigrid.

Chiang, H.-T. L., Hsu, J., Fiser, M., Tapia, L. and Faust, A. 2019. Rl-rrt: Kinodynamic motion planning via learning reachability estimators from rl policies. *IEEE Robotics and Automation Letters* 4(4), pp. 4298–4305.

Chin, R. T. and Dyer, C. R. 1986. Model-based recognition in robot vision. *ACM Computing Surveys (CSUR)* 18(1), pp. 67–108.

Chitnis, R., Tulsiani, S., Gupta, S. and Gupta, A. 2020. Efficient bimanual manipulation using learned task schemas. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 1149–1155.

Chitta, S., Sturm, J., Piccoli, M. and Burgard, W. 2011. Tactile sensing for mobile manipulation. *IEEE Transactions on Robotics* 27(3), pp. 558–568.

Chitta, S., Sucan, I. and Cousins, S. 2012. Moveit! *IEEE Robotics & Automation Magazine* 19(1), pp. 18–19.

Cho, D., Kim, J. and Kim, H. J. 2022. Unsupervised reinforcement learning for transferable manipulation skill discovery. *IEEE Robotics and Automation Letters* .

Chu, F.-J., Xu, R., Seguin, L. and Vela, P. A. 2019. Toward affordance detection and ranking on novel objects for real-world robotic manipulation. *IEEE Robotics and Automation Letters* 4(4), pp. 4070–4077.

Collins, J., Chand, S., Vanderkop, A. and Howard, D. 2021. A review of physics simulators for robotic applications. *IEEE Access* 9, pp. 51416–51431.

Corke, P. I. and Khatib, O. 2011. *Robotics, vision and control: fundamental algorithms in MATLAB*, vol. 73. Springer.

Coumans, E. and Bai, Y. 2016. Pybullet, a python module for physics simulation for games, robotics and machine learning .

Dalal, M., Pathak, D. and Salakhutdinov, R. R. 2021. Accelerating robotic reinforcement learning via parameterized action primitives. *Advances in Neural Information Processing Systems* 34, pp. 21847–21859.

Dang, H. and Allen, P. K. 2012. Semantic grasping: Planning robotic grasps functionally suitable for an object manipulation task. In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, pp. 1311–1317.

De Bruin, T., Kober, J., Tuyls, K. and Babuška, R. 2018. Integrating state representation learning into deep reinforcement learning. *IEEE Robotics and Automation Letters* 3(3), pp. 1394–1401.

Degris, T., White, M. and Sutton, R. S. 2012. Off-policy actor-critic. In: *ICML*. p. 179–186.

Detry, R., Papon, J. and Matthies, L. 2017. Task-oriented grasping with semantic and geometric scene understanding. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 3266–3273.

Dilokthanakul, N., Kaplanis, C., Pawlowski, N. and Shanahan, M. 2019. Feature control as intrinsic motivation for hierarchical reinforcement learning. *IEEE transactions on neural networks and learning systems* 30(11), pp. 3409–3418.

Do, T.-T., Nguyen, A. and Reid, I. 2018. Affordancenet: An end-to-end deep learning approach for object affordance detection. In: *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, pp. 5882–5889.

Du, G., Wang, K., Lian, S. and Zhao, K. 2021. Vision-based robotic grasping from object localization, object pose estimation to grasp estimation for parallel grippers: a review. *Artificial Intelligence Review* 54(3), pp. 1677–1734.

Engel, Y., Mannor, S. and Meir, R. 2005. Reinforcement learning with gaussian processes. In: *ICML*. pp. 201–208.

Eschmann, J. 2021. Reward function design in reinforcement learning. *Reinforcement Learning Algorithms: Analysis and Applications* pp. 25–33.

Exarchos, I., Jiang, Y., Yu, W. and Liu, C. K. 2021. Policy transfer via kinematic domain randomization and adaptation. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 45–51.

Eysenbach, B., Geng, X., Levine, S. and Salakhutdinov, R. R. 2020. Rewriting history with inverse rl: Hindsight inference for policy improvement. *Advances in neural information processing systems* 33, pp. 14783–14795.

Fang, B., Jia, S., Guo, D., Xu, M., Wen, S. and Sun, F. 2019a. Survey of imitation learning for robotic manipulation. *International Journal of Intelligent Robotics and Applications* 3(4), pp. 362–369.

Fang, H.-S., Wang, C., Gou, M. and Lu, C. 2020a. Graspnet-1billion: A large-scale benchmark for general object grasping. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. pp. 11444–11453.

Fang, K., Zhu, Y., Garg, A., Kurenkov, A., Mehta, V., Fei-Fei, L. and Savarese, S. 2020b. Learning task-oriented grasping for tool manipulation from simulated self-supervision. *The International Journal of Robotics Research* 39(2-3), pp. 202–216.

Fang, M., Zhou, T., Du, Y., Han, L. and Zhang, Z. 2019b. Curriculum-guided hindsight experience replay. *Advances in neural information processing systems* 32.

Fujimoto, S., Hoof, H. and Meger, D. 2018. Addressing function approximation error in actor-critic methods. In: *International conference on machine learning*. PMLR, pp. 1587–1596.

Garrett, C. R., Chitnis, R., Holladay, R., Kim, B., Silver, T., Kaelbling, L. P. and Lozano-Pérez, T. 2021. Integrated task and motion planning. *Annual review of control, robotics, and autonomous systems* 4, pp. 265–293.

Gibson, E. J. and Collins, W. 1982. The concept of affordances in development: The renascence of functionalism. In: *The concept of development: The Minnesota symposia on child psychology*. *Vol.* vol. 15, pp. 55–81.

Goodfellow, I., Bengio, Y. and Courville, A. 2016. *Deep learning*. MIT press.

Grafton, S. T. and Hamilton, A. F. d. C. 2007. Evidence for a distributed hierarchy of action representation in the brain. *Human movement science* 26(4), pp. 590–616.

Gu, S., Holly, E., Lillicrap, T. and Levine, S. 2017. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In: *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE, pp. 3389–3396.

Gu, S., Lillicrap, T., Sutskever, I. and Levine, S. 2016. Continuous deep q-learning with model-based acceleration. In: *International conference on machine learning*. PMLR, pp. 2829–2838.

Guenter, F., Hersch, M., Calinon, S. and Billard, A. 2007. Reinforcement learning for imitating constrained reaching movements. *Advanced Robotics* 21(13), pp. 1521–1544.

Guo, X., Singh, S., Lee, H., Lewis, R. L. and Wang, X. 2014. Deep learning for real-time atari game play using offline monte-carlo tree search planning. *Advances in neural information processing systems* 27.

Gupta, A., Kumar, V., Lynch, C., Levine, S. and Hausman, K. 2019. Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning. *arXiv preprint arXiv:1910.11956* .

Haarnoja, T., Tang, H., Abbeel, P. and Levine, S. 2017. Reinforcement learning with deep energy-based policies. In: *International conference on machine learning*. PMLR, pp. 1352–1361.

Haarnoja, T., Zhou, A., Abbeel, P. and Levine, S. 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In: *ICML*. pp. 1861–1870.

Hakhamaneshi, K., Zhao, R., Zhan, A., Abbeel, P. and Laskin, M. 2022. Hierarchical few-shot imitation with skill transition models. In: *International Conference on Learning Representations*.

Hämäläinen, A., Arndt, K., Ghadirzadeh, A. and Kyrki, V. 2019. Affordance learning for end-to-end visuomotor robot control. In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 1781–1788.

Hamaya, M., Tanaka, K., Shibata, Y., Von Drigalski, F., Nakashima, C. and Ijiri, Y. 2021. Robotic learning from advisory and adversarial interactions using a soft wrist. *IEEE Robotics and Automation Letters* 6(2), pp. 3878–3885.

Harutyunyan, A., Dabney, W., Borsa, D., Heess, N., Munos, R. and Precup, D. 2019. The termination critic. *arXiv preprint arXiv:1902.09996* .

Hasselt, H. 2010. Double q-learning. *Advances in neural information processing systems* 23.

Hauser, K. and Latombe, J.-C. 2010. Multi-modal motion planning in non-expansive spaces. *The International Journal of Robotics Research* 29(7), pp. 897–915.

Holas, J. and Farkaš, I. 2020. Adaptive skill acquisition in hierarchical reinforcement learning. In: *International Conference on Artificial Neural Networks*. Springer, pp. 383–394.

Horak, P. C. and Trinkle, J. C. 2019. On the similarities and differences among contact models in robot simulation. *IEEE Robotics and Automation Letters* 4(2), pp. 493–499.

Houthooft, R., Chen, X., Duan, Y., Schulman, J., De Turck, F. and Abbeel, P. 2016. Vime: Variational information maximizing exploration. *Advances in neural information processing systems* 29.

Hu, Y., Fang, Y., Ge, Z., Qu, Z., Zhu, Y., Pradhana, A. and Jiang, C. 2018. A moving least squares material point method with displacement discontinuity and two-way rigid body coupling. *ACM Transactions on Graphics (TOG)* 37(4), pp. 1–14.

Hu, Y., Li, T.-M., Anderson, L., Ragan-Kelley, J. and Durand, F. 2019a. Taichi: a language for high-performance computation on spatially sparse data structures. *ACM Transactions on Graphics (TOG)* 38(6), pp. 1–16.

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

Huang, Z., Hu, Y., Du, T., Zhou, S., Su, H., Tenenbaum, J. B. and Gan, C. 2021. Plasticinelab: A soft-body manipulation benchmark with differentiable physics. In: *International Conference on Learning Representations*. Available at: https://openreview.net/forum?id=xCcdBRQEDW.

Hussein, A., Gaber, M. M., Elyan, E. and Jayne, C. 2017. Imitation learning: A survey of learning methods. *ACM Computing Surveys (CSUR)* 50(2), pp. 1–35.

Ibarz, J., Tan, J., Finn, C., Kalakrishnan, M., Pastor, P. and Levine, S. 2021. How to train your robot with deep reinforcement learning: lessons we have learned. *The International Journal of Robotics Research* 40(4-5), pp. 698–721.

Ivaldi, S., Peters, J., Padois, V. and Nori, F. 2014. Tools for simulating humanoid robot dynamics: a survey based on user feedback. In: *2014*

IEEE-RAS International Conference on Humanoid Robots. IEEE, pp. 842–849.

James, S., Freese, M. and Davison, A. J. 2019. Pyrep: Bringing v-rep to deep robot learning. *arXiv preprint arXiv:1906.11176* .

James, S. and Johns, E. 2016. 3d simulation for robot arm control with deep q-learning. *arXiv preprint arXiv:1609.03759* .

James, S., Ma, Z., Arrojo, D. R. and Davison, A. J. 2020. Rlbench: The robot learning benchmark & learning environment. *IEEE Robotics and Automation Letters* 5(2), pp. 3019–3026.

Janson, L., Schmerling, E., Clark, A. and Pavone, M. 2015. Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions. *The International journal of robotics research* 34(7), pp. 883–921.

Jauhri, S., Peters, J. and Chalvatzaki, G. 2022. Robot learning of mobile manipulation with reachability behavior priors. *IEEE Robotics and Automation Letters* 7(3), pp. 8399–8406.

Jiang, Y., Gu, S. S., Murphy, K. P. and Finn, C. 2019. Language as an abstraction for hierarchical deep reinforcement learning. *Advances in Neural Information Processing Systems* 32.

Jin, X.-B., Robert Jeremiah, R. J., Su, T.-L., Bai, Y.-T. and Kong, J.-L. 2021. The new trend of state estimation: From model-driven to hybrid-driven methods. *Sensors* 21(6), p. 2085.

Johns, E. 2021. Coarse-to-fine imitation learning: Robot manipulation from a single demonstration. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 4613–4619.

Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., Tunyasuvunakool, K., Bates, R., Žídek, A., Potapenko, A. *et al.* 2021. Highly accurate protein structure prediction with alphafold. *Nature* 596(7873), pp. 583–589.

Kalashnikov, D., Irpan, A., Pastor, P., Ibarz, J., Herzog, A., Jang, E., Quillen, D., Holly, E., Kalakrishnan, M., Vanhoucke, V. *et al.* 2018. Scalable deep reinforcement learning for vision-based robotic manipulation. In: *Conference on Robot Learning*. PMLR, pp. 651–673.

Karpas, E. and Magazzeni, D. 2020. Automated planning for robotics. *Annual Review of Control, Robotics, and Autonomous Systems* 3, pp. 417–439.

Khetarpal, K., Ahmed, Z., Comanici, G., Abel, D. and Precup, D. 2020a. What can i do here? a theory of affordances in reinforcement learning. In: *International Conference on Machine Learning*. PMLR, pp. 5243–5253.

Khetarpal, K., Ahmed, Z., Comanici, G. and Precup, D. 2021. Temporally abstract partial models. *Advances in Neural Information Processing Systems* 34, pp. 1979–1991.

Khetarpal, K., Klissarov, M., Chevalier-Boisvert, M., Bacon, P.-L. and Precup, D. 2020b. Options of interest: Temporal abstraction with interest functions. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. pp. 4444–4451.

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

Koenig, N. and Howard, A. 2004. Design and use paradigms for gazebo, an open-source multi-robot simulator. In: *2004 IEEE/RSJ international conference on intelligent robots and systems (IROS)(IEEE Cat. No. 04CH37566)*. IEEE, vol. 3, pp. 2149–2154.

Kokic, M., Kragic, D. and Bohg, J. 2020. Learning task-oriented grasping from human activity datasets. *IEEE Robotics and Automation Letters* 5(2), pp. 3352–3359.

Kokic, M., Stork, J. A., Haustein, J. A. and Kragic, D. 2017. Affordance detection for task-specific grasping using deep learning. In: *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*. IEEE, pp. 91–98.

Kolluru, R., Valavanis, K. P. and Hebert, T. M. 1998. Modeling, analysis, and performance evaluation of a robotic gripper system for limp material handling. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 28(3), pp. 480–486.

Konda, V. and Tsitsiklis, J. 1999. Actor-critic algorithms. *Advances in neural information processing systems* 12.

Kormushev, P., Calinon, S. and Caldwell, D. G. 2011. Imitation learning of positional and force skills demonstrated via kinesthetic teaching and haptic input. *Advanced Robotics* 25(5), pp. 581–603.

Krishnan, S., Fox, R., Stoica, I. and Goldberg, K. 2017. Ddco: Discovery of deep continuous options for robot learning from demonstrations. In: *Conference on robot learning*. PMLR, pp. 418–437.

Krizhevsky, A., Sutskever, I. and Hinton, G. E. 2017. Imagenet classification with deep convolutional neural networks. *Communications of the ACM* 60(6), pp. 84–90.

Kroemer, O., Niekum, S. and Konidaris, G. 2021. A review of robot learning for manipulation: Challenges, representations, and algorithms. *The Journal of Machine Learning Research* 22(1), pp. 1395–1476.

Kuka. 2022. *Lbr iiwa*, [Online]. https://www.kuka.com/en-gb/products/robotics-systems/industrial-robots/lbr-iiwa. Last accessed: 2022-12-05.

Kumra, S. and Kanan, C. 2017. Robotic grasp detection using deep convolutional neural networks. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 769–776.

Kwak, J. H., Lee, J., Whang, J. J. and Jo, S. 2022. Semantic grasping via a knowledge graph of robotic manipulation: A graph representation learning approach. *IEEE Robotics and Automation Letters* 7(4), pp. 9397–9404.

Ladosz, P., Weng, L., Kim, M. and Oh, H. 2022. Exploration in deep reinforcement learning: A survey. *Information Fusion* .

Latombe, J.-C. 2012. *Robot motion planning*, vol. 124. Springer Science & Business Media.

Lazaridis, A., Fachantidis, A. and Vlahavas, I. 2020. Deep reinforcement learning: A state-of-the-art walkthrough. *Journal of Artificial Intelligence Research* 69, pp. 1421–1471.

Leão, G., Costa, C. M., Sousa, A. and Veiga, G. 2020. Detecting and solving tube entanglement in bin picking operations. *Applied Sciences* 10(7), p. 2264.

Lechner, M., Amini, A., Rus, D. and Henzinger, T. A. 2023. Revisiting the adversarial robustness-accuracy tradeoff in robot learning. *IEEE Robotics and Automation Letters* 8(3), pp. 1595–1602.

Lechner, M., Hasani, R., Grosu, R., Rus, D. and Henzinger, T. A. 2021. Adversarial training is not ready for robot learning. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 4140–4147.

Lee, H.-R., Sreenivasan, R. A., Jeong, Y., Jang, J., Shim, D. and Lee, C.-G. 2022. Multi-policy grounding and ensemble policy learning for transfer learning with dynamics mismatch. In: *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22. International Joint Conferences on Artificial Intelligence Organization*.

Lenz, I., Lee, H. and Saxena, A. 2015. Deep learning for detecting robotic grasps. *The International Journal of Robotics Research* 34(4-5), pp. 705–724.

Lesort, T., Díaz-Rodríguez, N., Goudou, J.-F. and Filliat, D. 2018. State representation learning for control: An overview. *Neural Networks* 108, pp. 379–392.

Levine, S., Kumar, A., Tucker, G. and Fu, J. 2020. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643* .

Levine, S., Pastor, P., Krizhevsky, A., Ibarz, J. and Quillen, D. 2018. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International journal of robotics research* 37(4-5), pp. 421–436.

Levy, A., Konidaris, G., Platt, R. and Saenko, K. 2019. Learning multi-level hierarchies with hindsight. *ICLR* .

Li, A. C., Florensa, C., Clavera, I. and Abbeel, P. 2019. Sub-policy adaptation for hierarchical reinforcement learning. *arXiv preprint arXiv:1906.05862* .

Li, J. K., Lee, W. S. and Hsu, D. 2018. Push-net: Deep planar pushing for objects with unknown physical properties. In: *Robotics: Science and Systems*. vol. 14, pp. 1–9.

Li, K., Baron, N., Zhang, X. and Rojas, N. 2022. Efficientgrasp: A unified data-efficient learning to grasp method for multi-fingered robot hands. *IEEE Robotics and Automation Letters* 7(4), pp. 8619–8626.

Li, M., Zhang, T., Chen, Y. and Smola, A. J. 2014. Efficient mini-batch training for stochastic optimization. In: *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. pp. 661–670.

Li, Z., Hsu, P. and Sastry, S. 1989. Grasping and coordinated manipulation by a multifingered robot hand. *The International Journal of Robotics Research* 8(4), pp. 33–50.

Li, Z. and Sastry, S. S. 1988. Task-oriented optimal grasping by multifingered robot hands. *IEEE Journal on Robotics and Automation* 4(1), pp. 32–44.

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

Lin, L.-J. 1992. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning* 8(3), pp. 293–321.

Liu, A., Shi, G., Chung, S.-J., Anandkumar, A. and Yue, Y. 2020. Robust regression for safe exploration in control. In: *Learning for Dynamics and Control*. PMLR, pp. 608–619.

Liu, R., Nageotte, F., Zanne, P., de Mathelin, M. and Dresp-Langley, B. 2021. Deep reinforcement learning for the control of robotic manipulation: a focussed mini-review. *Robotics* 10(1), p. 22.

Liu, X., Xu, Z., Cao, L., Chen, X. and Kang, K. 2019. Deep reinforcement learning via past-success directed exploration. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. pp. 9979–9980.

Lopez, N. G., Nuin, Y. L. E., Moral, E. B., Juan, L. U. S., Rueda, A. S., Vilches, V. M. and Kojcev, R. 2019. gym-gazebo2, a toolkit for reinforcement learning using ros 2 and gazebo. *arXiv preprint arXiv:1903.06278* .

Lu, Q., Chenna, K., Sundaralingam, B. and Hermans, T. 2020. Planning multi-fingered grasps as probabilistic inference in a learned deep network. In: *Robotics Research*, Springer, pp. 455–472.

Luh, J. 1983. Conventional controller design for industrial robots—a tutorial. *IEEE Transactions on Systems, Man, and Cybernetics* pp. 298–316.

Lundström, G. 1974. Industrial robot grippers. *Industrial Robot: An International Journal* .

Lynch, K. M. and Park, F. C. 2017. *Modern robotics.* Cambridge University Press.

Mahler, J., Liang, J., Niyaz, S., Laskey, M., Doan, R., Liu, X., Ojea, J. A. and Goldberg, K. 2017. Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics. *arXiv preprint arXiv:1703.09312* .

Mandikal, P. and Grauman, K. 2021a. Learning dexterous grasping with object-centric visual affordances. In: *2021 IEEE International Conference on Robotics and Automation (ICRA).* IEEE, pp. 6169–6176.

Mandikal, P. and Grauman, K. 2021b. Learning dexterous grasping with object-centric visual affordances. In: *IEEE International Conference on Robotics and Automation (ICRA).*

Mason, M. T. 2018. Toward robotic manipulation. *Annual Review of Control, Robotics, and Autonomous Systems* 1(1).

Masoudi, N., Fadel, G. M., Pagano, C. C. and Elena, M. V. 2019. A review of affordances and affordance-based design to address usability. In: *Proceedings of the Design Society: International Conference on Engineering Design.* Cambridge University Press, pp. 1353–1362.

Matsumura, R., Domae, Y., Wan, W. and Harada, K. 2019. Learning based robotic bin-picking for potentially tangled objects. In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).* IEEE, pp. 7990–7997.

Mayer, V., Feng, Q., Deng, J., Shi, Y., Chen, Z. and Knoll, A. 2022. Ffhnet: Generating multi-fingered robotic grasps for unknown objects in real-time. In: *2022 International Conference on Robotics and Automation (ICRA).* IEEE, pp. 762–769.

McClelland, T. 2019. Representing our options: The perception of affordances for bodily and mental action. *Journal of Consciousness Studies* 26(3-4), pp. 155–180.

McDermott, D. 1991. Regression planning. *International Journal of Intelligent Systems* 6(4), pp. 357–416.

Minaee, S., Boykov, Y. Y., Porikli, F., Plaza, A. J., Kehtarnavaz, N. and Terzopoulos, D. 2021. Image segmentation using deep learning: A survey. *IEEE transactions on pattern analysis and machine intelligence* .

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

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G. *et al.* 2015. Human-level control through deep reinforcement learning. *nature* 518(7540), pp. 529–533.

Moerland, T. M., Broekens, J. and Jonker, C. M. 2020. Model-based reinforcement learning: A survey. *arXiv preprint arXiv:2006.16712* .

Moosmann, M., Kulig, M., Spenrath, F., Mönnig, M., Roggendorf, S., Petrovic, O., Bormann, R. and Huber, M. F. 2021a. Separating entangled workpieces in random bin picking using deep reinforcement learning. *Procedia CIRP* 104, pp. 881–886.

Moosmann, M., Spenrath, F., Kleeberger, K., Khalid, M. U., Mönnig, M., Rosport, J. and Bormann, R. 2020. Increasing the robustness of random bin picking by avoiding grasps of entangled workpieces. *Procedia CIRP* 93, pp. 1212–1217.

Moosmann, M., Spenrath, F., Mönnig, M., Khalid, M. U., Jaumann, M., Rosport, J. and Bormann, R. 2021b. Using deep neural networks to separate entangled workpieces in random bin picking. In: *Advances in Automotive Production Technology–Theory and Application*, Springer, pp. 238–246.

Motiv-Space-Systems. 2022. *Motiv space systems and blue origin announce modulink*, [Online]. https://motivss.com/motiv-space-systems-and-blue-origin-announce-modulink/. Last accessed: 2022-12-02.

Mousavian, A., Eppner, C. and Fox, D. 2019. 6-dof graspnet: Variational grasp generation for object manipulation. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. pp. 2901–2910.

Mu, T., Ling, Z., Xiang, F., Yang, D. C., Li, X., Tao, S., Huang, Z., Jia, Z. and Su, H. 2021. Maniskill: Generalizable manipulation skill benchmark with large-scale demonstrations. In: *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*. Available at: https://openreview.net/forum?id=zQIvkXHS_U5.

Murali, A., Liu, W., Marino, K., Chernova, S. and Gupta, A. 2021. Same object, different grasps: Data and semantic knowledge for task-oriented grasping. In: Kober, J., Ramos, F. and Tomlin, C., eds., *Proceedings of the 2020 Conference on Robot Learning*. PMLR, vol. 155 of *Proceedings of Machine Learning Research*, pp. 1540–1557.

Muratore, F., Ramos, F., Turk, G., Yu, W., Gienger, M. and Peters, J. 2022. Robot learning from randomized simulations: A review. *Frontiers in Robotics and AI* p. 31.

Nachum, O., Gu, S. S., Lee, H. and Levine, S. 2018. Data-efficient hierarchical reinforcement learning. *Advances in neural information processing systems* 31.

Nair, A., McGrew, B., Andrychowicz, M., Zaremba, W. and Abbeel, P. 2018. Overcoming exploration in reinforcement learning with demonstrations. In: *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, pp. 6292–6299.

Narvekar, S., Peng, B., Leonetti, M., Sinapov, J., Taylor, M. E. and Stone, P. 2022. Curriculum learning for reinforcement learning domains: A framework and survey. *J. Mach. Learn. Res.* 21.

Newbury, R., Gu, M., Chumbley, L., Mousavian, A., Eppner, C., Leitner, J., Bohg, J., Morales, A., Asfour, T., Kragic, D. *et al.* 2022. Deep learning approaches to grasp synthesis: A review. *arXiv preprint arXiv:2207.02556* .

Nguyen, V.-D. 1988. Constructing force-closure grasps. *The International Journal of Robotics Research* 7(3), pp. 3–16.

Nica, A., Khetarpal, K. and Precup, D. 2022. The paradox of choice: Using attention in hierarchical reinforcement learning. *Advances in Neural Information Processing Systems Workshop: All Things Attention* .

Nitzan, D. and Rosen, C. A. 1976. Programmable industrial automation. *IEEE Transactions on Computers* 25(12), pp. 1259–1270.

Noreen, I., Khan, A. and Habib, Z. 2016. Optimal path planning using rrt* based approaches: a survey and future directions. *International Journal of Advanced Computer Science and Applications* 7(11).

Oomichi, T., Okino, A., Higuchi, M., Maekawa, A. and Ohnishi, K. 1990. Development of working multifinger hand manipulator. In: *EEE International Workshop on Intelligent Robots and Systems, Towards a New Frontier of Applications*. IEEE, pp. 873–880.

Orban, G., Lanzilotto, M. and Bonini, L. 2021. From observed action identity to social affordances. *Trends in Cognitive Sciences* 25(6), pp. 493–505.

Orbik, J., Agostini, A. and Lee, D. 2021. Inverse reinforcement learning for dexterous hand manipulation. In: *2021 IEEE International Conference on Development and Learning (ICDL)*. IEEE, pp. 1–7.

Ortega, J., Shaker, N., Togelius, J. and Yannakakis, G. N. 2013. Imitating human playing styles in super mario bros. *Entertainment Computing* 4(2), pp. 93–104.

Ortenzi, V., Controzzi, M., Cini, F., Leitner, J., Bianchi, M., Roa, M. A. and Corke, P. 2019. Robotic manipulation and the role of the task in the metric of success. *Nature Machine Intelligence* 1(8), pp. 340–346.

Ortner, R., Gajane, P. and Auer, P. 2020. Variational regret bounds for reinforcement learning. In: *Uncertainty in Artificial Intelligence*. PMLR, pp. 81–90.

Paavai Anand, P. *et al.* 2021. A brief study of deep reinforcement learning with epsilon-greedy exploration. *International Journal Of Computing and Digital System* .

Pateria, S., Subagdja, B., Tan, A.-H. and Quek, C. 2021a. End-to-end hierarchical reinforcement learning with integrated subgoal discovery. *IEEE Transactions on Neural Networks and Learning Systems* .

Pateria, S., Subagdja, B., Tan, A.-h. and Quek, C. 2021b. Hierarchical reinforcement learning: A comprehensive survey. *ACM Computing Surveys (CSUR)* 54(5), pp. 1–35.

Peng, X. B., Andrychowicz, M., Zaremba, W. and Abbeel, P. 2018. Sim-to-real transfer of robotic control with dynamics randomization. In: *2018 IEEE international conference on robotics and automation (ICRA).* IEEE, pp. 3803–3810.

Peng, X. B., Berseth, G., Yin, K. and Van De Panne, M. 2017. Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning. *ACM Transactions on Graphics (TOG)* 36(4), pp. 1–13.

Peng, X. B., Chang, M., Zhang, G., Abbeel, P. and Levine, S. 2019. Mcp: Learning composable hierarchical control with multiplicative compositional policies. *Advances in Neural Information Processing Systems* 32.

Pertsch, K., Lee, Y. and Lim, J. 2021. Accelerating reinforcement learning with learned skill priors. In: *Conference on robot learning.* PMLR, pp. 188–204.

Petrík, V., Tapaswi, M., Laptev, I. and Sivic, J. 2021. Learning object manipulation skills via approximate state estimation from real videos. In: *Conference on Robot Learning.* PMLR, pp. 296–312.

Pezzulo, G. and Cisek, P. 2016. Navigating the affordance landscape: feedback control as a process model of behavior and cognition. *Trends in cognitive sciences* 20(6), pp. 414–424.

Pfrommer, S., Halm, M. and Posa, M. 2021. Contactnets: Learning discontinuous contact dynamics with smooth, implicit representations. In: *Conference on Robot Learning.* PMLR, pp. 2279–2291.

Plappert, M., Houthooft, R., Dhariwal, P., Sidor, S., Chen, R. Y., Chen, X., Asfour, T., Abbeel, P. and Andrychowicz, M. 2020. Parameter space noise for exploration. *ICLR* .

Ploskas, N. and Sahinidis, N. V. 2021. Review and comparison of algorithms and software for mixed-integer derivative-free optimization. *Journal of Global Optimization* pp. 1–30.

Prats, M., Sanz, P. J. and Del Pobil, A. P. 2007. Task-oriented grasping using hand preshapes and task frames. In: *Proceedings 2007 IEEE International Conference on Robotics and Automation*. IEEE, pp. 1794–1799.

Premebida, C., Ambrus, R. and Marton, Z.-C. 2018. Intelligent robotic perception systems. In: *Applications of Mobile Robots*, IntechOpen London, UK, pp. 111–127.

Puterman, M. L. 2014. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons.

Qi, C. R., Yi, L., Su, H. and Guibas, L. J. 2017. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in neural information processing systems* 30.

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

Rahman, N., Carbonari, L., D'Imperio, M., Canali, C., Caldwell, D. G. and Cannella, F. 2016. A dexterous gripper for in-hand manipulation. In: *2016 IEEE International Conference on Advanced Intelligent Mechatronics (AIM)*. IEEE, pp. 377–382.

Ramírez, J., Yu, W. and Perrusquía, A. 2022. Model-free reinforcement learning from expert demonstrations: a survey. *Artificial Intelligence Review* 55(4), pp. 3213–3241.

Rao, D., Sadeghi, F., Hasenclever, L., Wulfmeier, M., Zambelli, M., Vezzani, G., Tirumala, D., Aytar, Y., Merel, J., Heess, N. and raia hadsell. 2022. Learning transferable motor skills with hierarchical latent mixture policies. In: *International Conference on Learning Representations*.

Ravichandar, H., Polydoros, A. S., Chernova, S. and Billard, A. 2020. Recent advances in robot learning from demonstration. *Annual review of control, robotics, and autonomous systems* 3, pp. 297–330.

Rayamane, P., Ji, Z. and Packianather, M. 2022. Design and development of a robust vision-based tactile sensor. In: *2022 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*. IEEE, pp. 1417–1423.

Reuleaux, F. 2013. *The kinematics of machinery: outlines of a theory of machines*. Courier Corporation.

Roa, M. A. and Suárez, R. 2015. Grasp quality measures: review and performance. *Autonomous robots* 38(1), pp. 65–88.

Rohmer, E., Singh, S. P. and Freese, M. 2013. V-rep: A versatile and scalable robot simulation framework. In: *2013 IEEE/RSJ international conference on intelligent robots and systems*. IEEE, pp. 1321–1326.

Ross, S., Gordon, G. and Bagnell, D. 2011. A reduction of imitation learning and structured prediction to no-regret online learning. In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, pp. 627–635.

Sadat, A., Casas, S., Ren, M., Wu, X., Dhawan, P. and Urtasun, R. 2020. Perceive, predict, and plan: Safe motion planning through interpretable semantic representations. In: *European Conference on Computer Vision*. Springer, pp. 414–430.

Sahbani, A., El-Khoury, S. and Bidaud, P. 2012. An overview of 3d object grasp synthesis algorithms. *Robotics and Autonomous Systems* 60(3), pp. 326–336.

Salvato, E., Fenu, G., Medvet, E. and Pellegrino, F. A. 2021. Crossing the reality gap: A survey on sim-to-real transferability of robot controllers in reinforcement learning. *IEEE Access* 9, pp. 153171–153187.

Sasaki, F., Yohira, T. and Kawaguchi, A. 2018. Sample efficient imitation learning for continuous control. In: *International conference on learning representations*.

Schaul, T., Horgan, D., Gregor, K. and Silver, D. 2015. Universal value function approximators. In: *International conference on machine learning*. PMLR, pp. 1312–1320.

Schulman, J., Chen, X. and Abbeel, P. 2017a. Equivalence between policy gradients and soft q-learning. *arXiv preprint arXiv:1704.06440* .

Schulman, J., Levine, S., Abbeel, P., Jordan, M. and Moritz, P. 2015. Trust region policy optimization. In: *International conference on machine learning*. PMLR, pp. 1889–1897.

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

Schwarz, M. and Behnke, S. 2020. Stillleben: Realistic scene synthesis for deep learning in robotics. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 10502–10508.

Schwenkel, L., Guo, M. and Bürger, M. 2020. Optimizing sequences of probabilistic manipulation skills learned from demonstration. In: *Conference on Robot Learning*. PMLR, pp. 273–282.

Scieur, D. and Pedregosa, F. 2020. Universal average-case optimality of polyak momentum. In: *International conference on machine learning*. PMLR, pp. 8565–8572.

Shah, D., Toshev, A. T., Levine, S. and brian ichter. 2022. Value function spaces: Skill-centric state abstractions for long-horizon reasoning. In: *ICLR*. Available at: https://openreview.net/forum?id=vgqS1vkkCbE.

Shimoga, K. B. 1996. Robot grasp synthesis algorithms: A survey. *The International Journal of Robotics Research* 15(3), pp. 230–266.

Shridhar, M., Manuelli, L. and Fox, D. 2023. Perceiver-actor: A multi-task transformer for robotic manipulation. In: *Conference on Robot Learning*. PMLR, pp. 785–799.

Siciliano, B., Khatib, O. and Kröger, T. 2008. *Springer handbook of robotics*, vol. 200. Springer.

Siciliano, B. and Valavanis, K. P. 1998. *Control problems in robotics and automation*. Springer.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot,

M. *et al.* 2016. Mastering the game of go with deep neural networks and tree search. *nature* 529(7587), pp. 484–489.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D. and Riedmiller, M. 2014. Deterministic policy gradient algorithms. In: *International conference on machine learning*. PMLR, pp. 387–395.

Singh, B., Kumar, R. and Singh, V. P. 2021. Reinforcement learning in robotic applications: a comprehensive survey. *Artificial Intelligence Review* pp. 1–46.

Smith, L., Kew, J. C., Peng, X. B., Ha, S., Tan, J. and Levine, S. 2022. Legged robots that keep on learning: Fine-tuning locomotion policies in the real world. In: *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 1593–1599.

Smith, M., Hoof, H. and Pineau, J. 2018. An inference-based policy gradient method for learning options. In: *International Conference on Machine Learning*. PMLR, pp. 4703–4712.

Sola, J. and Sevilla, J. 1997. Importance of input data normalization for the application of neural networks to complex industrial problems. *IEEE Transactions on nuclear science* 44(3), pp. 1464–1468.

Staranowicz, A. and Mariottini, G. L. 2011. A survey and comparison of commercial and open-source robotic simulator software. In: *Proceedings of the 4th International Conference on PErvasive Technologies Related to Assistive Environments*. pp. 1–8.

Staroverov, A., Yudin, D. A., Belkin, I., Adeshkin, V., Solomentsev, Y. K. and Panov, A. I. 2020. Real-time object navigation with deep neural networks and hierarchical reinforcement learning. *IEEE Access* 8, pp. 195608–195621.

Stepputtis, S., Campbell, J., Phielipp, M., Lee, S., Baral, C. and Ben Amor, H. 2020. Language-conditioned imitation learning for robot manipulation tasks. *Advances in Neural Information Processing Systems* 33, pp. 13139–13150.

Strudel, R., Pashevich, A., Kalevatykh, I., Laptev, I., Sivic, J. and Schmid, C. 2020. Learning to combine primitive skills: A step towards versatile robotic manipulation. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 4637–4643.

Subramanian, K., Isbell Jr, C. L. and Thomaz, A. L. 2016. Exploration from demonstration for interactive reinforcement learning. In: *Proceedings of the 2016 international conference on autonomous agents & multiagent systems*. pp. 447–456.

Sun, M. and Gao, Y. 2021. Gater: Learning grasp-action-target embeddings and relations for task-specific grasping. *IEEE Robotics and Automation Letters* 7(1), pp. 618–625.

Sutton, R. S. 1984. *Temporal credit assignment in reinforcement learning.* University of Massachusetts Amherst.

Sutton, R. S. 1991. Dyna, an integrated architecture for learning, planning, and reacting. *ACM Sigart Bulletin* 2(4), pp. 160–163.

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

Sutton, R. S., McAllester, D., Singh, S. and Mansour, Y. 1999a. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems* 12.

Sutton, R. S., Modayil, J., Delp, M., Degris, T., Pilarski, P. M., White, A. and Precup, D. 2011. Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In: *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*. pp. 761–768.

Sutton, R. S., Precup, D. and Singh, S. 1998. Intra-option learning about temporally abstract actions. In: *ICML*. vol. 98, pp. 556–564.

Sutton, R. S., Precup, D. and Singh, S. 1999b. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence* 112(1-2), pp. 181–211.

Takahashi, T., Tsuboi, T., Kishida, T., Kawanami, Y., Shimizu, S., Iribe, M., Fukushima, T. and Fujita, M. 2008. Adaptive grasping by multi fingered hand with tactile sensor based on robust force and position control. In: *2008 IEEE International Conference on Robotics and Automation*. IEEE, pp. 264–271.

Tang, H., Houthooft, R., Foote, D., Stooke, A., Xi Chen, O., Duan, Y., Schulman, J., DeTurck, F. and Abbeel, P. 2017. # exploration: A study of count-based exploration for deep reinforcement learning. *Advances in neural information processing systems* 30.

Tesauro, G. *et al.* 1995. Temporal difference learning and td-gammon. *Communications of the ACM* 38(3), pp. 58–68.

Tessler, C., Givony, S., Zahavy, T., Mankowitz, D. and Mannor, S. 2017. A deep hierarchical approach to lifelong learning in minecraft. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. vol. 31.

Tiong, T., Saad, I., Teo, K. T. K. and bin Lago, H. 2020. Deep reinforcement learning with robust deep deterministic policy gradient. In: *2020 2nd International Conference on Electrical, Control and Instrumentation Engineering (ICECIE)*. IEEE, pp. 1–5.

Todorov, E., Erez, T. and Tassa, Y. 2012. Mujoco: A physics engine for model-based control. In: *2012 IEEE/RSJ international conference on intelligent robots and systems*. IEEE, pp. 5026–5033.

Turpin, D., Wang, L., Heiden, E., Chen, Y.-C., Macklin, M., Tsogkas, S., Dickinson, S. and Garg, A. 2022. Grasp'd: Differentiable contact-rich grasp synthesis for multi-fingered hands. In: *European Conference on Computer Vision*. Springer, pp. 201–221.

Van Hasselt, H., Guez, A. and Silver, D. 2016. Deep reinforcement learning with double q-learning. In: *Proceedings of the AAAI conference on artificial intelligence*.

Vecerik, M., Hester, T., Scholz, J., Wang, F., Pietquin, O., Piot, B., Heess, N., Rothörl, T., Lampe, T. and Riedmiller, M. 2017. Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. *arXiv preprint arXiv:1707.08817* .

Wang, J., Chi, W., Li, C., Wang, C. and Meng, M. Q.-H. 2020. Neural rrt*: Learning-based optimal path planning. *IEEE Transactions on Automation Science and Engineering* 17(4), pp. 1748–1758.

Wang, X., Lee, K., Hakhamaneshi, K., Abbeel, P. and Laskin, M. 2022. Skill preferences: Learning to extract and execute robotic skills from human feedback. In: *Conference on Robot Learning*. PMLR, pp. 1259–1268.

Watkins, C. J. and Dayan, P. 1992. Q-learning. *Machine learning* 8(3), pp. 279–292.

Wei, E., Wicke, D. and Luke, S. 2018. Hierarchical approaches for reinforcement learning in parameterized action space. In: *Proceedings of the AAAI Conference on Artificial Intelligence*.

Wen, B., Lian, W., Bekris, K. and Schaal, S. 2022. Catgrasp: Learning category-level task-relevant grasping in clutter from simulation. In: *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 6401–6408.

West, D. B. *et al.* 2001. *Introduction to graph theory*, vol. 2. Prentice hall Upper Saddle River.

Williams, R. J. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning* 8(3), pp. 229–256.

Wu, B., Akinola, I. and Allen, P. K. 2019. Pixel-attentive policy gradient for multi-fingered grasping in cluttered scenes. In: *2019 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE, pp. 1789–1796.

Wu, H., Zhang, Z., Cheng, H., Yang, K., Liu, J. and Guo, Z. 2020. Learning affordance space in physical world for vision-based robotic object manipulation. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 4652–4658.

Xu, D., Mandlekar, A., Martín-Martín, R., Zhu, Y., Savarese, S. and Fei-Fei, L. 2021. Deep affordance foresight: Planning through what can be done in the future. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 6206–6213.

Yang, C., Lan, X., Zhang, H. and Zheng, N. 2019. Task-oriented grasping in object stacking scenes with crf-based semantic model. In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 6427–6434.

Yang, C., Yuan, K., Zhu, Q., Yu, W. and Li, Z. 2020. Multi-expert learning of adaptive legged locomotion. *Science Robotics* 5(49), p. eabb2174.

Yang, S., Zhang, W., Song, R., Cheng, J. and Li, Y. 2021. Learning multi-object dense descriptor for autonomous goal-conditioned grasping. *IEEE Robotics and Automation Letters* 6(2), pp. 4109–4116.

Yang, X., Ji, Z., Wu, J. and Lai, Y.-K. 2023. Recent advances of deep robotic affordance learning: a reinforcement learning perspective. *IEEE Transactions on Cognitive and Developmental Systems* .

Yu, W., Kumar, V. C., Turk, G. and Liu, C. K. 2019. Sim-to-real transfer for biped locomotion. In: *2019 ieee/rsj international conference on intelligent robots and systems (iros)*. IEEE, pp. 3503–3510.

Yuan, W., Dong, S. and Adelson, E. H. 2017. Gelsight: High-resolution robot tactile sensors for estimating geometry and force. *Sensors* 17(12), p. 2762.

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

Zeng, A., Song, S., Lee, J., Rodriguez, A. and Funkhouser, T. 2020. Tossingbot: Learning to throw arbitrary objects with residual physics. *IEEE Transactions on Robotics* 36(4), pp. 1307–1319.

Zeng, A., Song, S., Welker, S., Lee, J., Rodriguez, A. and Funkhouser, T. 2018. Learning synergies between pushing and grasping with self-supervised deep reinforcement learning. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 4238–4245.

Zeng, A., Song, S., Yu, K.-T., Donlon, E., Hogan, F. R., Bauza, M., Ma, D., Taylor, O., Liu, M., Romo, E. *et al.* 2022. Robotic pick-and-place of novel objects in clutter with multi-affordance grasping and cross-domain image matching. *The International Journal of Robotics Research* 41(7), pp. 690–705.

Zhang, F., Leitner, J., Ge, Z., Milford, M. and Corke, P. 2019. Adversarial discriminative sim-to-real transfer of visuo-motor policies. *The International Journal of Robotics Research* 38(10-11), pp. 1229–1245.

Zhang, S. and Whiteson, S. 2019. Dac: The double actor-critic architecture for learning options. *Advances in Neural Information Processing Systems* 32.

Zhang, X., Koyama, K., Domae, Y., Wan, W. and Harada, K. 2021. A topological solution of entanglement for complex-shaped parts in robotic bin-picking. In: *2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)*. IEEE, pp. 461–467.

Zhao, B., Zhang, H., Lan, X., Wang, H., Tian, Z. and Zheng, N. 2021. Regnet: Region-based grasp network for end-to-end grasp detection in point clouds. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 13474–13480.

Zhao, W., Queralta, J. P. and Westerlund, T. 2020. Sim-to-real transfer in deep reinforcement learning for robotics: a survey. In: *2020 IEEE symposium series on computational intelligence (SSCI)*. IEEE, pp. 737–744.

Zheng, K., Chen, X., Jenkins, O. C. and Wang, X. 2022. Vlmbench: A compositional benchmark for vision-and-language manipulation. *Advances in Neural Information Processing Systems* 35, pp. 665–678.

Zheng12, Z., Yuan, C., Lin12, Z. and Cheng12, Y. 2018. Self-adaptive double bootstrapped ddpg. In: *International Joint Conference on Artificial Intelligence*.

Zhou, M., Liu, Z., Sui, P., Li, Y. and Chung, Y. Y. 2020. Learning implicit credit assignment for cooperative multi-agent reinforcement learning. *Advances in neural information processing systems* 33, pp. 11853–11864.

Zhu, Y., Stone, P. and Zhu, Y. 2022. Bottom-up skill discovery from unsegmented demonstrations for long-horizon robot manipulation. *IEEE Robotics and Automation Letters* 7(2), pp. 4126–4133.

Zhu, Y., Wong, J., Mandlekar, A., Martín-Martín, R., Joshi, A., Nasiriany, S. and Zhu, Y. 2020. robosuite: A modular simulation framework and benchmark for robot learning. *arXiv preprint arXiv:2009.12293* .

Zou, F., Shen, L., Jie, Z., Zhang, W. and Liu, W. 2019a. A sufficient condition for convergences of adam and rmsprop. In: *Proceedings of the IEEE/CVF Conference on computer vision and pattern recognition*. pp. 11127–11135.

Zou, Z., Shi, Z., Guo, Y. and Ye, J. 2019b. Object detection in 20 years: A survey. *arXiv preprint arXiv:1905.05055* .