Enhancing Reinforcement Learning with a Context-based Approach

A thesis submitted in partial fulfilment
of the requirement for the degree of Doctor of Philosophy

Francisco Munguia-Galeano

October 2023

School of Engineering
Ysgol Peirianneg

In 1939, George Bernard Dantzig mistook two problems written on the blackboard as homework. He found the problems to be more challenging than normal. Despite their difficulty, he solved them. Those problems turned out to be open mathematical problems in statistics. This resulted in one of the shortest PhD theses ever.

Abstract

Reinforcement Learning (RL) has shown outstanding capabilities in solving complex computational problems. However, most RL algorithms lack an explicit method for learning from contextual information. In reality, humans rely on context to identify patterns and relations among elements in the environment and determine how to avoid making incorrect actions. Conversely, what may seem like obvious poor decisions from a human perspective could take hundreds of steps for an agent to learn how to avoid them. This thesis aims to investigate methods for incorporating contextual information into RL in order to enhance learning performance.

The research follows an incremental approach in which, first, contextual information is incorporated into RL in simulated environments, more concisely in games. The experiments show that all the algorithms which use contextual information significantly outperform the baseline algorithms by 77 % on average. Then, the concept is validated with a hybrid approach that comprises a robot in a Human-Robot Interaction (HRI) scenario dealing with rigid objects. The robot learns in simulation while executing actions in the real world. For this setup, based on contextual information, the proposed algorithm trains in a reduced amount of time (2.7 seconds). It reaches an 84% success rate in a grasp and release-related task while interacting with a human user, while the baseline algorithm with the highest success rate reached 68% after learning during a significantly longer period of time (91.8 seconds). Consequently, CQL suits the robot's learning requirements in observing the current scenario configuration and learning to solve it while dealing with dynamic changes provoked by the user.

Additionally, the thesis explores using an RL framework that uses contextual information to learn how to manipulate bags in the real world. A bag is a deformable object that presents challenges from grasping to planning, and RL has the potential to address this issue. The learning process is accomplished through a new RL algorithm introduced in this work called Π-learning, designed to find the best grasping points of the bag based on a set of compact state representations. The framework utilises a set of primitive actions and represents the task in five states. In the experiments, the framework reaches a 60% and 80% success rate after around three hours of training in the real world when starting the bagging task from folded and unfolded positions, respectively. Finally, the trained model is tested on two more bags of different sizes to evaluate its generalisation capacities.

Overall, this research seeks to contribute to the broader advancement of RL and robotics, aiming to enhance the development of intelligent, autonomous systems that can effectively operate in diverse and dynamic real-world settings. Besides that, this research seeks to explore new possibilities for automation, HRI, and the utilisation

of contextual information in RL.

Dedication

To my dearest wife,

You have been my main support and inspiration throughout this entire journey. I could not have accomplished this PhD without you by my side. I dedicate this thesis to you with all my heart.

To my dear baby,

As I write this dedication, you have yet to be born, but I already feel excited to meet you. May you always find joy and the strength to fight for your dreams. I dedicate this thesis to you with all my love.

To my dear mom, who passed away,

You were my first inspiration and my greatest supporter. Although you are no longer with us, your memory, life lessons, and love guide me. I would have achieved nothing without you. I dedicate this thesis to you with all the love in my heart.

To my loving dad,

You have been my constant source of guidance. Your support and encouragement have been essential in shaping me into the person I am today. I dedicate this thesis to you with all the love in my heart.

Declaration

This thesis is the result of my own independent work, except where otherwise stated, and the views expressed are my own. Other sources are acknowledged by explicit references. The thesis has not been edited by a third party beyond what is permitted by Cardiff University's Use of Third Party Editors by Research Degree Students Procedure.

Signed **Francisco Munguia Galeano**

Date **22/October/2023**

Statement 1

This thesis is being submitted in partial fulfilment of the requirements for the degree of PhD.

Signed **Francisco Munguia Galeano**

Date **22/October/2023**

Statement 2

This work has not been submitted in substance for any other degree or award at this or any other university or place of learning, nor is it being submitted concurrently for any other degree or award.

Signed **Francisco Munguia Galeano**

Date **22/October/2023**

Statement 3

I hereby give consent for my thesis, if accepted, to be available on the University's Open Access repository (or, where approved, to be available in the University's library and for inter-library loans), and for the title and summary to be made available to outside organisations, subject to the expiry of a University-approved bar on access if applicable.

Signed **Francisco Munguia Galeano**

Date **22/October/2023**

Acknowledgements

I want to thank the Consejo Nacional de Humanidades, Ciencias y Tecnologías (CONAHCyT) for funding and supporting part of my doctoral studies.

I am also grateful to Cardiff University for hosting my PhD studies. The academic environment provided me access to excellent resources, and my interactions with my colleagues were both stimulating and informative.

I would like to express my gratitude to my supervisors, Dr. Ze Ji and Dr. Juan David Hernández Vega, for their mentorship, support, knowledge, patience, and advice. I feel lucky to have had such dedicated researchers to guide me along the way.

I would like to thank my wife, family, and friends for their support and encouragement. Their belief in me was essential in completing my doctoral studies. I am truly blessed to have such wonderful people in my life.

Contents

List of Figures

List of Tables

List of Algorithms

List of Acronyms

A2C Advantage Actor-Critic

A3C Asynchronous Advantage Actor-Critic

CAD Computer-Aided Design

CKF Contextual Key Frame

CQL Contextual Q-learning

CMDP Contextual Markov Decision Process

CNNs Convolutional Neural Networks

DDDQN Double Dueling Deep Q-network

DDQN Double Deep Q-network

DDPG Deep Deterministic Policy Gradient

DP Dynamic Programming

DQNfD Deep Q-network from Demonstrations

DQN Deep Q-network

DRL Deep Reinforcement Learning

DuDQN Dueling Deep Q-network

GRASP Generative Action Selection Through Probability

IECR Iota Explicit Context Representation

IDDDQN Iota Double Dueling Deep Q-network

IDDQN Iota Double Deep Q-network

IDQN Iota Deep Q-network

IDuDQN Iota Dueling Deep Q-network

IRL Inverse Reinforcement Learning

HER Hindsight Experience Replay

HRI Human-Robot Interaction

LfD Learning from Demonstration

LLMs Large Language Models

LSTM Long Short-Term Memory

MC Monte Carlo

MDP Markov Decision Process

ML Machine Learning

MMCRL Multi-Modal Contextual Reinforcement Learning

MSE Mean Squared Error

PDF Probability Density Function

PPO Proximal Policy Optimisation

QL Q-learning

ReLU Rectified Linear Unit

RL Reinforcement Learning

RNNs Recurrent Neural Networks

RRT Rapid-exploration Random Trees

SAC Soft Actor-Critic

SARSA State-Action-Reward-State-Action

TD Temporal Difference

TD3 Twin Delayed Deep Deterministic Policy Gradient

TL Transfer Learning

List of Publications

The work introduced in this thesis is based on the following publications:

- **Francisco Munguia-Galeano**, Ah-wee Tan and Ze Ji. Deep Reinforcement Learning With Explicit Context Representation. IEEE Transactions on Neural Networks and Learning Systems, 2023 doi: 10.1109/TNNLS.2023.3325633.
- **Francisco Munguia-Galeano**, Satheeshkumar Veeramani, Juan David Hernández, Qingmeng Wen and Ze Ji. Affordance-based human-robot interaction with reinforcement learning. IEEE Access, 2023 doi: 10.1109/ACCESS.2023.3262450
- **Francisco Munguia-Galeano**, Jihong Zhu, Juan David Hernández and Ze Ji. Learning to bag with a simulation-free reinforcement learning framework for robots. IET Cyber-Systems and Robotics, 2024 doi: 10.1049/csy2.12113

Other publications:

- **Francisco Munguia-Galeano**, Lesli Ortega-Arroyo, Miguel Gabriel Villarreal-Cervantes and Ze Ji. Towards integrating 3D printing and automated assembly. Presented at: 2023 IEEE 19th International Conference on Automation Science and Engineering (CASE), Auckland, New Zealand, 2023.
- Yanzhang Tong, Qiyuan Zhang, **Francisco Munguia Galeano** and Ze Ji. Designing an AR Facial Expression System to Improve Trust in Human-Robots Collaboration. Presented at: 28th IEEE International Conference on Automation and Computing (ICAC2023), 2023.
- Prasad Rayamane, **Francisco Munguia-Galeano**, Seyed Amir Tafrishi and Ze Ji. Towards smooth human-robot handover with a vision-based tactile sensor. Presented at: Towards Autonomous Robotic Systems: 24th Annual Conference, TAROS 2023, Cambridge, UK, 2023.
- Prasad Rayamane, Peter Herbert, **Francisco Munguia-Galeano** and Ze Ji. Presented at: 2023 IEEE 29th International Conference on Mechatronics and Machine Vision in Practice (M2VIP), Queenstown, New Zealand, 2023.
- Yanzhang Tong, Qiyuan Zhang, **Francisco Munguia Galeano** and Ze Ji. Towards Flexibility and Efficiency for Smart Manufacturing: Advancing Human-Machine Perception and Collaboration. Presented at: The 25th IEEE International Conference on Industrial Technology (ICIT2024), 2024.

Chapter 1

Introduction

Reinforcement Learning (RL) has significantly evolved over the years, resulting in its implementation in various fields, such as gaming (Shao et al. 2019; Silver et al. 2016) and robotics (Brunke et al. 2022). RL algorithms can learn complex tasks from scratch by relying only on stochastic exploration and producing their own training dataset. The goal of RL is to train agents to learn optimal actions based on rewards and interact with the environment autonomously. These characteristics of RL make it suitable for several tasks where either the mathematical model of the system to control is not available or challenging to create. Despite the success of RL, several concerns still need addressing. For example, the learning process of agents involves massive exploration of the environment and repeating the same actions even when the agent has made the same mistake before. On the other hand, humans can identify patterns and utilise contextual information (e.g., semantics, rules, or affordances) to optimise their decision-making capabilities and avoid repeating mistakes (Munguia-Galeano et al. 2023a,b, 2024). Ideally, an RL agent should learn as fast as a human does (Voss et al. 2020). This thesis investigates methods involving the incorporation of contextual information into RL algorithms, with the goal of enhancing RL capabilities in exploration and learning, particularly in the context of robotics applications.

1.1 Motivation

The performance of an RL agent depends on the quality of the data that it has been training with (Raffin et al. 2022). In RL approaches, this data is produced by the own RL agent in a process known as the exploration-exploitation trade-off. The exploration stage is based on selecting actions stochastically, while the exploitation stage is based on taking the best action according to the agent's knowledge. This trade-off offers advantages such as not requiring a dataset since the own agent can generate it while training (Sutton and Barto 1998). Besides, RL is a human-like learning approach that allows the learning of complex behaviours (Y. Zhang et al. 2018). For example, RL has been implemented in robotics, in which, without knowing the mathematical model of the robot itself, robots can learn to walk from scratch (Lele et al. 2020). However, there also exist disadvantages, such as the complexity of the sequential interaction (which often produces low-quality training datasets) (Yarats et al. 2022), the reward function structure (Icarte et al. 2022), differences between simulation and the real world debase RL performance (W. Zhao et al. 2020) and generalisation (Cobbe et al. 2019).

The complexity involved in leveraging the exploration-exploitation trade-off of RL has a high impact on the quality of the data produced by the own agent. Consequently, this factor usually only depends on stochasticity, and the dataset grows exponentially with low-quality data, which dramatically decelerates the learning of the agent (Blondé and Kalousis 2019). In the literature, several approaches focus on using demonstrations to reduce the exploration space of RL agents. This approach is known as Learning from Demonstration (LfD) and is usually carried out using three methods: teleoperation, passive observation and kinesthetic demonstrations (Ravichandar et al. 2020). In teleoperated-based approaches, the user guides the robot with a joystick, a haptic device or a teach pendant (Pareek and Kesavadas 2019). Passive observation refers to robots that can learn from video streams containing human demonstrations of the task (Hwang et al. 2020). In kinesthetic

demonstrations, the user manually guides the robot by pulling or pushing the end effector (Stavridis et al. 2022). Despite the advantages of LfD, there are constraints for these approaches regarding the solution shown by the user because more optimal solutions than the demonstrated one are often discarded (Hester et al. 2018). Consequently, RL provides alternative solutions to this problem by encouraging the agent to explore further and find better solutions than the user's demonstrated one.

Additionally, data sampling quality significantly impacts the quality of the learning progress of the agent. Offline RL learning approaches have shown that high-quality data improves RL performance (Ren et al. 2018; T. Yu et al. 2021). The primary distinction between off-line and on-line RL algorithms lies in their training data. Online RL algorithms train with data generated in real-time during the training process, whereas offline RL algorithms use data collected beforehand. Typically, RL algorithms use an ϵ-greedy policy (Cruz et al. 2018b), which consists of selecting actions randomly and eventually bias preference to actions from the agent. Data sampling depends on the quality of the exploration process; it is essential to define when and how to carry out data sampling (M. Bellemare et al. 2016; Osband et al. 2019; Pan et al. 2018). The main problem with the state-of-the-art approaches is that undesirable data become trivial because the ϵ-greedy policy and the agent will keep selecting wrong actions for multiple episodes until the ϵ-greedy policy reaches a low probability and the agent learns to avoid them. Consequently, the training data set grows exponentially with low-quality data, and the agent decelerates its learning process. This is known as the sampling efficiency problem (Blondé and Kalousis 2019; Nair et al. 2018).

An additional problem with RL is that, because of its stochastic nature during the exploration-exploitation process, the actions taken by the agent rely only on the reward function, which can be sparse for complex tasks. Hence, the agent wastes time learning what not to do while increasing the complexity of the design of reward functions (Millan-Arias et al. 2021). This is because (i) RL relies on data that are

the product of the policy through random exploration, and the policy is trained through only the indirect information that is given as a reward (Naeem et al. 2020), and (ii) contextual information is often ignored, such that the agent must learn it from scratch. An approach to solve this problem is based on a concept that Gibson (1977) coined as affordances, which is contextual information that represents the relationships between actions and objects. Affordances give information about the effect of a given action, i.e., whether a particular task affords an action or not (C. Wang et al. 2013; Zeng 2019). Affordances enhance the performance of the ϵ-greedy policy (Sutton and Barto 1998). Applying affordances in robotics is essential because it encourages human-like generalisation capabilities (Ardón et al. 2019).

Despite the vast literature exploring several RL approaches in simulation, less attention has been paid to RL in the real world due to several problems (Dulac-Arnold et al. 2021), such as costly robot time, motion constraints, stochastic behaviours of objects surrounding the robot, and unnecessary robot wear due to the intrinsic exploration and learning process of RL. Although training RL in simulation is effective, transferring the trained policies to the real world is a challenge that must be taken seriously, often resulting in significant performance degradation due to the simulation-to-reality gap (Salvato et al. 2021; W. Zhao et al. 2020). Even though RL provides an alternative to hard-coded solutions, a significant gap exists regarding what is possible in simulation and learning in the real world (Hanna et al. 2021a). Among several problems, deformable objects present a challenge regarding large object configuration spaces, the difficulty of modelling the object's behaviour, and considerable change in the object's state resulting from manipulation attempts (Matas et al. 2018). Therefore, tackling complex problems such as learning to manipulate deformable objects in the real world may open the door to addressing further challenges in robotics and RL.

Once the issues of finding a proper exploration-exploitation trade-off and a good reward function are solved, it does not guarantee that the agent will perform well

when there are changes in the environment. This is known as generalisation and is a current challenge in RL (Hansen and X. Wang 2021; K. Wang et al. 2020). For example, setups involving Human-Robot Interaction (HRI) present highly dynamic and unpredictable behaviours (Modares et al. 2015). Although RL has the potential to deal with this problem (El-Shamouty et al. 2020), the performance of RL agents is seriously affected when they have to deal with unseen domains (Kostas et al. 2021). There has been research to address this problem. For example, Anoopkumar et al. (Sonar et al. 2021) proposed learning a representation that makes the optimal policy, to be built based on this representation, invariant across training domains. This approach aims to learn and exploit the causes of successful actions instead of the actions themselves. Another interesting approach is the one presented by Raileanu and Fergus (Raileanu and Fergus 2021), which shows that the need to capture level-specific features to estimate the value function can result in a policy that does not generalise well to new task instances. Consequently, both approaches demonstrate that learning abstract representations of the state and adding a contextual understanding of the task, rather than trying to understand pixels, enhances the generalisation capacities of RL. This is because when a neural network is fed only pixels, it takes time for the neural network to group and establish relationships between the numerical data and the actual task. In contrast, when this relationship is provided through demonstrations or state representations, the neural network saves time and does not need to learn such information from scratch. This enhances its learning performance.

To summarise, RL performance relies on good-quality data produced in a deficient exploration-exploitation trade-off that lacks context. Moreover, the difficulty of generalising and bringing the simulation to the real world is still a crucial challenge that needs to be addressed to improve RL. In particular, adding human-like capabilities, such as using context to enhance the exploration-exploitation trade-off process of RL, is essential. The hypothesis of this work is that injecting contextual

information (e.g., semantics, rules, or affordances) will result in improved sampling efficiency for RL agents and enhanced learning performance.

1.2 Research questions

The following research questions will be addressed in this thesis:

- How does contextual information impact the exploration-exploitation trade-off of RL agents?
- How to include context in the learning process?
- Can context-based RL agents be used in the real world with robots?
- Is context-based RL suitable for HRI?
- Can RL be used to learn how to manipulate deformable objects such as bags?

A detailed examination based on a literature review, experiments and analysis will be conducted for each one of the questions above. These questions answers hold the potential to provide helpful information about the benefits and limitations of using contextual information for training RL agents, as well as understanding the factors required to carry out RL in the real world.

1.3 Aim and Objectives

This thesis aims to investigate the effectiveness of using contextual information to improve the learning performance of RL agents. To achieve this aim, the following objectives are pursued:

- To investigate the effectiveness of using contextual information for training reinforcement learning agents in discrete environments. The goal of the first objective is to develop methods that can be safely tested and demonstrate how

context enhances RL sampling efficiency. Therefore, discrete environments such as games serve as suitable proof of concept before progressing to more complex scenarios involving robots.

- To develop and test a robotic RL context-based system to perform HRI for rigid objects in the real world. The second objective, which involves HRI scenarios, encompasses dynamic changes in the environment. Consequently, extracting contextual information has the potential to provide additional insights to the RL agent. Therefore, developing methods and testing the RL agent's capabilities to react to uncertain behaviours offers an ideal setup to identify weaknesses in current RL approaches. This process allows for the development of methods that address these weaknesses and enhance overall performance.

- To develop and test a robotic RL context-based system for learning to manipulate deformable objects in the real world. The third objective aims to explore the current capabilities of RL in handling complex manipulation problems. In this study, bags are employed for this purpose. Bags are deformable objects that have not been thoroughly explored to date. Therefore, it is essential to investigate how RL methods can effectively address this task.

By pursuing these objectives, this thesis aims to contribute to developing robust RL algorithms that can learn in simulation and the real world.

1.4 Outline

This outline presents a concise summary of the thesis' content comprised of six chapters, aiming to provide a clear overview of the research presented in the following chapters.

Chapter 2 introduces relevant literature in RL, affordances and robotics, and a critical analysis of the existing approaches by identifying the gaps and limitations in current research.

Chapter 3 introduces a framework for discrete environments called Iota Explicit Context Representation (IECR). This framework utilises the principle of Deep Q-network and involves encoding each state using the Contextual Key Frame (CKF) representation, which can then be used to extract a function that represents the affordances of the state (referred to as Iota in this work, which gives the name to the proposed framework); in addition, two loss functions are introduced with respect to the affordances of the state. The novelty of the IECR framework lies in its capacity to extract contextual information from the environment and learn from the CKFs' representation. The framework is validated by developing four new algorithms that learn using context: Iota Deep Q-network (IDQN), Iota Double Deep Q-network (IDDQN), Iota Dueling Deep Q-network (IDuDQN), and Iota Double Dueling Deep Q-network (IDDDQN). Furthermore, the framework and the new algorithms are evaluated in five discrete environments. The experiments show that all the algorithms which use contextual information significantly outperform the baseline algorithms by 77 % on average.

Chapter 4 introduces an affordance-based human-robot interaction framework, aiming to reduce the action space size that would considerably impede the exploration efficiency of RL agents. The framework is based on a new algorithm called CQL based on the principle of classic Q-learning, a tabular RL approach. The experiments show that the proposed algorithm trains in a reduced amount of time (2.7 seconds) and reaches an 84% success rate, while the baseline algorithm with the highest success rate reached 68% after learning during a significantly longer period of time (91.8 seconds). This suits the robot's learning efficiency in observing

the current scenario configuration and learning to solve it. During the HRI, the robot uses semantic information from the state and the optimal policy of the last training step to search for relevant changes in the environment that may trigger the generation of a new policy.

Chapter 5 presents an efficient learning-based robot-bagging framework where the novelty lies in its capacity to learn *bagging* in the real world. The learning process is accomplished through a new RL algorithm introduced in this work called Π-learning, designed to find the best grasping points of the bag based on a set of compact state representations. Moreover, Π-learning is a table-based RL approach. The framework utilises a set of primitive actions and represents the task in five states. In the experiments, the framework reaches a 60% and 80% success rate after around three hours of training in the real world when starting the bagging task from folded and unfolded positions, respectively. Finally, the trained model is tested on two more bags of different sizes to evaluate its generalisation capacities.

Chapter 6 concludes the thesis by summarising the main contributions and achievements of this research work. In addition, the limitations and challenges of the approaches developed in this thesis are discussed, and future research directions are proposed .

Overall, this thesis aims to contribute to the ongoing efforts to improve RL reliability and robustness in real-world applications.

Chapter 2

Literature Review

In the field of Reinforcement Learning (RL) and considering the objectives of this thesis, a comprehensive review of the current state-of-the-art is necessary to uncover insights into current challenges and unresolved problems. By examining the various applications, which include robotics and the use of contextual information as part of the training of RL agents, this chapter aims to explore the diverse RL algorithms, their work principles and applications in robotics. The rest of the chapter is organised as follows: section 2.1 introduces relevant concepts in RL. Then, section 2.2 explores how contextual information is involved in current state-of-the-art approaches. This is followed by section 2.3 that focuses on robotic applications, more concisely, on the use of RL for the manipulation of rigid and deformable objects. Finally, the main findings of the literature review are discussed in section 2.4 and summarised in section 2.5.

2.1 Reinforcement Learning Overview

RL is a sub-field of Machine Learning (ML) that focuses on training agents to learn how to maximise a reward in a given environment based on a human-like approach to learning by trial and error. RL algorithms are designed to allow the agent to learn from its own experience and, in this way, improve its performance

within each iteration with the environment. The RL framework is comprised of the following key concepts: agent, environment, state, action, reward, policy, state-value function, action-value function, exploration-exploitation trade-off, and Markov Decision Process (MDP). The definitions of these concepts are as follows:

- An **agent** is the learner, the one that makes decisions in the environment and gets feedback from the environment in the form of rewards.
- The **environment** is the system where the agent interacts, which can be simulated or real. An environment can be discrete (contains countable or finite elements) or continuous (contains uncountable or infinite elements). Additionally, an environment can be deterministic or stochastic.
- A **state** represents the current configuration of the environment based on sensor readings, images, or numbers. The state is denoted by s, and it belongs to the set of states S such that $s \in S$. The agent observes the state and makes a decision based on its training. The state can be discrete or continuous.
- An **action** can be taken by the agent given the current state in a certain environment. The set of actions can be discrete (i.e., up, down, left, and right) or continuous (a robot's joint that can rotate from 0 to 2π radians). An action an agent can take is denoted by a, and it belongs to the set of all possible actions A such that $a \in A$.
- A **model** describes the changes in the environment when the agent takes an action. The model can be provided or "discovered" by the agent through interaction with the environment. RL algorithms designed to learn with a given model are known as model-based, while those that do not rely on a predefined model are classified as model-free.
- The **reward** is a numerical value that the agent gets as feedback from the environment in a given state after performing an action and is denoted by R.

The reward has the goal of guiding the learning process of the agent throughout the training stage.

- A **policy** is the strategy the agent uses to make decisions in the environment. In other words, it is in charge of mapping the action-state pairs, which can be deterministic or stochastic. The policy is denoted by π.

- The **state-value function** predicts the expected reward starting from a given state s while following a policy π. This function is denoted by $V(s)$.

- The **action-value function** predicts the expected reward an agent would get if starting from a given state s and selecting a given action a while following a policy π. This function is denoted by $Q(s, a)$ and expresses the quality of a given state-action pair.

- The **exploration-exploitation trade-off** is a process in which the exploration phase is when the agent makes decisions stochastically, with the purpose of finding better alternatives. The exploitation phase seeks to use the knowledge of the agent to make decisions. Therefore, establishing a good exploration-exploitation trade-off strategy has a significant impact on the learning performance of RL agents.

- An **MDP** is a 4-tuple $\langle S, A, R, P \rangle$ where S is a set of states, A is a set of actions, R is a reward function, and $P(s'|s, a)$ is a transition function (Sutton and Barto 1998).

RL algorithms are designed to learn and solve an MDP. RL agents' objective is to learn a policy π that maximises the reward while mapping which action a is the best given a state s where $a \in A$ and $s \in S$. Usually, RL agents perform the following steps: observing the environment, selecting an action a, receiving a reward R, transitioning into a new state s' based on $P(s'|s, a)$, and updating the policy. To

this end, RL agents utilise a value function to estimate the quality of the action-state pairs.

Overall, RL algorithms can be classified into two branches (Fig. 2.1). The first one is where the algorithms utilise a table to store the transitions of the environment (classical RL algorithms), and the second one refers to the algorithms that are based on the use of neural networks (Deep RL algorithms). In the subsequent sections, the algorithms will be described.

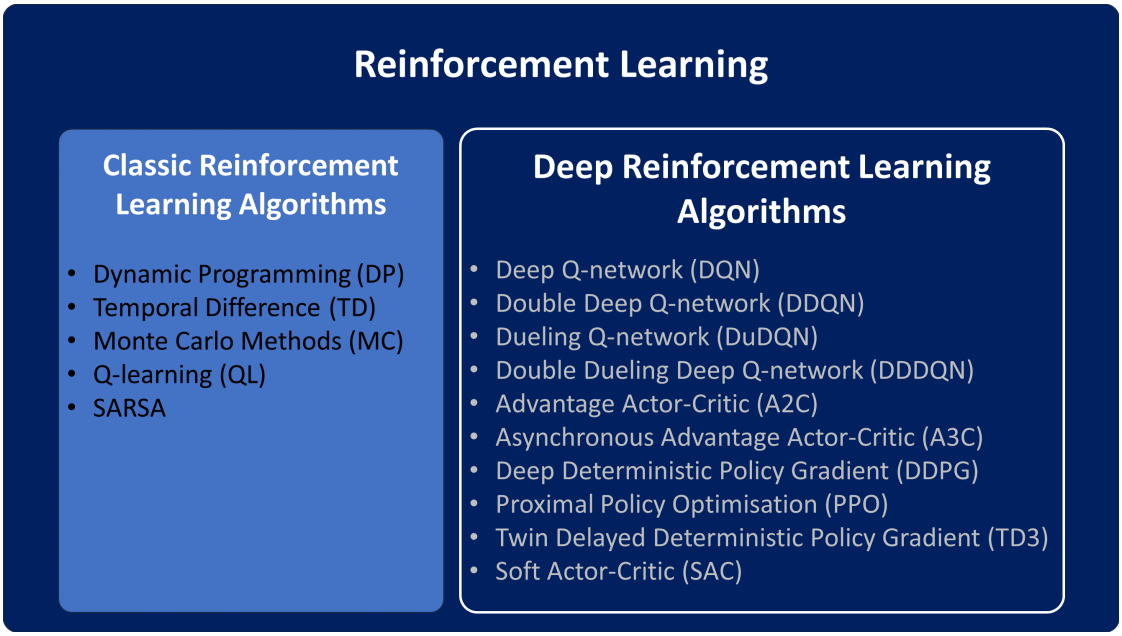


Figure 2.1: Popular RL algorithms.

2.1.1 Bellman Equations

The last section introduced the concept of the state-value function. However, how this function is calculated is not explained. In RL, the Bellman equation (Bellman 1966) is a fundamental concept that allows the computation of the value function $V(s)$. In this context, the Bellman equation can be expressed as follows:

$$V(s) = R(s,a) + \gamma \cdot V(s') \tag{2.1}$$

Here, $V(s)$ is the value of the state s, $R(s,a)$ is the immediate reward for transi-

tioning from s to s' while taking action a, γ is a discount factor and $V(s')$ is the value of the next state. When a policy π is given, then the value function can be expressed as follows:

$$V^{\pi}(s) = R(s,a) + \gamma \cdot V^{\pi}(s') \tag{2.2}$$

This form of the Bellman equation works when the environment is deterministic. However, when the environment presents stochastic behaviours, it is necessary to add the transition function $P(s'|s,a)$:

$$V^{\pi}(s) = \sum_{s' \in S} P(s'|s,a) \cdot (R(s,a) + \gamma \cdot V^{\pi}(s')) \tag{2.3}$$

Despite the fact that the equation above includes the stochasticity of the environment, it still needs to be considered when the policy is stochastic and not deterministic. Hence, to include a stochastic policy:

$$V^{\pi}(s) = \sum_{a \in A} \pi(a|s) \cdot \sum_{s' \in S} P(s'|s,a) \cdot (R(s,a) + \gamma \cdot V^{\pi}(s')) \tag{2.4}$$

The equation above is known as the Bellman expectation equation, and it can be rewritten in its expectation form:

$$V^{\pi}(s) = \mathbb{E}_{\substack{s' \sim P \\ a \sim \pi}} \Big[R(s,a) + \gamma \cdot V^{\pi}(s') \Big] \tag{2.5}$$

In equation (2.5), it can be observed that $V(s)$ evaluates the whole value of the state. However, it does not evaluate each action independently, and this is a limitation that can be solved by calculating the quality of every action-state pair with the action-value function $Q(s,a)$, better known as the Q function. Then, the Bellman equation of the Q function can be expressed as follows:

$$Q(s,a) = R(s,a) + \gamma \cdot Q(s',a') \tag{2.6}$$

Here, $Q(s, a)$ is the Q value of an action-state pair, $R(s, a)$ is the reward for transitioning from s to s' while taking action a, γ is a discount factor and $Q(s', a')$ is the Q value of the next state's action a'. When a policy π is given, then the Q-function can be expressed as follows:

$$Q^{\pi}(s, a) = R(s, a) + \gamma \cdot Q^{\pi}(s', a') \tag{2.7}$$

This form of the Bellman equation works when the environment is deterministic. However, when the environment presents stochastic behaviours, it is necessary to add the transition function $P(s'|s, a)$:

$$Q^{\pi}(s) = \sum_{s' \in S} P(s'|s, a) \cdot (R(s, a) + \gamma \cdot Q^{\pi}(s', a')) \tag{2.8}$$

Despite the fact that the equation above includes the stochasticity of the environment, it still needs to be considered when the policy is stochastic and not deterministic. Hence, to include a stochastic policy:

$$Q^{\pi}(s, a) = \sum_{s' \in S} P(s'|s, a) \cdot (R(s, a) + \gamma \sum_{a' \in A} \pi(a|s) \cdot Q^{\pi}(s', a')) \tag{2.9}$$

The equation above is known as the Bellman expectation equation of the Q function, and it can be rewritten in its expectation form:

$$Q^{\pi}(s, a) = \mathbb{E}_{s' \sim P}\left[R(s, a) + \gamma \cdot \mathbb{E}_{a' \sim \pi} Q^{\pi}(s', a')\right] \tag{2.10}$$

In general, the value and Q function Bellman equations are used for calculating the value of the state and the quality of the Q value's state-action pairs, respectively. These equations are based on the models of the environment given by $P(s'|s, a)$, a policy π and the reward function $R(s, a)$.

2.1.2 The Bellman optimality equation

The Bellman optimality equation expresses the expected maximum or total reward that can be achieved from a given state based on the value function (Eq. (2.4)). The equation defines the relationship between the value of a state and the values of its neighbouring states. The Bellman optimality equation of the value function is given by the following:

$$V^*(s) = \max_a \sum_{s' \in S} P(s'|s,a) \cdot (R(s,a) + \gamma \cdot V^*(s')) \tag{2.11}$$

The equation states that the optimal value of a state is the maximum expected value obtained by taking the best action in the current state. Moreover, this equation considers the expected values of the resulting states. On the other hand, the Bellman optimality equation of the Q function can also be calculated and is given by the following:

$$Q^*(s,a) = \sum_{s' \in S} P(s'|s,a) \cdot (R(s,a) + \gamma \cdot \max_{a'} Q^*(s',a')) \tag{2.12}$$

Additionally, the relationship between the value and Q functions is that the value function can be derived from the Q function by selecting the maximum Q value for each state. In order to calculate the value function based on the Q function, the maximum Q value over all possible actions in a given state is selected. Consequently, for each state, the action that maximises the Q value is selected, which becomes the value of that state. Therefore, the value $V(s)$ is equal to the maximum Q value for a given state s. Then, the relationship can be expressed as follows:

$$V^*(s) = \max_a Q^*(s,a) \tag{2.13}$$

Following the logic from the previous sentence, if the maximum value of $V^*(s)$ corresponds to the maximum value of $Q^*(s,a)$, then the Q function can be derived

from the value function by substituting Eq. (2.13) in Eq. (2.12), then:

$$Q^*(s,a) = \sum_{s'} P(s'|s,a) \cdot (R(s,a) + \gamma \cdot V^*(s)) \tag{2.14}$$

Overall, the relationship between the value and Q functions is that the value function can be derived from the Q function by selecting the maximum Q value for each state. At the same time, the Q function can be derived from the value function by using the Bellman equation.

2.1.3 Dynamic Programming

Dynamic Programming (DP) is a method used in mathematics to solve optimisation problems. This method divides the problem into smaller subproblems. Then, the method recursively solves the subproblems and stores their solutions in a table or array. In the context of RL, DP is used to find an optimal policy π^* from the Bellman equations introduced in the last subsection. To this end, two DP methods exist: value iteration and policy iteration.

In the value iteration method, the optimal value function is computed by iteratively taking the maximum Q value over the Q function (Eq. (2.14)). Then, within each step, the value function is updated with Eq. (2.13). Lastly, the optimal policy is extracted from the optimal value function while computing the Q values with the following:

$$\pi^* = \arg\max_a Q(s,a) \tag{2.15}$$

The policy interaction method computes the optimal value function using the policy iteratively. In other words, the method starts with a random policy π_0 and computes its value function with Eq. (2.4). Then, it is necessary to extract a policy π_1 derived from the previous one. To this end, the $V(s)$ values al already available from the last interaction, and they can be used to compute the Q values with Eq. (2.14).

Consequently, a new policy can be extracted by using Eq. (2.15). This process is repeated till finding π^*.

In summary, the Bellman optimality equations and the relationship between the value and Q functions can be applied interactively with DP to find an optimal policy. The value iteration method first computes the Q values to update the value function, and at the end, it extracts the optimal policy. The policy interaction method utilises a random policy that eventually becomes optimal.

2.1.4 Monte Carlo Methods

A problem with the approaches mentioned above is that the transition function is given (model-based), and they do not consider the case when the dynamics of the environment are unknown (model-free). In the context of RL, Monte Carlo (MC) methods are algorithms used to estimate the value function and find the dynamics of the environment through interaction (Browne et al. 2012). The expected value of the value function of a given state can be approximated by visiting that state for N times and obtaining the total return:

$$V(s_t) \approx \frac{1}{N_t} \sum_{i=1}^{N_t} G_i \tag{2.16}$$

Here, G_i is the total return, s_t corresponds to the step t when that state was visited (this must not be confused with the expression $V(s)$ or $V(s')$ from last subsections, in the case of MC methods the term t is necessary to update the value of a state at a given step), and N_t denotes the number of times the s_t state in a given step t has been visited. In order to establish a balance between computational efficiency and memory requirements, instead of using the arithmetic mean as in Eq. (2.16), the incremental mean is more commonly used, and it is expressed as follows:

$$V(s_t) \approx V(s_t) + \alpha \cdot (G_t - V(s_t)), \tag{2.17}$$

where $\alpha = \frac{1}{N_t}$. MC methods also focus on solving the problem of estimating the Q function for a given policy. Hence, the following expression can be deduced:

$$Q(s_t, a_t) \approx Q(s_t, a_t) + \alpha \cdot (G_t - Q(s_t, a_t)) \tag{2.18}$$

MC methods have two variations: first-visit MC and every-visit MC. In the first-visit MC, the algorithm only updates the value or Q function once per episode. Contrary, every-visit MC updates the value or Q function every time. The main idea is to use returns obtained from actual episodes to approximate the expected returns. The MC algorithms are comprised of the following steps: episode collection, value function estimation, and policy improvement.

During the episode collection, the agent interacts with the environment by following a given policy. Besides that, it collects experience and computes the return during each episode. Every episode ends with a final state or a terminal condition. For the value function estimation, depending on the variant, Eq. (2.17) or Eq. (2.18) is used. The return is the cumulative sum of discounted rewards from a given state until the end of the episode. The more episodes, the more accurate the estimation. Once the value function is estimated, the agent can use it to optimise the policy. This is done by selecting actions that maximise the estimated value function.

In general, MC methods are model-free algorithms that perform the described steps iteratively. Where the agent collects episodes, updates the estimated value function, and improves the policy until it transforms it into an optimal one.

2.1.5 Temporal Difference

Despite the potential of MC methods, their working principle design relies on reaching a terminal state to approximate a value function. A drawback is that if the episode is too long, it means costly computation. In this context, Sutton and Barto (1998) proposed an alternative that balances Bellman equation methods and MC

methods. The approach is known as Temporal Difference (TD). These approaches learn by bootstrapping, meaning that instead of waiting until the end of an episode, they update their value estimation based on the following:

$$V(s) \approx r + \gamma \cdot V(s'), \tag{2.19}$$

where r is the immediate reward obtained after transitioning from state s to the next state s'. Here, similar to the replacement of the average mean for the incremental mean in Eq. (2.17), Eq. (2.19) can be expressed as follows:

$$V(s) \longleftarrow V(s) + \alpha \cdot (r + \gamma \cdot V(s') - V(s)), \tag{2.20}$$

The last expression, known as the TD estimate rule, can be used to estimate the value of the state. However, it is necessary to apply the same logic to the Q function to have an optimal policy based on that estimate. The TD estimate rule for the Q function is given by the following:

$$Q(s,a) \longleftarrow Q(s,a) + \alpha \cdot (r + \gamma \cdot Q(s',a') - Q(s,a)). \tag{2.21}$$

Similar to MC methods, TD methods can be on-policy or off-policy. On-policy methods utilise the same policy that the agent is improving. The policy used for exploration-exploitation is also the policy that is updated based on the collected experiences. MC methods are the most common on-policy methods. Off-policy methods are the ones that update their action-value function regardless of the action actually taken. In other words, off-policy methods are designed to let the agent learning from experiences generated by a different policy than the one being updated. For example, the State-Action-Reward-State-Action (SARSA) algorithm (Rummery and Niranjan 1994) is a TD method that learns the policy-value function from the policy that is following. Another approach is Q-learning QL, an off-policy algorithms. Both algorithms utilise an ϵ-greedy policy that encourages exploration

of the environment, iteratively updates the value estimates, and improves the policy.

In summary, one of the most important advantages of TD methods is their ability to learn online while combining the benefits of MC methods (being model-free) and DP (bootstrapping from estimated values).

2.1.6 Q-learning

Among several approaches, Q-learning (QL) is a popular off-policy RL algorithm that learns an MDP in discrete environments. This algorithm is also model-free because it does not require any previous model of the environment. QL is designed to update the quality values Q of a state-action combination, given by:

$$Q : A \times S \longrightarrow \mathbb{R}, \tag{2.22}$$

where Q are the Q-values (usually stored in a table), and they represent the quality of the state-action pair. In other words, the higher the Q-value, the better the action for that state. Then, a Q-table is necessary to represent each Q-value such that QL updates the state-action pair by using the following:

$$Q(s,a) \longleftarrow Q(s,a) + \alpha \cdot [r + \gamma \cdot \max_{a'} Q(s',a') - Q(s,a)], \tag{2.23}$$

where α is the learning rate, r is the reward and γ is the discount factor. The discount factor is usually a value between 0 and 1 ($0 \leq \gamma \leq 1$) that balances the importance the agent puts on future rewards rather than immediate rewards. According to equation (2.23), the state-action pair is updated based on the next state s' even when that state has not been explored, which is why QL is considered an off-policy method.

2.1.7 State-Action-Reward-State-Action

The State-Action-Reward-State-Action (SARSA) algorithm was first proposed by Rummery and Niranjan (1994), and its main difference with QL lies in the manner in which the Q values are updated. SARSA is designed to learn from its own policy. Hence, SARSA is categorised as an on-line RL algorithm. The update rule of SARSA is given by the following:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \cdot [r + \gamma \cdot Q(s',a') - Q(s,a)], \tag{2.24}$$

where α is the learning rate, r is the reward and γ is the discount factor. In the case of SARSA, the maximum Q value of the next state-action pair is not used as in QL (see Eq. (2.23)). In terms of equations, this is the main difference between both algorithms.

Characteristic	**Bellman Equations**	**MC**	**DP**	**TD**	**QL**	**SARSA**
Off-Line	✓	✓	✓	✓		
On-Line	✓	✓		✓	✓	✓
Model-Free		✓		✓	✓	✓
Model-Based	✓		✓			
Off-Policy	✓	✓			✓	
On-Policy		✓	✓	✓		✓
Discrete Action	✓	✓	✓	✓	✓	✓
Continuous Action	✓	✓	✓	✓		
Discrete State	✓	✓	✓	✓	✓	✓
Continuous State	✓	✓	✓	✓		
Supports large State Spaces						

Table 2.1: Classical RL algorithms characteristics.

2.1.8 Deep Q-network

A problem with the approaches described before is that when the state space is too large, or the number of states is infinite, the approach becomes impractical and impossible to implement (see Table 2.1). Hence, a solution to this problem is to

utilise a neural network that can approximate all possible state-action pairs. Since the implementation of neural networks RL is commonly referred to as Deep Reinforcement Learning (DRL) (Goodfellow et al. 2016), the approach of implementing neural networks into QL is known as Deep Q-network (DQN). However, the problem now is related to finding a proper set of weights for the neural network Q^{θ}. In this context, the Q function $Q^{\pi}(s,a)$ denotes the value of an action-state pair that approximates the expected future reward that can be obtained from $R(s,a)$ when a policy π is given. The optimal value function $Q^{*}(s,a)$ can be obtained from the Bellman optimality equation:

$$Q^{*}(s,a) = \mathbb{E}_{s' \sim P}\left[R(s,a) + \gamma \cdot \max_{a'} Q^{*}(s',a')\right] \tag{2.25}$$

In Equation (2.25), the reward function $R(s,a)$ is equivalent to the immediate reward r, which is obtained when performing an action a in a given state. Therefore, the Q function can be approximated with a neural network $Q_{\theta}(s,a)$ that can be trained by utilising k transitions from a replay buffer D^{replay}. Then, the expectation can be removed and approximated by using bootstrapping. The optimal value function $Q^{*}(s,a)$ is:

$$Q^{*}(s,a) = r + \gamma \cdot \max_{a'} Q(s',a') \tag{2.26}$$

In order to stabilise the learning performance, DQN uses two neural networks (Mnih et al. 2015). The main neural network Q^{θ} computes the current value of the given pair (s,a). The target neural network parameterised by θ and denoted by $Q^{\theta'}(s,a)$ is frozen for n training steps and is used to compute the next pair (s',a'). Consequently, the target function $Q^{\theta'}(s,a)$, the current values of the main neural network $Q^{\theta}(a,s)$, and equation (2.26) can be used to obtain the loss function $J_{dn}(Q)$, as follows:

$$J_{dn}(Q) = r + \gamma \cdot \max_{a'} Q^{\theta'}(s',a') - Q^{\theta}(s,a) \tag{2.27}$$

In order to use the replay buffer to train the neural network, it is necessary to calculate the Mean Squared Error (MSE) and minimise it. Hence, instead of using only equation (2.27), it is necessary to use the k transitions stored in the buffer, as follows:

$$J_{dn}(Q) = \frac{1}{k}\sum_{i=1}^{k}(r_i + \gamma \cdot \max_{a'} Q^{\theta'}(s'_i, a') - Q^{\theta}(s_i, a_i))^2 \tag{2.28}$$

For simplicity, the expression $r + \gamma \cdot \max_{a'} Q^{\theta'}(s'_i, a')$ can be expressed as the target value y_i, which is restricted to the following condition:

$$y_i = \begin{cases} r & \text{if the state is terminal,} \\ r + \gamma \cdot \max_{a'} Q^{\theta'}(s'_i, a') & \text{otherwise.} \end{cases} \tag{2.29}$$

Then the loss function can be expressed as follows:

$$J_{dn}(Q) = \frac{1}{k}\sum_{i=1}^{k}(y_i - Q^{\theta}(s_i, a_i))^2 \tag{2.30}$$

The weights θ are updated with the following:

$$\theta = \theta - \alpha \cdot \nabla_{\theta} J_{dn}(Q) \tag{2.31}$$

Overall, DQN utilises two neural networks. One is the main source of action selection, known as the main neural network with weights θ. The second one is the target neural network with weights θ', and this structure is used to stabilise the learning progress of the agent. The equation (2.26) detonates that the optimal value function depends on the reward and the Q-value of the next state-action pair. To this end, a loss function, given by equation (2.27), increases its value when the current Q-values are different from the optimal values. Consequently, after training for n steps, a replay buffer is collected, and the loss function is used interactively to

get the gradients that reduce the loss. Finally, the weights *theta* are used to update the θ' weights after a certain number of training steps (usually 100).

2.1.9 Double Deep Q-network

For some environments, DQN may overestimate the Q-values of the next state, leading to bias preference towards actions that are not optimal. Double Deep Q-network (DDQN) is a variant of DQN, which is designed to deal with this problem (Hasselt 2010). More concisely, while DQN takes the maximum Q-value of the target neural network, DDQN takes the index of the maximum Q-value from the main neural network and then takes the value of that index from the target neural network. In this way, DDQN solves the problem of overestimating the states that DQN presents in some environments. DDQN takes the maximum index of the main neural network and then the actual value of that index in the target neural network to calculate the loss $J_{ddn}(Q)$:

$$J_{ddn}(Q) = r + \gamma \cdot Q^{\theta'}(s', \arg\max_{a'} Q^{\theta}(s', a')) - Q_{\theta}(s, a) \tag{2.32}$$

Similar to DQN, when using the replay buffer to train the neural network with DDQN, the loss function can be expressed as follows:

$$J_{ddn}(Q) = \frac{1}{k}\sum_{i=1}^{k}(r_i + \gamma \cdot Q^{\theta'}(s'_i, \arg\max_{a'} Q^{\theta}(s'_i, a'_i)) - Q_{\theta}(s_i, a_i))^2 \tag{2.33}$$

2.1.10 Dueling Architectures

A problem with DQN and DDQN is that both algorithms have the value estimation function embedded in the same network, which may provoke some problems, such as ambiguity, when the agent can take two actions that lead to the same goal. This can result in overgeneralisation, in other words, the value estimates for actions

that are not relevant or are propagated to other states. Dueling architectures (Z. Wang et al. 2016) use two output separate estimators: the advantage output stream $A_\theta(s,a)$ and the $V_\theta(s)$ output stream (refer to Fig. 2.2). By using this configuration, dueling architectures enhance their learning efficiency while allowing for better value estimation. Dueling Deep Q-network (DuDQN) uses two streams in the neural network to estimate separately the advantage $A_\theta(s,a)$ that is given by the following:

$$A_\theta(s,a) = Q_\theta(s,a) - V_\theta(s), \tag{2.34}$$

where $A_\theta(s,a)$ is the advantage, and $V_\theta(s)$ is the value of the state s. Even though the actions are selected using the advantage stream channel, the loss function (2.27) for DuDQN is the same used for DQN. Finally, Double Dueling Deep Q-network (DDDQN) is the implementation of the DDQN principle to DuDQN.

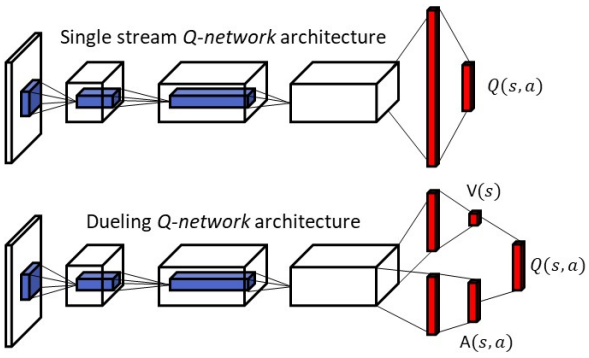


Figure 2.2: Typical neural network architectures used in DQN's variants (Hasselt 2010).

2.1.11 Actor-Critic Methods

The algorithms discussed above are designed to learn a value function. On the contrary, Actor-Critic methods (Mnih et al. 2016) are model-free approaches combining policy- and value-based elements. These methods consist of two components: an actor and a critic. On one hand, the actor selects actions based on the current state. It is represented by a policy which can be stochastic or deterministic. The critic

evaluates the actions taken by the actor by estimating the expected cumulative reward or value function associated with a given state or state-action pair. On the other hand, the critic learns to approximate the value function based on the TD or MC methods discussed previously. Both components, the actor and critic, work iteratively such that while the actor explores the environment, the critic estimates the value function to provide feedback to the actor.

Broadly speaking, there are two algorithms based on Actor-Critic methods: Advantage-Actor Critic (A2C) and Asynchronous Advantage Actor-Critic (A3C). A2C algorithm utilises two neural network function approximations, one for the actor-network θ, and the other one for the critic-network ϕ. The parameters of the actor-network are updated with the following:

$$\theta = \theta + \alpha \cdot \nabla_\theta J(\theta) \tag{2.35}$$

Here, $J(\theta)$ is the loss function. The preceding equation aims to maximise the rewards obtained by the policy π_θ such that the gradient with respect to θ of the loss is given by the following:

$$\nabla_\theta J(\theta) = \nabla_\theta \log \pi_\theta(a_t|s_t)(r + V_\phi(s'_t) - V_\phi(s_t)), \tag{2.36}$$

where $\pi_\theta(a_t|s_t)$ is a policy that follows a probability density function at a given time-step t. In other words, the neural network's output is the mean and standard deviation of the Probability Density Function (PDF) given by the policy $\pi_\theta(a_t|s_t)$. Whereas $V_\phi(s'_t)$ and $V_\phi(s_t)$ are the values of the current and next state predicted with the neural network ϕ. The weight values ϕ are updated with the following:

$$\phi = \phi - \alpha \cdot \nabla_\phi J(\phi) \tag{2.37}$$

Here, the loss $J(\phi)$ is given by the following:

$$J(\phi) = r + V_\phi(s'_t) - V_\phi(s_t) \tag{2.38}$$

The A3C algorithm introduces the concept of asynchrony to improve training efficiency. Besides that, A3C train multiple agents in parallel. For this purpose, a copy of the environment is given to each agent. All the agents collect experiences independently. Then, these experiences are used to update the shared parameters of the actor and critic networks. A3C adds a new term to the loss function utilised by A2C, which is known as the entropy or measurement of randomness given by $M(\pi(s)$. By encouraging a higher entropy, A3C promotes exploration and prevents the policy from focusing on a limited set of actions. The loss is given by the following:

$$J(\theta) = \log \pi_\theta(a_t|s_t)(r + V_\phi(s'_t) - V_\phi(s_t)) + \beta M(\pi(s)), \tag{2.39}$$

where β controls the significance of the entropy. A3C utilises several agents known as workers and the global agent. In A3C, several workers perform actions and compute the losses (Eq. (2.38) and Eq. (2.39)) that are then used to update the global agent parameters.

In general, agents based on actor-critic methods can learn a value function and optimise a policy simultaneously while iteratively interacting with the environment.

2.1.12 Deep Deterministic Policy Gradient

The algorithms discussed above are designed to solve environments with discrete action spaces. Contrary, Deep Deterministic Policy Gradient (DDPG) (Lillicrap et al. 2015) is an Actor-Critic model-free method that combines elements from both policy-based methods and value-based methods. DDPG works in continuous action space environments. Similar to DQN variants, DDPG also utilises the idea of main and target neural networks. A replay buffer, which stores experiences encountered during training, is also utilised. Additionally, DDPG uses the Actor-Critic principle

to train an agent. Hence, four neural networks are used: main critic ϕ and target criticϕ', main actor θ and target actor θ'. The main critic neural network weights are updated with the following:

$$\theta = \theta + \alpha \cdot \nabla_\theta J(\theta) \tag{2.40}$$

Here, the loss $J(\theta)$ is computed as follows:

$$J(\theta) = \frac{1}{k}\sum_{i=1}^{k}(r_i + \gamma \cdot Q_\theta(s'_i, \mu_{\phi'}(s_i)) - Q_\theta(s_i, a_i))^2, \tag{2.41}$$

where i is the index of experiences in the replay buffer. Each replay experience is denoted by the tuple (s_i, a_i, r_i, s'_i), s_i is the current state, a_i is the action taken, r_i is the reward received, and s'_i is the next state. The loss $J(\theta)$ is computed by summing all the transitions in the replay buffer and involves the main critic's and actor's outputs. Additionally, $\mu_{\phi'}$ is the deterministic policy generated by the actor neural network ϕ', and k samples are randomly taken from the replay buffer. The exploration of this algorithm is achieved by applying the Ornstein-Uhlenbeck random process consisting of injecting random noise such that:

$$a = \mu_\phi + \mathbf{N}, \tag{2.42}$$

where $\mathbf{N}$ is the noise. The actor's main neural network is updated by using the following equation:

$$\phi = \phi - \alpha \cdot \nabla_\phi J(\phi) \tag{2.43}$$

Here, the objective function $J(\phi)$ is given by:

$$J(\phi) = \frac{1}{K}\sum_{i=1}^{K} Q_\theta(s_i, \mu_\phi(s_i)) \tag{2.44}$$

For updating the target actor neural network, it is necessary to copy the weights

from the main actor neural network by applying soft replacement:

$$\phi' = \omega \cdot \phi + (1 - \omega)\phi', \tag{2.45}$$

where ω is usually set to 0.001. In a similar manner, the target critic neural network can be updated by applying soft replacement:

$$\theta' = \omega \cdot \theta + (1 - \omega)\theta' \tag{2.46}$$

DDPG is designed to learn an optimal policy that maximises the expected cumulative reward in continuous action spaces. This is achieved by iteratively updating the actor and critic networks with Eq. (2.40) and Eq. (2.44) with samples from the replay buffer.

2.1.13 Twin Delayed Deep Deterministic Policy Gradient

Twin Delayed Deep Deterministic Policy Gradient (TD3) (Fujimoto et al. 2018) the successor of DDPG, and it implements three major changes: clipped double Q-Learning, delayed policy update and target policy smoothing. First, similar to DQN, DDPG tends to overestimate the action-state pair of the next state. To address this issue, the Double Q-learning principle is implemented. For this purpose, TD3 utilises twin critics, which are two critic neural networks instead of the only one DDPG uses. This is known as clipped double Q-learning. In this way, two target critic networks are used to compute the two Q values and select the minimum value out of these two. Then, based on that information, the target value is calculated. Consequently, two main critic neural networks with parameters θ_1 and θ_2 are defined, while for the target critic neural networks, the parameters are θ'_1 and θ'_2. The parameters are updated with the following equations:

$$\theta_1 = \theta_1 - \alpha \cdot \nabla_{\theta_1} J(\theta_1) \tag{2.47}$$

$$\theta_2 = \theta_2 - \alpha \cdot \nabla_{\theta_2} J(\theta_2) \tag{2.48}$$

Their respective loss functions are expressed as follows:

$$J(\theta_1) = \frac{1}{K} \sum_{i=1}^{K} (r + \gamma \cdot \min_{j=1,2} Q_{\theta_j}(s_i', \hat{a}) - Q_{\theta_1}(s_i, a_i))^2 \tag{2.49}$$

$$J(\theta_2) = \frac{1}{K} \sum_{i=1}^{K} (r + \gamma \cdot \min_{j=1,2} Q_{\theta_j}(s_i', \hat{a}) - Q_{\theta_2}(s_i, a_i))^2 \tag{2.50}$$

Here, the k represents a sample minibatch of transitions, and the target policy smoothing principle applies by adding some noise to $\hat{a}$:

$$\hat{a} = \mu_{\phi'} + \epsilon, \tag{2.51}$$

where $\epsilon \sim \mathcal{N}(0, \sigma)$. Contrary to DDPG, TD3 does not update the actor, target critic and target actor networks within every step of execution. Instead, TD3 waits for t steps and then updates the parameters, this is known as delayed policy update. The actor-network is updated with the following:

$$\phi = \phi - \alpha \cdot \nabla_{\phi} J(\phi) \tag{2.52}$$

Here, the objective function $J(\phi)$ is given by:

$$J(\phi) = \frac{1}{k} \sum_{i=1}^{K} Q_{\theta_1}(s_i, \mu_{\phi}(s_i)) \tag{2.53}$$

For updating the target actor neural network and both target critic neural networks, it is necessary to copy the parameters from the main neural networks by applying soft replacement:

$$\phi' = \omega \cdot \phi + (1 - \omega)\phi', \tag{2.54}$$

$$\theta_1' = \omega \cdot \theta_1 + (1 - \omega)\theta_1', \tag{2.55}$$

$$\theta_2' = \omega \cdot \theta_2 + (1 - \omega)\theta_2', \tag{2.56}$$

TD3 is an actor-critic method designed for continuous action spaces and learns from a replay buffer generated by itself. In general, the aim of TD3 is to mitigate the overestimation problem of DDPG and to maximise the cumulative reward.

2.1.14 Soft Actor-Critic

Soft Actor-Critic (SAC) (Haarnoja et al. 2018) is designed for continuous action spaces and combines the advantages of Actor-Critic methods. For example, SAC utilises entropy regularisation, and contrary to DDPG and TD3, SAC follows a stochastic policy. Besides that, SAC uses five neural networks. Two of these are in charge of approximating the value, and there is a main value network represented by ψ and a target value network ψ'. Similar to TD3, it has two main Q-networks represented by θ_1 and θ_2. Since SAC is an Actor-Critic method, the parameters of the neural networks are updated every step of the episode. Let:

$$a = \pi_\phi(s), \tag{2.57}$$

be the policy that returns the action a parameterised by ϕ. After performing the action, it is possible to calculate the value loss with the following:

$$J_v(\psi) = \frac{1}{k} \sum_{i=1}^{k} (\min_{j=1,2} Q_{\theta_j}(s_i, a_i) - \alpha \cdot \log \pi_\phi(a_i|s_i) - V_\psi(s_i))^2 \tag{2.58}$$

Then, the main value neural network can be updated with:

$$\psi = \psi + \lambda \cdot \nabla_\psi J_v(\psi) \tag{2.59}$$

Now, it is possible to compute the Q-network's losses by using the following:

$$J_Q(\theta_j) = \frac{1}{K}\sum_{i=1}^{K}(r_i + \gamma \cdot V_{\phi'}(s_i') - Q_{\theta_j}(s_i))^2 \tag{2.60}$$

Here, $j = 1, 2$ and for updating the parameters of the Q-networks the following expressions are used:

$$\theta_1 = \theta_1 - \lambda \cdot \nabla_{\theta_1} J_Q(\theta_1) \tag{2.61}$$

$$\theta_2 = \theta_2 - \lambda \cdot \nabla_{\theta_2} J_Q(\theta_2) \tag{2.62}$$

At this point, it can be noted why SAC is an Actor-Critic approach. However, a key characteristic of SAC is in the policy objective function that integrates the entropy of the policy. In other words, by maximising this objective function, the agent not only seeks a better action-state pair but also encourages exploration at the same time. The policy objective function is given by:

$$J_\phi(\theta) = \frac{1}{k}\sum_{i=1}^{k}[Q_{\theta_1}(s_i, a_i) - \alpha \cdot \log \pi_\phi(a_i|s_i)], \tag{2.63}$$

where $\log \pi_\phi(a|s_i)$ represents the entropy of the policy. Based on the preceding objective function, the parameters of the policy network can be updated as follows:

$$\phi = \phi + \lambda \cdot \nabla_\phi J_\pi(\phi) \tag{2.64}$$

At the end, the target value network parameters are updated by using soft replacement:

$$\psi' = \omega \cdot \psi + (1 - \omega)\psi' \tag{2.65}$$

The same process is repeated iteratively, aiming to improve the policy within each

training step. Overall, the two key components of SAC are the addition of a stochastic policy and the use of entropy regularisation, which encourages exploration while maximising the cumulative rewards.

2.1.15 Proximal Policy Optimisation

Proximal Policy Optimisation (PPO) is an on-policy algorithm suitable for discrete and continuous environments. PPO utilises a policy network to represent the policy. The policy network outputs a probability distribution over the available actions, which can be soft-max or Gaussian for discrete or continuous action spaces, respectively. PPO generates a number of experiences N by following the current policy π_{theta} such that $\tau^{i}{}_{i=1}^{N}$ are the experiences. The algorithm uses these experiences to evaluate the policy by estimating the advantage values for each state-action pair. Moreover, PPO utilises a value network parameterised by ϕ. The objective function that PPO aims to maximise is the following:

$$L(\theta) = \mathbb{E}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \cdot A_t \right] \tag{2.66}$$

Here, the term $\pi_\theta(a_t|s_t)/\pi_{\theta_{old}}(a_t|s_t)$ expresses the probability ratio, which serves as a proximity measure. In other words, it represents how far or close the new policy is from the old one. For simplicity, the probability ratio can be expressed by:

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{old}(a_t|s_t)} \tag{2.67}$$

PPO ensures that the change of the policy is within a certain range or better known as a thrust region by adding a function called clipping function, such that the objective function is rewritten as:

$$L(\theta) = \mathbb{E}_t[\min(r_t(\theta)A_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t)], \tag{2.68}$$

where ϵ is a value usually set to 0.1 or 0.2. Besides, the preceding equation implies that when $A_t > 0$, the action must be preferred over the others. Hence, the probability ratio $r_t(\theta)$ can be increased. However, to keep the value close to the old policy, the restriction $1 + \epsilon$ applies. On the other hand, when $A_t < 0$, the corresponding action should not be preferred over the other ones. Therefore, the $r_t(\theta)$ value should be decreased while respecting the bound $1 - \epsilon$. In general, this is what the function clip does. For updating the policy network, the following equation is used:

$$\theta = \theta + \alpha \cdot \nabla_\theta L(\theta) \tag{2.69}$$

The loss function of the value network is given by:

$$J(\phi) = \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{T-1} (R_t - V_\phi(s_t))^2, \tag{2.70}$$

where R_t is the return collected by following the policy π_θ for N times. The parameters of the value network are updated with the following:

$$\phi = \phi - \alpha \cdot \nabla_\phi L(\phi) \tag{2.71}$$

This process is repeated iteratively for several training steps. Overall, PPO collects experiences, estimates advantages, and seeks to optimise the policy and value networks. Finally, PPO aims to keep the policy updates within the trust region by using the clipping function and, in this way, stabilise the agent's learning process.

2.1.16 Hindsight Experience Replay

Hindsight Experience Replay (HER) (M. Andrychowicz et al. 2017) is a method that aims to address the problem of sparse rewards in goal-oriented tasks. As the name suggests, HER has the ability to understand and evaluate experiences after they have occurred. Unlike other RL algorithms, HER allows learning from both successful and unsuccessful outcomes. This is achieved by defining hindsight goals,

which are created when the agent fails to reach its intended destination. In other words, by incorporating multiple hindsight goals, HER encourages generalisation as the agent learns to reach different configurations instead of solely focusing on the initial main goal during training.

In this way, the agent can learn even from failed attempts. Consequently, the agent not only generalises better but also explores alternative trajectories, thereby enhancing its exploration efficiency. Additionally, this technique transforms unsuccessful transitions into valuable ones because when the agent fails to reach the main goal, a hindsight goal is defined. The next time the agent samples that transition from the replay buffer, it already performed the correct way to reach the hindsight goal.

Overall, HER is a useful technique that can be applied to various RL algorithms (see Table 2.2), such as DQN, DDPG, TD3, SAC, and PPO, to address problems related to sparse rewards and multi-goal setups.

Characteristic	**DQN**	**DDQN**	**DuDQN**	**DDDQN**	**A2C**	**A3C**	**DDPG**	**TD3**	**SAC**	**PPO**
Off-Line	✓	✓	✓	✓						
On-Line	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Model-Free	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Model-Based										
Off-Policy	✓	✓	✓	✓			✓	✓	✓	
On-Policy					✓	✓				✓
Discrete Action	✓	✓	✓	✓	✓	✓				✓
Continuous Action					✓	✓	✓	✓	✓	✓
Discrete State	✓	✓	✓	✓	✓	✓				✓
Continuous State	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
HER	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Table 2.2: Deep RL algorithms characteristics.

2.2 Context and Reinforcement Learning

Context provides meaning to raw data, reduces ambiguity, and focuses attention on a clear objective; a situation is incomprehensible without context. In many RL

environment settings, contextual data can be accessed, but it takes time for an RL agent to learn directly from this information. Moreover, it is not easy to differentiate between two states because of the stochastic nature of RL (Fujimoto et al. 2019; Ribeiro 2002). A critical gap in knowledge is the difficulty in exploiting available contextual information, such as affordances, objects, positions, shapes, and more physical characteristics, which may be used to differentiate between two states and, in this way, improve the sampling quality of the agent during the exploration of the environment. Ideally, an agent should learn as fast as a human does (Finn et al. 2017; Voss et al. 2020). This section introduces popular and relevant literature about how contextual information has been incorporated into RL, and it is summarised in three subsections: context-free, implicit context-based and explicit context-based methods. Table 2.3 summarises the main differences among the three classifications. For context-free methods, any demonstrations or pertaining are given. Besides that, all the learning is performed on-line. Implicit context-based methods are the ones that include some off-line training, including previous generation of datasets from demonstrations and pre-trained neural networks. Lastly, explicit context-based methods are the ones that do not need any demonstrations or pre-training. However, a set of rules (affordances) can be applied during the agent's training in an on-line setup.

Characteristic	Context-free	Implicit context	Explicit context
Demonstrations		✓	
Affordances			✓
Pre-training		✓	
Off-Line		✓	
On-Line	✓		✓

Table 2.3: Context consideration in RL.

2.2.1 Context-free Methods

This subsection comprises methods in which no contextual information is explicitly given, and consequently, the agent learns everything from scratch by relying only upon the reward function and states. For example, DQN, which, as discussed before, has several state-of-the-art variants (Urtans and Nikitenko 2018), has shown

versatility across different applications. Part of its success can be attributed to its scalability. Therefore, DQN has been used for many applications and has proved its effectiveness in multiple fields (Arulkumaran et al. 2017) that include DQN for same-day deliveries (X. Chen et al. 2022), path planning for autonomous surface vehicles (Chai et al. 2022; H. Li et al. 2019; Luis et al. 2021; Yang et al. 2018), and even in optimisation of demand-side management systems for energy consumption (Tai et al. 2022). Among the vast literature, one of the most well-known implementations of DQN is for playing Atari games (Mnih et al. 2015), where DQN-trained policies outperformed humans in most games. The preceding work introduced the fundamental idea of using a main and target neural network to stabilise the agent's learning.

Several researchers explore different applications in the context of combining ideas of the algorithms discussed in section 2.1. For example, deep SARSA is a novel approach that combines SARSA with deep learning techniques to address video game control problems (D. Zhao et al. 2016). By utilising deep convolutional neural networks and an experience replay buffer, similar to DQN, deep SARSA demonstrates superior performance compared to deep DQN for some cases. However, the use of contextual information is also avoided. Some researchers, aware of the limited understanding of DQN, focus on algorithmic and statistical analyses, aiming to establish convergence rates and justify the key components of DQN, such as experience replay and target networks (Fan et al. 2020). Besides that, based on their findings and the concept of two-player zero-sum setting (Littman 1994), the authors propose the Minimax-DQN. This algorithm introduces the concept of a minimax objective, aiming to find a policy that minimises the opponent's reward while maximising the agent's reward. The algorithm trains two separate DQN networks, one for the opponent and the other for the agent. This approach is distinct from DQN because it does not use a main and target neural network.

Another approach that modifies the original QL is asynchronous QL (G. Li et al.

2020), an algorithm that integrates multiple agents that run in parallel. Each agent interacts with the environment independently. To this end, this parallelisation seeks to allow the agents for more efficient exploration. There has also been a modification of asynchronous QL. For example, incorporating the principle of pessimism (Y. Yan et al. 2022). In other words, they penalise the less visited state-action pairs, which encourages a better exploration. Another enhancement to asynchronous QL is a bound proposed in (G. Li et al. 2021), which aims to estimate the number of samples needed for finding a proper Q function. However, there is a problem with this sort of approach because it is not easy to verify if any of the agents is not improving while the others do improve (Casgrain et al. 2022). Therefore, it is necessary to find a Nash equilibria (Bai et al. 2020). In this context, a modification of DDPG called multi-agent DDPG (Lowe et al. 2017), was proposed aiming to train multiple agents that consider each other's progress.

Overall, the discussed works highlight advancements in RL by borrowing ideas of the various state-of-the-art algorithms and following different strategies such as multi-agent architectures or including demonstrations in the replay buffer. The main contribution of these works is to improve the exploration efficiency of RL in a context-free setup.

2.2.2 Implicit Context-based Methods

Several existing approaches use indirect contextual information in the form of previous experiences, intending to enhance their learning capabilities. For example, Transfer Learning (TL) (M. E. Taylor and Stone 2009) is an approach that utilises previous experiences from solving preliminary source tasks to learn a new policy and use it for a new target task. As a consequence of this, TL uses fewer samples than if the policy had been trained from scratch. Given a target task, an RL transfer agent must perform three steps: select a correct source task, find the relation between source and target tasks, and transfer knowledge from source to target task. Within

the existing literature, various approaches use state-of-the-art RL algorithms and combine them with TL. For instance, QL has been combined with TL in Transfer Reinforcement Learning under Unobserved Contextual Information (Y. Zhang and Zavlanos 2020), in which the authors propose to solve a Contextual Markov Decision Process (CMDP) (Hallak et al. 2015) using TL. This is achieved by providing casual bounds and a context-aware policy to assist the agent's learning process. Despite the advantages of TL, there are still several concerns regarding over-fitting one of the tasks and, in this way, difficulty transferring the knowledge to a new task (Weiss et al. 2016).

In RL, sampling adequate data from the training environment is particularly important as it improves the learning process because the agent incrementally updates its parameters while producing its own training set (Deisenroth and Rasmussen 2011). An approach that aims to enhance the sampling efficiency of RL is Deep Q-network from Demonstrations (DQNfD) (Hester et al. 2018). DQNfD is an excellent example of how the quality of the sampled data improves the agent's performance. For example, although DQNfD takes 1 million steps to achieve good scores, DDQN takes 84 to 85 million steps to accomplish a similar performance. The approach creates a data set with human demonstrations to pre-train a neural network. After using the DQNfD algorithm, the agent outperforms DDQN in 41 out of 42 Atari games. However, these algorithms lack the human ability to recognise actions while observing video streams (S. Liu et al. 2022, 2020), and this contextual information is often omitted. This situation leaves the agent with the task of learning it from scratch. Another method that implicitly embeds contextual information is Multi-Modal Contextual Reinforcement Learning (MMCRL) (Kabra et al. 2021b) that proposes using a contextual exploration technique in real-time recommendations for users by including previous experiences to avoid training an agent from scratch and reducing the number of random recommendations. A similar approach is employed in (Kabra et al. 2021a), in which by using RL and contextual resources, an agent

learns to make top-k recommendations [1]. Another strategy that can be employed to overcome the problems of stochastic explorations is Generative Action Selection Through Probability (GRASP) (Xu et al. 2021). GRASP uses generative adversarial networks to generate exploration spaces so that the exploration efficiency improves by indirectly including this contextual information.

An alternative perspective to indirectly include demonstrations into the learning process of RL is Inverse Reinforcement Learning (IRL), a method that allows inferring a reward function from an observed behaviour (Arora and Doshi 2021). IRL aims to avoid the difficulties that involve the manual definition of reward functions in RL (Ng, Russell, et al. 2000; Russell 1998). Maximum entropy IRL (Ziebart et al. 2008) is the most popular method to infer the reward function. This method involves finding the reward function that maximises the entropy of the policy subject to matching the learned policy with the demonstrated one. Nevertheless, one of the most common problems with IRL is the lack of generalisation to new scenarios (Adams et al. 2022). This is because the demonstration may not be optimal, and the reward function limits the agent to explore better alternatives. Under other conditions, context can be indirectly included in the agent's learning process, which does not depend on the reward function. Hence, the agent is not constrained to the quality of the demonstration. RL algorithms can be equipped with Recurrent Neural Networks (RNNs) (X. Li et al. 2015). Since RNNs are based on the Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber 1997) architecture, they use the previous output as part of the input. Hence, since it is possible to store past information, it can be said that RNNs feed the agent with indirect contextual data.

Summarising, these approaches demonstrate the impact of adding context in RL in an implicit manner. While there are advantages, such as avoiding the tedious definition of reward functions, challenges, such as lack of generalisation, overfitting, and the manual generation of demonstrations, still remain significant issues.

[1] Top-k recommendations refer to a method of generating a ranked list of the k most relevant items.

2.2.3 Explicit Context-based Methods

Several methods exist in the literature that include explicit contextual information by encoding it into the state or using affordances. An affordance is a semantic link between the environment and the possibility of taking an action (Gibson 1977). For example, a hammer affords to hit, while a pen affords to write. Affordances have been applied successfully to different fields (Koppula and Saxena 2015; Yamanobe et al. 2017), and even there has been a considerable amount of research that involves how to learn affordances (Khetarpal et al. 2020; Wu et al. 2020). Despite this, the focus of the research on affordances is mainly on how to extract them from the environment and, in some cases, combine them with planning methods (i.e. Şahin et al. (2007)) that do not involve RL. As a case in point, AffordanceNet (Do et al. 2018) is a deep learning approach that detects multiple objects and their affordances with only the images as input. This approach allows object detection and pixel assignation of labels representing the affordances.

On the other hand, Chalmers et al. (2017) present a method based on affordances to predict a human's next action such that a robot reacts accordingly, in which an example of a successful implementation of affordances in an RL agent is illustrated. Moreover, the method not only reaches any environment where its predictions are valid but also accelerates RL in new scenarios. Additionally, affordances have been used as constraints. For instance, Cruz et al. (2016a) introduced a resource called "contextual affordances" that works as a constraint of high-level actions depending on the elements present in the environment. This method relies on a table where the affordances are defined, and during the learning process, the actions are sampled, taking the table into account. Eventually, the authors proved that incorporating affordances accelerates the learning process of the RL agent used in their experiments. A similar approach is followed in (Cruz et al. 2018a), where the authors escalated the use of affordances, vocal commands and hand gestures to enhance multi-modal RL learning performance in a robot cleaning scenario.

Nevertheless, another way to approach the direct involvement of contextual information is through frameworks designed to solve CMDPs (Benjamins et al. 2021, 2022). The framework incorporates information such as gravity, target distance, actuator strength, and joint stiffness in the learning process. This approach demonstrates how these contextual data affect the performance learning of the agents. Another approach is to encode the state and the contextual features (Sodhani et al. 2021). The method developed relies on the capacity to relate tasks to external supplementary data to improve the agent's learning performance. Injecting domain knowledge into the neural network (Teng et al. 2014) is another method that proves contextual data's positive influence on an agent's learning performance. However, all the methods mentioned above still fail to include the affordances of the actions, which is also essential contextual information.

In general, affordances have great potential for enhancing the exploration efficiency of RL agents (Y. Yu 2018). In the same way, methods that involve encoding the state have also shown the potential to improve learning performance in RL (A. Zhang et al. 2020). Despite the vast number of research articles focusing on affordances and observation encoding, the methods utilised vary, and this situation makes it challenging to implement a universal solution that captures the benefits of using affordances and encoding the observation of the environment.

2.3 Reinforcement Learning in Robotics

This section introduces relevant research in the field of robotics and RL, where classical and RL approaches are applied to solve Human-Robot Interaction (HRI) problems and manipulation tasks of rigid and deformable objects.

2.3.1 Why Reinforcement Learning in Robotics?

Based on RL approaches, many researchers have worked towards improving the autonomy of robots for several applications, including but not limited to motion planning (X. Li et al. 2021; S. Wen et al. 2018), obstacle avoidance (Kontoudis and Vamvoudakis 2019), wel2ding (Zhong et al. 2021), robot manipulation (D. Liu et al. 2020), robot-assisted rehabilitation (Y. Zhang et al. 2019), and dual-arm motion planning (Wong et al. 2021). In these works, classical methods, including Rapid-exploration Random Trees (RRT) (LaValle et al. 1998) and variants of it like RRT* (Karaman and Frazzoli 2010), are combined with RL approaches such as Actor-Critic architectures (Mnih et al. 2016), and DDPG. This combination enhances the performance and quality of the robot's navigation and motion planning capabilities.

One of the most significant advantages of RL in robotics is its capacity to learn and adapt (Polydoros and Nalpantidis 2017). Unlike classical methods (kinematic control, inverse kinematics, path planning, control theory, computer vision and state estimation), which rely on static programming, mathematical models and predetermined rules, robots utilising RL algorithms can continue learning and evolving in almost any environment. By directly learning from interactions and experiences, robots using RL approaches can adjust their behaviour based on the reward received so that they keep enhancing their performance even when the models of the environment and the robot are not available (Kormushev et al. 2013).

RL algorithms are designed for handling complex and dynamic environments, such as the ones that robots usually face. Hence, it is essential to recognise that a robot must also be re-programmed when the environment changes (K. Zhao et al. 2022). For this reason, classical planning methods may struggle in scenarios with large and continuously changing states and action spaces because of their reliability on predefined action sequences or heuristics. RL agents, on the other hand, can adapt their strategies to handle uncertainties, making them more suitable for such

environments.

In terms of perception, classical methods based on computer vision rely on static programming and are usually focused on extracting a limited number of features (e.g., detecting motion). For example, H.-Y. Lin et al. (2020) propose a framework that utilises the objects' Computer-Aided Design (CAD) models to match with the point cloud produced by a depth camera to find the best grasping pose for an object. Nonetheless, the generalisation of the previously mentioned approach would involve having a CAD model of every object the robot needs to grasp. On the other hand, RL-based methods commonly rely on the camera's stream data only, which could be RGB images or a point cloud. Cheng et al. (2022) and Cheng and Meng (2018) use Convolutional Neural Networks (CNNs) for predicting the grasping rotations and locations of several objects. A drawback of this method is that large datasets are required to achieve a decent success rate. There also exist alternatives to RL, such as the one Zeng et al. (2022) proposed, in which by using genetic algorithms, the utilisation of any datasets is avoided.

Despite the advantages of RL-based approaches, there exist some drawbacks, such as the long training time, the necessity of simulated environments, the reward function design being complex for most of the tasks, and the challenging task of transferring the knowledge from simulation to real-world puts extra difficulty into the generalisation of RL approaches. Besides that, the presence of humans and their interaction with robots, as well as the potential of using contextual information as part of the agent's training, are factors which are often omitted. Table 2.4 summarises the various advantages and disadvantages of RL methods applied to robotics. The following subsections discuss several works that aim to address these drawbacks from distinct perspectives.

Advantages	Disadvantages
• Model-free framework. • Handle complex dynamic environments. • Learn and adapt beyond classical methods.	• Long training time. • Necessity of simulated environments. • Complex reward function design. • Sim-to-real challenge.

Table 2.4: Advantages and disadvantages of RL methods applied to robotics.

2.3.2 Robotic Manipulation of Rigid Objects with Reinforcement Learning

One of the most popular applications of RL in robotics is the manipulation of rigid objects. Among RL methods, QL has been a popular algorithm that allows robots to learn and perform manipulation tasks (Jain et al. 2018). When hybridised with secondary optimisation algorithms, it is also known that the QL algorithm achieves better results (Konar et al. 2013; Maoudj and Hentout 2020). In RL, secondary optimisation algorithms modify the principal optimisation algorithm. Some approaches have employed secondary optimisation algorithms in QL agents to set initial optimistic values, and in robotics in several works (S. Li et al. 2015; C. Yan and Xiang 2018), with promising results. For instance, integrating a novel flower pollination algorithm with QL to initialise the Q-table and selection of control parameters accelerates the learning process of the traditional firefly algorithm (Low et al. 2019). This approach establishes a balance between the exploration and exploitation of the computational agent during the learning process. Splitting the task into a hierarchy of small parts is proved to be an effective method in QL (Veeramani and Muthuswamy 2022). Ji et al. (2019) alluded to a novel QL-based approach that efficiently computes the path of the robot arm based on a hybrid path planning method, which splits the planning problem into two separate parts: active finding (finds simple actions for the robot arm) and passive finding (computes joint angles).

Hand-eye coordination is the ability to coordinate the movements of the eyes and the hands to perform actions or tasks. In robotics, there are approaches to developing this ability based on RL. For example, CNNs can be trained to predict which gripper's motion will result in a successful grasp in the context of the hand-eye coordination problem (Levine et al. 2018). Besides being implemented in the real world (see Fig. 2.3), the approach only uses camera images and the current robot pose. Contrary, Mahler et al. (2017) utilises synthetic data and a depth camera instead of a monocular one to feed the CNNs. Their experiments demonstrated that their approach could find the right grasping pose in less than a second. Another famous approach is the one presented by O. M. Andrychowicz et al. (2020), where RL is utilised to learn dexterous in-hand manipulation policies. First, the training is performed in a simulated environment, and then the knowledge is transferred to a real robot. The authors employ a variety of technical manoeuvres to facilitate the learning and transfer of knowledge to the real robot, such as objects easy to identify and cameras at the fingertips. Contrary to relying on simulations, H. Zhu et al. (2019) propose a simple system framework that allows learning to open the door with a flexible handle, rotating a cross-shaped valve, rotating the same valve made of deformable foam, and box flipping. They show that their approach, based on DRL, learned to perform the tasks in 4-7 hours, and when integrating demonstrations, the time was reduced to 2-3 hours.

Moreover, Quillen et al. (2018) evaluates RL algorithms for vision-based robotic grasping. The results show that DDPG is more reliable than MC methods and DDQN. Another approach that involves a massive quantity of training data is in (Pinto and Gupta 2016), where the results show the benefit of using large-scale datasets by training a robot to learn to grasp from 50,000 tries and 700 robot hours. However, when dealing with tasks, these approaches are often criticised due to their huge data requirements. Aiming to address this concern, Pinto and Gupta (2017) proposed using shared representations of tasks. Besides, it showed that adding

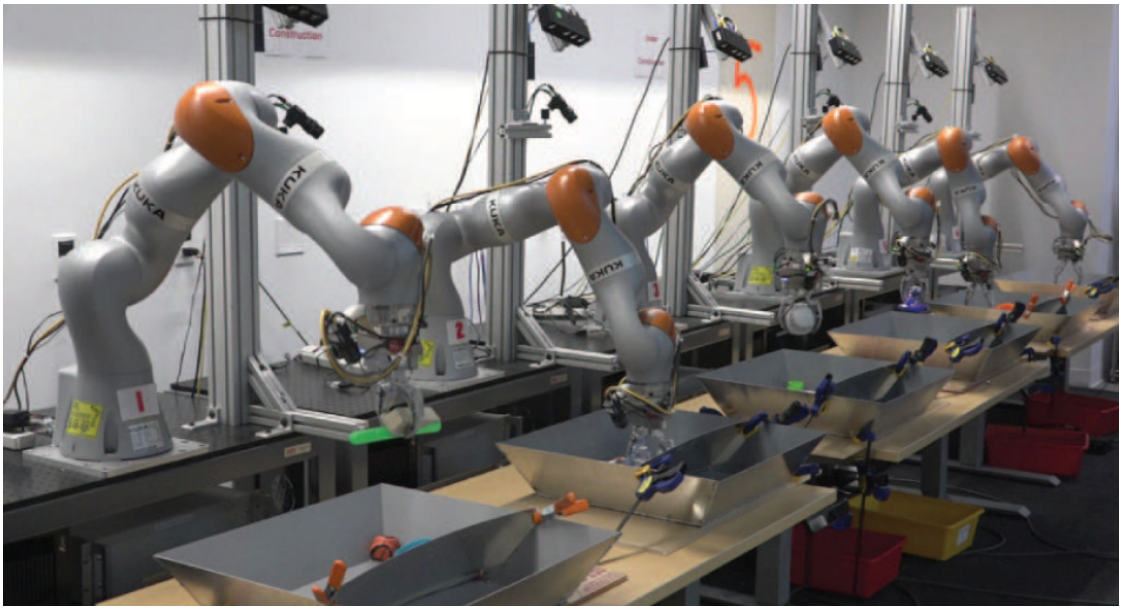


Figure 2.3: Collection of data with several robots. This approach is designed to learn in the real world (Levine et al. 2018). Despite being a relevant approach in the literature, not many laboratories in the world can afford that number of robots.

additional data from different tasks is more important than adding data from the main task. In other words, by learning to push and poke, the agent learns faster to grasp. Levine et al. (2016) proposed a method for training policies that directly map raw image observations to robot motor torques using CNNs. To achieve this, a guided policy search method is employed, which transforms the policy search into supervised learning. The supervision is provided by a RL algorithm. The approach is tested on several real-world manipulation tasks that involve rigid objects, such as screwing a cap onto a bottle.

Overall, vast literature exists in RL-based robotic manipulation of rigid objects. Notoriously, the discussed approaches show potential in dealing with difficult problems, from finding the right grasping points of an object to dexterous manipulation using only one hand. Finally, learning in the real world and simulation has been widely explored, and contextual information, such as demonstrations, has effectively reduced training times.

2.3.3 Reinforcement Learning in Human-Robot Interaction

The preceding subsection discussed works in which the objects the robot is learning to manipulate are static, or their starting position is on the robot's hand. In contrast, in HRI, an area that studies how humans and robots communicate and collaborate (Sheridan 2016), human users introduce the challenge of dynamic and stochastic behaviours. Research in the intersection of RL and HRI have led to significant progress in addressing challenges related to robotics. A vast part of the literature focuses on learning from users in HRI setups. A proposed solution methodology to address the problem is using affordances. When implemented in MDP, the affordances make the agent choose optimal actions sooner, dramatically reducing the number of state-action pairs the robot needs to evaluate (Zeng et al. 2018). Affordances improve the planning, control, recognition, transferability, and programming style of robots (C. Chen et al. 2015). Techniques such as affordances and modifications to the RL algorithms presented in section 2.1 have been proposed to enhance understanding, cooperation, and coordination between humans and robots. The incorporation of human feedback (Fig. 2.4), usually carried out through demonstrations or spoken instructions, has proven effective in improving the learning performance of the robot. These advancements demonstrate the importance of exploring methods of HRI that involve RL approaches and affordances.

In the context of RL and HRI, the use of affordances has been studied to solve problems, such as cleaning tables (Cruz et al. 2016b), reacting to non-verbal user's clues (Khamassi et al. 2018), identifying users' behaviours (Tabrez and Hayes 2019), coordinating human and robot actions (Ghadirzadeh et al. 2020; Roy et al. 2019), and learning from the user (Dromnelle et al. 2020; Lowe et al. 2019). Besides that, facilitating fluent and effective communication and understanding in HRI from both parts emphasise the need for cooperation and coordination of robots operating in human-populated environments (Luebbers et al. 2022; Tabrez et al. 2020). Research has been carried out to address this challenge. For example, Zakershahrak et

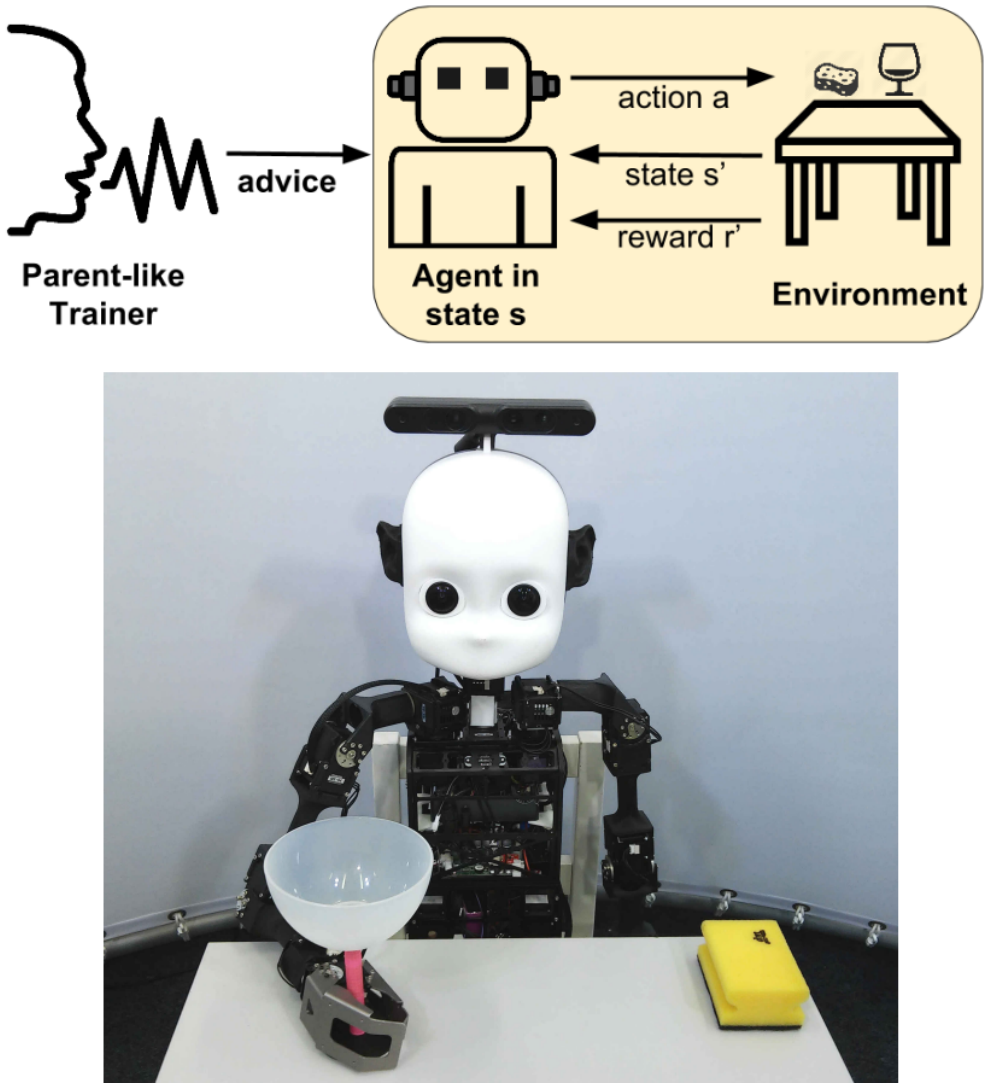


Figure 2.4: Cruz et al. (2016b) proposes a framework in which a user guides the learning of a robot through spoken instructions.

al. (Zakershahrak et al. 2021) address the issue of understanding explanations from the human recipient in a planning context. This is achieved by formulating the problem as a goal-based MDP, and the reward function is learned with IRL. Another approach based on DRL is proposed in (Shafti et al. 2020), in which through the use of a collaborative maze game where co-learning between the human and robot is necessary to complete the task, the human and robot actions require joint effort to solve. The authors showed that it is possible to find an effective collaborative policy that leads to consistent success in the game setup proposed in the paper.

Moreover, several researchers have explored the intersection of RL and HRI in

dynamic and stochastic scenarios. By way of illustration, Z. Liu et al. (2021) focuses on HRI in dynamic and stochastic environments, more specifically in collaborative assembly. The authors proposed a modification of DQN, aiming to achieve successful collaboration. The study demonstrates the effectiveness of training the robot by reacting correctly to the random behaviour of the human. Another task-specific solution for onion sorting is proposed in (Sengadu Suresh et al. 2023). The work introduces a method based on IRL that allows learning in collaborative tasks with a human partner. The main novelty of the approach is that it avoids agents needing to access the global state. Furthermore, the agent achieves superior performance in the onion sorting task, proving that IRL has the potential to deal with problems in HRI setups. An additional technique that relies on IRL is the one proposed by Nikolaidis et al. (2015), in which the torque data produced by the demonstration of a user is utilised for training a model that can adapt faster to new users. On the other hand, human feedback can also improve RL. As a case in point, Thomaz and Breazeal (2006) propose a modification of QL called interactive QL, which incorporates human actions into the classical algorithm. In their experiments, the agent that incorporates human feedback improves in terms of exploration and speed of task learning.

To summarise, RL has been used to address various HRI tasks. However, when dealing with challenging environments involving users, the solutions and modification of the baseline RL algorithms become task-specific. This sheds light on generalisation problems due to the difficulty that HRI involve. Furthermore, the tasks primarily centre around manipulating rigid objects, emphasising that knowledge transfer and algorithm design for robot learning heavily rely on perceptual abilities, which are more straightforward for rigid objects. On the other hand, human feedback in the form of affordances or demonstrations has proved to be an effective tool for enhancing learning performance. This shows the impact of involving contextual information in the learning process of RL agents.

2.3.4 Robotic Manipulation of Deformable Objects with Reinforcement Learning

Robots with human-level dexterity that can handle deformable objects may encourage a smoother integration of robots in daily activities. In practice, daily activities depend on more than manipulating rigid objects. In this context, the robots' capacity to manipulate deformable objects to operate in human environments is a necessity (J. Zhu et al. 2022). In the literature, several approaches mainly focus on how to manipulate 2D deformable objects, such as paper (Balkcom and Mason 2004, 2008; Elbrechter et al. 2011, 2012), fabrics (Borràs et al. 2020; Hoque et al. 2022; Jangir et al. 2020), ropes (Nair et al. 2017; Shi et al. 2022; Sundaresan et al. 2020) and cables (Zhou et al. 2020; J. Zhu et al. 2019). The cited approaches rely on classical approaches where the model of the robot and the object are necessary to complete the task. The model-free capabilities of RL, on the other hand, have the potential to learn how to manipulate deformable objects without the necessity of any model.

One way to tackle the challenge of handling deformable objects with RL and robots is by using simulated environments. These are valuable resources for training agents to learn how to solve a given task. For example, Seita et al. (Seita et al. 2021) used transporter networks to learn how to rearrange deformable objects, where 3D deformable structures such as bags are included. In (Bahety et al. 2022), a method to rearrange objects is proposed and is based on two policies learned in simulation. The first policy rearranges the objects, and the second one learns to lift them. A disadvantage of this approach is that it assumes that the bag is always open. Therefore, in a simulated environment, all the information regarding the opening of the bag and the objects is available, which is useful when training an agent requires a large number of episodes to learn the task. However, when running the agent in the real world, the differences between the simulation and the real-world tasks may lead to undesirable and dangerous behaviours.

In the literature, methods exist to transfer the knowledge obtained during simulation to the real world (sim-to-real). X. Ma et al. (2022) introduce a method that sets several grasping points on a cloth surface. A graph neural network uses these points to learn their dynamics. Subsequently, when the task is transferred to the real world, it is easier to track the points than the whole cloth. X. Wang et al. (2022a) introduced a method in which an agent is trained to learn how to wrap boxes in a simulated environment. The actual texture of the deformable object is taken from the real object such that the transition from simulation to the real world is smoother than when taken from the pure simulation.

To summarise, all the current research on deformable objects mainly focuses on manipulating 2D objects. Furthermore, the methods used to deal with the sim-to-real gap vary among the different works (Fig. 2.5). This sheds light on the difficulty of manipulating deformable objects and how these rely on robust perception methods.

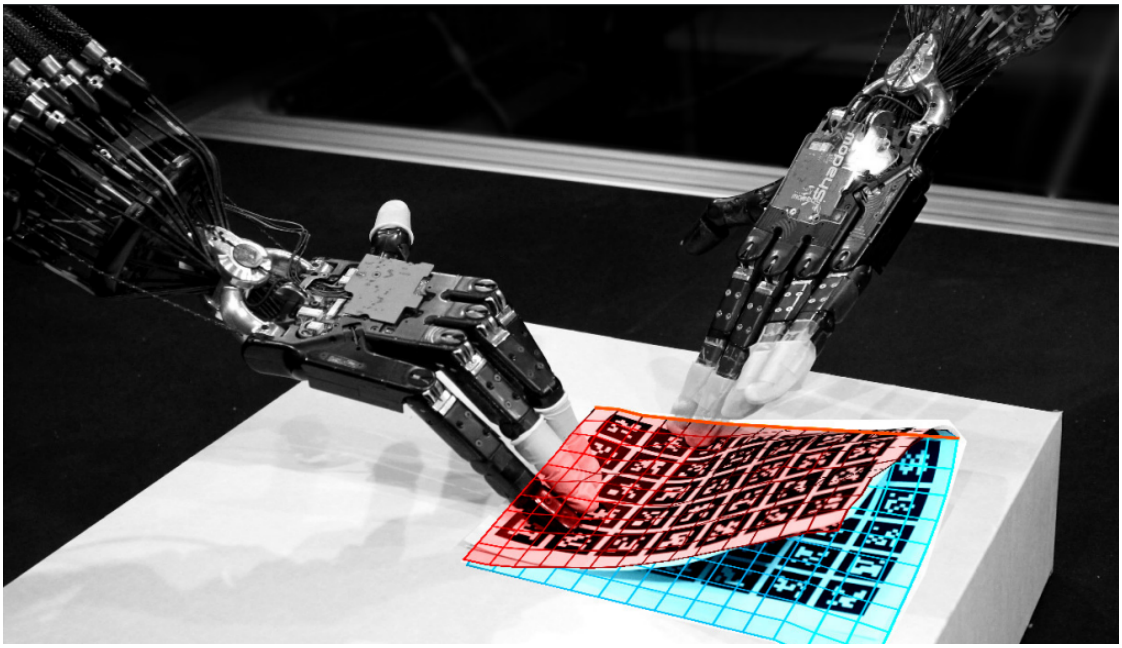


Figure 2.5: Two dexterous robotic hands manipulating paper (Elbrechter et al. 2012). This is a good example of how adding markers to the sheet of paper is necessary to extract its current state.

2.3.5 Robotic Manipulation of Deformable and Rigid Objects with Reinforcement Learning

In the context of manipulation of deformable and rigid objects, *bagging* shines for being one of the most widely human tasks that is not only necessary but also carried out every day and everywhere. As a case in point, Iterative Interactive Modelling for Knotting Plastic Bags (C. Gao et al. 2023) is an approach that focuses on the bag's handles, learns from demonstrations, and uses a set of primitive actions to knot the handles. However, the detection of the handles relies on a large dataset. Another interesting solution is the one presented by L. Y. Chen et al. (2022). The authors introduced the algorithm AutoBag, which is based on several primitive actions that reorientate plastic bags till the opening is visible and the objects can be put inside.

Gu et al. (2024) introduce ShakingBot (Fig. 2.6), which uses a perception module to identify the key region of a plastic bag and opens it using Bag Adjustment, Dual-arm Shaking, and One-arm Holding. The robot then inserts items and lifts the bag for transport. An interesting example to consider is the use of Singulating Layers using Interactive Perception (SLIP) in the task of autonomous bagging (L. Y. Chen et al. 2023). The authors developed an algorithm called SLIP-Bagging that makes use of SLIP to grasp the top layer of a plastic or fabric bag and open it for object insertion. In their physical experiments, a YuMi robot using the SLIP-Bagging algorithm achieved a success rate varying from 67% to 81% across bags of different materials.

Overall, the field of deformable object manipulation has not been explored as extensively as the field of rigid object manipulation, and the research becomes even more narrow when dealing with 3D objects such as bags. This is because of the difficulty involved in understanding the dynamics of the deformable object. RL, due to its learning nature, has the potential to use its model-free capabilities to indirectly learn the model of the objects and perform deformable object-oriented tasks.

Figure 2.6: ShakingBot using the physical capacities of the bag in its favour to open it in three steps (Gu et al. 2024). First the robot locates the bag handles and then uses the material's properties to maintain its shape in order to complete the task.

2.4 Discussion

This section delves into a detailed analysis of the advantages and disadvantages of various RL approaches and their implementation in robotics, which were presented during the previous sections. After carefully examining the discussed works, it becomes evident that significant progress in RL has been made by combining ideas from state-of-the-art algorithms presented in section 2.1 and applying them to solve a wide variety of problems in which robotic is one of the most benefited fields. Current research adopts strategies such as multi-agent architectures and classical robotics methods involvement. Another way to improve the learning efficiency of RL are the ones that implicitly or explicitly involve contextual information in the learning process in the form of demonstrations or through the use of affordances. Similarly, methods involving the encoding of state representations have demonstrated improved learning performance in RL. By encoding contextual information within the state, agents can better comprehend the environment and adapt their actions accordingly. These approaches primarily contribute to improving the exploration efficiency of RL in a context-free setup.

Additionally, current research endeavours have been made to explore the implementation of RL in robotics. It has been found that using affordances can greatly enhance a robot's exploration efficiency. Affordances help robots extract relevant information from their environment, enabling them to identify action possibilities and make decisions based on them. However, despite the proliferation of research articles on affordances and observation encoding, there is still no universal solution that captures all the benefits of these techniques. In other words, by encoding the state, all the semantics utilised for that purpose also have the potential to be used as a source of affordances. Besides that, the intersection between RL and HRI remains challenging to the point that the research carried out proposes task-specific solutions. This shows that the sole RL framework is not enough to provide a universal solution to all problems.

Despite the advantages of the discussed approaches, there are still challenges related to the implementation of RL in robotics. Firstly is the challenge of generalisation, wherein RL agents struggle to perform properly in unseen scenarios. Overfitting poses another obstacle, which often results in poor performance. Secondly, the manual generation of demonstrations remains a concern, as it is a time-consuming task that requires effort and expertise. Thirdly, when the exploration of the environment is performed stochastically, the only way to get feedback is through the reward function, and contextual information that can be used to make decisions about what to explore is often omitted. Moreover, the design of reward functions is usually carried out manually, or for some tasks, the reward is sparse, meaning that if the definition of the reward is simple, it complicates the agents' exploration process. Lastly, based on the literature review carried out in this chapter, the following gaps were identified:

- Lack of involvement in the whole learning process of RL of explicit contextual information. Some research focuses on using only affordances to improve exploration efficiency. At the same time, other research focuses on encod-

ing the whole information of the state. However, the equations introduced in section 2.1 are rarely modified to directly benefit from such modifications.

- The use of RL for HRI focuses more on improving the learning performance of the agent with human demonstrations and instructions and less on reacting in real-time to dynamic human behaviour. There are potential applications of the RL policy that can be combined with contextual information, such as semantics, that can be used to make decisions based on what the policy finds in its way to reach the goal.

- Lack of approaches designed to learn directly in the real world. Most of the research in RL utilises simulations as a source of data and avoids learning in the real world. This is because the baseline algorithms require a large number of training steps and dangerous behaviours that may be unsafe to perform with real robots. However, using simulations also brings some challenges with it. For example, transferring the knowledge gained in the simulator to the real robot often reduces the performance of the agent due to differences between reality and simulation. Hence, there is a necessity for designing algorithms that allow efficient learning in the real world in a safe manner.

- Current research in robotic manipulation focuses more on how to handle rigid objects than on how to handle deformable objects. At the same time, among deformable objects, there has been limited exploration in the area of learning to manipulate 3D objects such as bags.

2.5 Summary

This chapter has provided valuable insights into the implementation of RL in robotics. The main findings are discussed as follows. Firstly, significant progress has been made in RL and RL for robotics. Achieving this was possible by combining ideas

from state-of-the-art algorithms and adopting strategies such as multi-agent architectures, the inclusion of contextual information in the form of affordances and techniques to encode extra information into the state. These approaches have shown promise in improving the exploration efficiency and accelerating the learning process of RL. Secondly, RL has shown the potential to deal with dynamic and stochastic real-world scenarios such as the ones that involve HRI. Lastly, in the context of deformable objects, several approaches have been proposed to learn in simulated environments and have proved to be reliable. However, challenges remain, including difficulties in generalisation, perception, large dataset dependency, long training time, simulations dependency and sim-2-real knowledge transfer. This thesis focuses on proposing different RL algorithms that can provide robustness in learning in the real world and reduce the learning time by incorporating contextual information in the learning process.

This literature review has also identified several gaps in the current knowledge of RL. First, there is a lack of explicit involvement of contextual information throughout the RL learning process. Current research primarily focuses on either affordances or state encoding rather than effectively integrating both approaches and incorporating them directly into the objective or loss functions. Additionally, the intersection between RL and HRI requires further exploration. While there are several resources that can be derived from the context to enhance the robot's reactions to dynamic human behaviour, how to effectively utilise these resources within the RL framework remains unclear. The majority of the literature mainly concentrates on task-specific cases, making it difficult to apply the same approach to different tasks, which contradicts the original aim of RL as a universal framework that allows learning in a human-like manner. Furthermore, there is a need for algorithms that facilitate efficient learning directly in the real world. Most current RL approaches heavily rely on simulation, and the accuracy of these simulations presents challenges in transferring knowledge to real robots due to performance discrepancies. Lastly,

research on RL primarily focuses on interacting with rigid objects, even in the works discussed in section 2.3.3, where the tasks involving HRI mainly focus on rigid objects. Additionally, manipulation of 2D deformable objects, such as cables, cloths, or paper, receives more attention, while manipulation of 3D objects like bags remains underdeveloped.

This thesis follows an incremental approach to address these gaps. Firstly, the incorporation of contextual information and state encoding into RL algorithms is explored in simulated environments. Secondly, aiming to extend the applicability of these principles to real-world scenarios, a hybrid approach is proposed that integrates HRI observations and encoding within the framework, aiming to allow the robot to learn and react based on an RL approach. Finally, the underexplored domain of RL in deformable object manipulation is investigated, focusing on techniques that involve RL, state encoding, and contextual information, where the goal is to enable robots to learn how to manipulate bags in the real world. By addressing these gaps, this thesis aims to contribute to the field of RL and robotics by proposing novel frameworks that integrate contextual information, such as affordances and state encoding, throughout the RL learning process.

Chapter 3

Explicit Context Representation in Deep Reinforcement Learning

This chapter presents a framework that aims to smoothly integrate contextual data in Deep Reinforcement Learning (DRL) agents for solving discrete environments, in this case, 2D games. The rest of the chapter is organised as follows: Section 3.1 motivates and discusses current problems with state-of-the-art approaches. Followed by Section 3.2 that formally introduces the framework and details its use to generate four new algorithms. Then, Section 3.3 introduces the characteristics of the environments and neural networks and the evaluation metrics employed. Section 3.4 describes the results. Section 3.5 presents a discussion. Finally, the main findings of this chapter are summarised in Section 3.6.

3.1 Introduction

During the past few years, Reinforcement Learning (RL), which is a sub-field of Machine Learning (ML), has successfully shown to be capable of learning policies to control decision-making in sequential environments (Hanna et al. 2021b; Schaul et al. 2015; X. Wang et al. 2022b). QL is an off-policy RL approach that updates the action selection given a particular state using Bellman optimal equations based on the

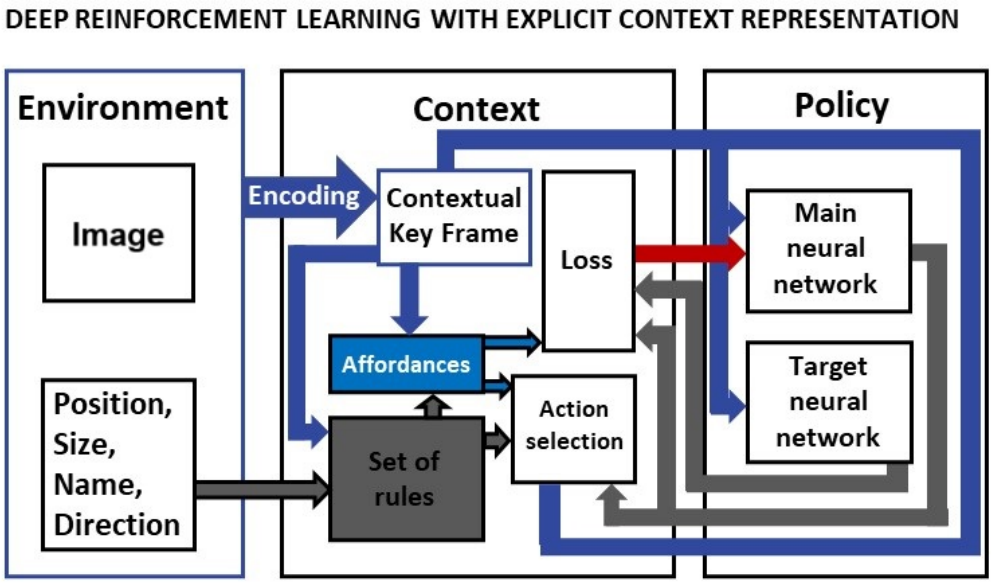


Figure 3.1: Deep reinforcement learning with explicit context representation.

Temporal Difference (TD) principle. TD combines the Monte Carlo (MC) methods and Dynamic Programming (DP) introduced by Watkins and Dayan (1992a).

This chapter introduces the Iota explicit context representation (IECR) framework, which aims to enhance the learning performance of RL agents by adding contextual data (Fig. 3.1). IECR initially transforms the environment information into Contextual Key Frames (CKFs), comprising a matrix in which each cell contains a token representing an element from the environment, its position, size, and direction. The main contributions are summarised as follows:

- IECR, a framework that uses contextual data to improve the learning process of RL agents, is introduced.

- Four new algorithms based on IECR: Iota Deep Q-network (IDQN), Iota Double Deep Q-network (IDDQN), Iota Dueling Deep Q-network (IDuDQN), and Iota Dueling Double Deep Q-network (IDDDQN).

In this chapter, two stages of experiments are performed: (i) the first stage aims to demonstrate the effect of the CKFs representation on the performance of IDQN

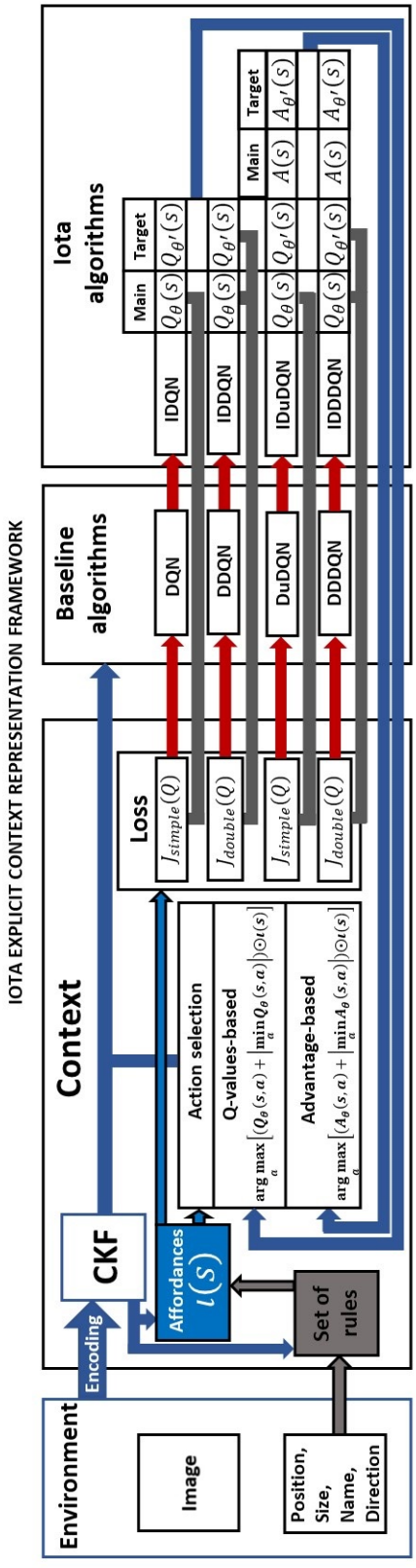


Figure 3.2: In the figure, the iota explicit context representation framework is applied to Deep Q-network (DQN), Double Deep Q-network (DDQN), Dueling Deep Q-network (DuDQN), and Double Dueling Deep Q-network (DDDQN) to create four new algorithms that learn with context. The affordances function $\iota(s)$ is connected with the whole framework, hence, the name of the framework.

learning, and (ii) the second stage evaluates the proposed framework and state-of-the-art algorithms' performance under the same conditions. For both stages, the stable-baselines (Hill et al. 2018) implementations are used. The problem domain to benchmark the algorithms comprises five discrete environments, and their characteristics are explained in detail in Section 3.3.

3.2 IECR Framework

In this section, the IECR framework (Fig. 3.2) is presented, aiming to allow the smooth integration of contextual information into the learning process of DRL agents based on DQN, DDQN, DuDQN and DDDQN. For each variant of DQN, the IECR framework uses three algorithms, detailed in the next three subsections: CKFs' generation, affordances function generation and learning. The nomenclature of this chapter is summarised in Table 3.1.

3.2.1 Contextual Key Frames

This subsection introduces Algorithm 1, which focuses on identifying known semantic data from the environment (objects, positions, sizes, and directions) and using it as a feature representation of the state. The contextual key frames algorithm takes as an input the semantic set F, the width of the screen sw, and its height sh. The output is a CKF that contains the tokens of all the elements in the environment. At this point, the goal is to create a set of tokens Z that represent an n_e number of elements in the environment, such that Algorithm 1 can transform it into CKFs. The set of tokens is given by the following:

$$Z = \{\zeta_i \,|\, i \in [0, n_e)\}, \tag{3.1}$$

Symbol	Description
CKF	Contextual Key Frame
F	Semantic set
P	Positions set
W	Sizes set
V	Directions set
A	Actions set
N	Set of tuples representing the pair name of the element with its respective key
Ψ	Rules set
ζ_i	Token of the i-th element
k_i	Key of each type of the i-th element
a_i	Vertical numerical position of the i-th element
b_i	Horizontal numerical position of the i-th element
d_i	Direction of the i-th element
μ	Token range value
n_e	Total number of elements in the environment
x_i	Horizontal position of the i-th element in pixels
y_i	Vertical position of the i-th element in pixels
u_i	Number of rows that of the i-th element occupies taking its own size as reference
v_i	Number of columns that the i-th element occupies taking its own size as reference.
sw	Width of the screen in pixels
sh	Height of the screen in pixels
w_i	Width of the i-th element in pixels
h_i	Height of the i-th element in pixels
w_i	Width the i-th element in pixels with respect to the first element
h_i	Height of the i-th element in pixels with respect to the first element
$\iota(s)$	Affordances function
$J_{\iota n}(\theta)$	Main simple neural network loss
$J_{affordance}(\theta)$	Affordances loss
$J_{simple}(\theta)$	Total simple loss
$J_{\iota nd}(\theta)$	Main double neural network loss
$J_{double}(\theta)$	Total double loss
Max_{IECR}	Maximum reward obtained by an *iota* algorithm
$Max_{baseline}$	Maximum reward obtained by a baseline algorithm
$P_{improvement}$	Percentage of improvement between an *iota* and a baseline algorithms

Table 3.1: Nomenclature table for Chapter 3.

where ζ_i is the token of the i-th element. Formally, a CKF is a $n \times m$ matrix where $\text{CKF}_{n,m} \in Z$. In order to obtain Z, first, extracting the information from the environment is necessary. For this purpose, the characteristics of every component in the environment are known and tokenised so that the agent can easily understand them. The token ζ_i is given by:

$$\zeta_i = k_i + a_i + b_i + d_i, \tag{3.2}$$

where k_i is the key of each type of element in the environment, a_i is the vertical numerical position, b_i is the horizontal numerical position, and d_i is the direction of the i-th element, respectively. Let:

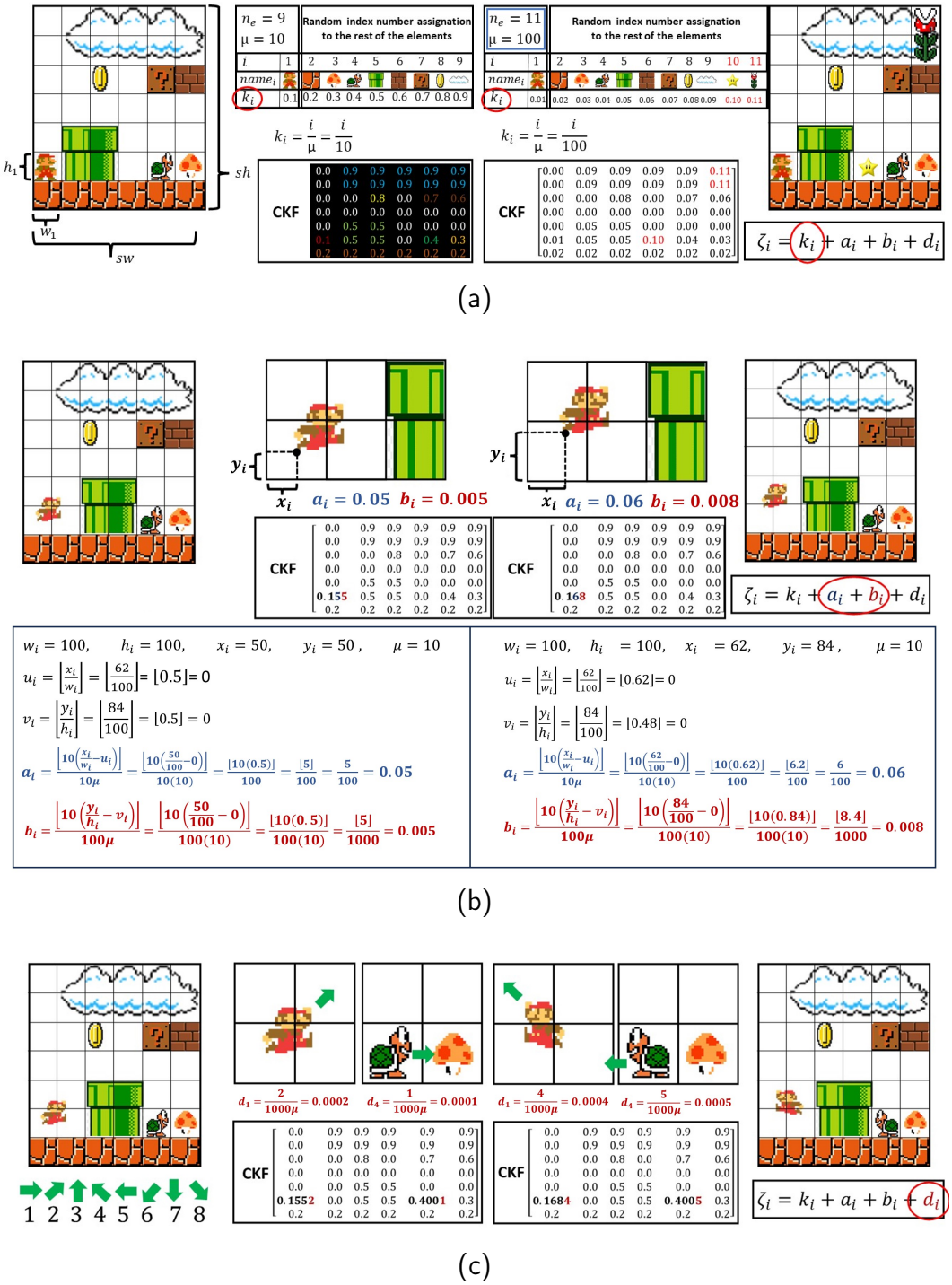


Figure 3.3: In (a), there is an example of tokenising the state according to the element type and the number of elements. In (b), the position of each element in the cell of the CKF is added to the token value. In (c), the direction of the elements and their numerical values are added to the token.

$$N = \{\langle name_i, k_i \rangle \mid i \in \mathbb{Z}^+, k_i = \frac{i}{\mu}\}, \tag{3.3}$$

be the set of tuples representing the pair name of the element with its respective key k_i. Set N represents the relation between the element name and its key. For example, $\langle mario, 0.1\rangle$, $\langle pipe, 0.3\rangle$, $\langle hole, 0.8\rangle$. Let:

$$\begin{aligned} W = \{\langle w_i, h_i, \dot{w}_i, \dot{h_i}\rangle \,|\, i \in [0, n_e), w \in [0, sw], h \in [0, sh], \\ (i = 1 \rightarrow \dot{w}_i, \dot{h_i}) \wedge (i > 1 \rightarrow \dot{w}_i = \lceil \frac{w_i}{w_1} \rceil, \dot{h_i} = \lceil \frac{h_i}{h_1} \rceil\}, \end{aligned} \tag{3.4}$$

be the set that represents the sizes of all the elements, where w_i is the width of the element in pixels, h_i is the height of the element in pixels, $\dot{w}_i$ is the width that is taken as a reference of the size of the main element of the environment, and $\dot{h_i}$ is the height of the i-th element taking the size of the main element of the environment as a reference. Set W contains the information that represents the environment elements' sizes. Let:

$$\begin{aligned} P = \{\langle x_i, y_i, u_i, v_i, a_i, b_i\rangle \,|\, i \in [0, n_e), x_i, y_i \in \mathbb{W}, \\ u_i = \lfloor \frac{x_i}{w_i} \rfloor, v_i = \lfloor \frac{y_i}{h_i} \rfloor, u_i, v_i \in \mathbb{R}^+, \\ a_i = \frac{\lfloor 10 \cdot (\frac{sw}{x_i} - u_i) \rfloor}{10\mu}, b_i = \frac{\lfloor 10 \cdot (\frac{y_i}{h_i} - v_i) \rfloor}{100\mu}\}, \end{aligned} \tag{3.5}$$

be the set containing the position in the image of each element of the environment where x_i and y_i are the horizontal and vertical positions of the agent in pixels, u_i is the number of rows, v_i the number of columns, sw and sh are the width and height size in pixels of the screen. This set defines the values of a_i and b_i that depend on the token range value μ.

On the right side of Fig. 3.3a, it can be observed that the number of elements n_e is equal to nine. For each element, a number of index i is designated. The main element the neural network will control must have $i = 1$, whereas the rest are randomly assigned. In order to calculate k_i, it is compulsory to obtain the token

range value $\mu = 10^{\tau}$, when $\tau \in \mathbb{Z}^{+}$, $10n_e \geq 10^{\tau} > n_e$, and $n_e < 100$. This means that for the example of $n_e = 9$ on the right side of Fig. 3.3a, $\mu = 10$ such that it is possible to use only one decimal to represent the key k_i of the element in ζ_i. When there are more than nine elements in the environment (See left side of Fig. 3.3a), then $\mu = 100$, in this manner, it is possible to represent the 11 elements with two decimals of ζ_i.

Since the elements in the environment are not only static but dynamic, a method to represent their position inside the CKF is necessary. The Set P contains the equivalences of the width and height of every element and the position of an element inside its own grid in $\text{CKF}_{n,m}$. This idea is illustrated in Fig. 3.3b. Despite the element being inside the same cell, the horizontal position a_i and vertical position b_i provide helpful information that can be encoded in ζ_i. This representation differentiates states even when the element is situated in the same cell of the CKF.

In Fig. 3.3c, the purpose of d_i is illustrated. When an element changes direction, this means a different state. Consequently, this information must be taken into account. The value of d_i is in the set of movement directions V. Let:

$$\begin{aligned} V = \{&\langle \rightarrow, \frac{1}{1000\mu} \rangle, \langle \nearrow, \frac{2}{1000\mu} \rangle, \langle \uparrow, \frac{3}{1000\mu} \rangle, \langle \nwarrow, \frac{4}{1000\mu} \rangle, \\ &\langle \leftarrow, \frac{5}{1000\mu} \rangle, \langle \swarrow, \frac{6}{1000\mu} \rangle, \langle \downarrow, \frac{7}{1000\mu} \rangle, \langle \searrow, \frac{8}{1000\mu} \rangle \}, \end{aligned} \tag{3.6}$$

be the set that contains the tuples $\langle V^1, V^2 \rangle$, where V^1 is the movement direction and V^2 its numerical value. With all the sets defined before, it is now possible to define the semantic set F, which is given by the following:

$$\begin{aligned} F = \{&\langle k_i, u_i, v_i, a_i, b_i, d_i, \dot{w_i}, \dot{h_i} \rangle \mid i \in [0, n_e), k_i \in N^2, \\ &u_i \in P^3, v_i \in P^4, a_i \in P^5, b_i \in P^6, d_i \in V^2, \\ &\dot{w_i} \in W^3, \dot{h_i} \in W^4 \}, \end{aligned} \tag{3.7}$$

Algorithm 1 CKFs generator.

Input : Semantic set F, the number of elements n_e, the screen width sw, and the screen height sh
Result : CKF
$n \leftarrow \lceil \frac{sh}{h_1} \rceil$;
$m \leftarrow \lceil \frac{sw}{w_1} \rceil$;
Initialise an $n \times m$ CKF as a zero matrix;
for $i = 1$, n_e **do**
  Get $k_i, u_i, v_i, a_i, b_i, d_i$ from F;
  for $r = 0$, w_i **do**
    for $c = 0$, h_i **do**
      $\text{CKF}_{(u_i+r, v_i+c)} = \zeta_i$;
    end for
  end for
end for

3.2.2 Iota Function

The affordances function explores the interactions among the elements of the environment, classifying what combinations of actions are not allowed according to the context of the environment. The affordances function is represented through $\iota(s)$ for a given state s. Let:

$$A = \{a_j \mid j \in \mathbb{Z}^+, a \in \{0, 1\}\}, \tag{3.8}$$

be the set of actions the agent can execute in the environment, where $n_a = |A|$ gives the total number of actions. Let:

$$\Psi = \{\langle a_j, k_i, \phi_i, \alpha_i \rangle \mid a_j \in A, i \in [0, n_e), \phi_i, \alpha_i \in \mathbb{Z}\}, \tag{3.9}$$

be the set of rules that contains which interactions among the actions and environment are evidently wrong, where a is the action, ϕ is the horizontal exploration range, and α is the vertical exploration range. These exploration ranges refer to how

many columns or rows from the agent's position should be considered to contemplate the affordability of an action. For example, if a pipe is three columns away from Mario and the vertical exploration range is set to 2, then the agent will keep selecting the action that moves Mario towards the pipe, but once Mario is two columns away, the agent will avoid that action. Table 3.3 shows five sets of rules manually defined for each environment. These sets of rules contain negative affordances, which are actions that the agent is not allowed to execute given that situation.

Algorithm 2 takes as input the state s in the shape of CKF, the set of rules Ψ, the first element of the set F, and the number of actions n_a. All the rules related to each action are explored through the CKF obtained using Algorithm 1. The exploration ranges ϕ, and α allow the agent to identify near or far elements that may put the agent into a bad state. However, decision-making using only $\iota(s)$ is not enough to find an optimal policy because there may be states with multiple choices where $\iota(s)$ does not provide the optimal action.

Algorithm 2 $\iota(s)$ generator.

Input : CKF, number of actions n_a, set of rules Ψ and semantic set F
Result : $\iota(s)$
Create an ι vector with n_a elements filled with *ones*;
Get k_1, u_1, v_1 from CKF;
for $a = 0$, n_a **do**
  Get $\Psi_a = \langle a_j, k_i, \phi_i, \alpha_i \rangle$ from Ψ;
  for $r = 0$, $|\Psi_a|$ **do**
    Get k_r, α_r, α_r from Ψ_a;
    for $p = u_1$, $u_1 + \phi_r$ **do**
      if $\text{CKF}_{p,v_1} = k_r$ **then**
        $\iota_a = 0$;
    end for
    for $l = v_1$, $v_1 + \alpha_r$ **do**
      if $\text{CKF}_{u_1,l} = k_r$ **then**
        $\iota_a = 0$;
    end for
  end for
end for
$\iota(s) \leftarrow \iota$;

3.2.3 Learning

This subsection explains how the CKFs and $\iota(s)$ are incorporated into the DQN, DDQN, DuDQN, and DDDQN algorithms. For this purpose, Algorithm 3 is introduced and how it can be applied to DQN variants to create the four new algorithms developed in this research: IDQN, IDDQN, IDuDQN, and IDDDQN. The main differences (Fig. 3.2) among the algorithms are in their neural network architecture (single or dueling), action selection (Q-values-stream-based or Advantage-stream-based) and loss functions (simple or double).

Algorithm 3 Context-based Deep Reinforcement Learning and its variants learning

```
Set neural network architecture;
\* Single stream for IDQN and IDDQN, and double stream for IDuDQN and
IDDDQN *\
Initialise main and target neural networks with random weights θ and θ', respectively;
Initialise a buffer D^replay;
for n number of episodes do
    for t = 1, T do
        Get a CKF with Algorithm 1;
        Get ι(s) with Algorithm 2;
        With probability ε, execute a valid action;
        \* Eq. (3.10) for IDQN and IDDQN or eq. (3.19)
        IDuDQN and IDDDQN *\
        Store a, s, s', r, ι(s), ι(s') transition in D^replay;
        Sample a transition from D^replay;
        if the state is terminal then
            Set J_ι(θ) = 0
            \* target = r for IDQN and IDuDQN or
            target_double = r for IDDQN and IDDDQN *\
        else
            Calculate the loss;
            \* J_simple(θ) for IDQN and IDuDQN or
            J_double(θ) for IDDQN and IDDDQN *\
        Perform gradient descent step on the loss;
        Update θ;
        if t mod τ = 0 then
            θ' ← θ
        s ← s'
    end for
end for
```

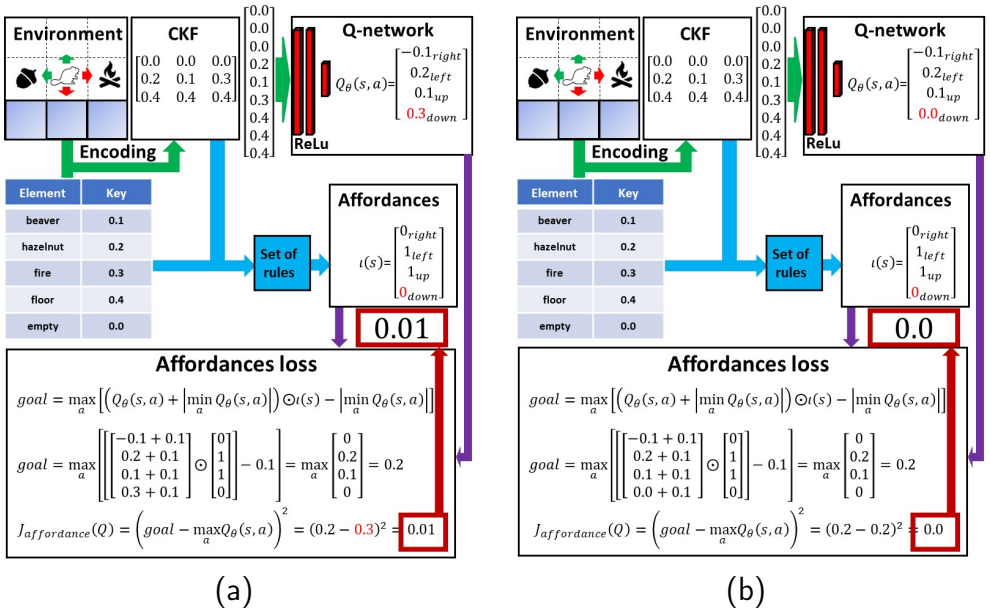


Figure 3.4: In (a), the neural network predicts a non-valid action. Consequently, the affordances loss increases. In (b), the neural network outputs respect the boundaries given by $\iota(s)$. This provokes the affordances loss to be equal to zero.

The approach begins with the integration of the IECR framework into DQN, referred to as **IDQN**. The affordances function $\iota(s)$ is used for selection-making tasks as follows:

$$a_t = \arg\max_a [(Q_\theta(s,a) + |\min_a Q_\theta(s,a)|) \odot \iota(s)] \tag{3.10}$$

Adding $\iota(s)$ to the action selection process produces useful data but not optimal solutions. This is because $\iota(s)$ can provide more than one action depending on the state, and consequently, this information is insufficient, and it is not possible to know which one of these actions is the best one. Given these circumstances, the only manner to find the optimal action and explore the environment using the $\iota(s)$ function is to embed this information into the target function of DQN given by Eq. (2.26). Hence, this is done by applying the Hadamard product operation:

$$target = r + \gamma \max_{a'}[(Q_{\theta'}(s', a') + |\min_{a'} Q_{\theta'}(s', a')|) \odot \iota(s') - \min_{a'} Q_{\theta'}(s', a')] \tag{3.11}$$

For IDQN, its main simple neural network loss $J_{\iota n}(\theta)$ is given by the following:

$$J_{\iota n}(\theta) = \frac{1}{K} \sum_{i=1}^{K} (target - Q_\theta(s, a))^2, \tag{3.12}$$

where K is a sample mini-batch of a given number of transitions from the replay buffer. In the experiments, the mini-batch size K is set to 64. Since it is possible to obtain $\iota(s)$, the target value goal for the main neural network, which indicates how the network over-estimates impossible actions according to $\iota(s)$, can be calculated as follows:

$$goal = r + \gamma \max_{a}[(Q_\theta(s, a) + |\min_{a} Q_{\theta'}(s, a)|) \odot \iota(s) - \min_{a} Q_\theta(s, a)] \tag{3.13}$$

In Fig. 3.4, the affordance loss $J_{affordance}(\theta)$ functionality is illustrated. The loss increases when the neural network overestimates a non-valid action (refer to the illustrative example in Fig. 3.4a). This overestimation happens when the neural network predicts a higher value for non-valid actions. Therefore, when the overestimated values of the neural network, corresponding to one or several actions, coincide with the zeros of $\iota(s)$, the loss increases. On the contrary, when the neural network respects the constraints given by $\iota(s)$, the loss is equal to zero (Fig. 3.4b). The affordance loss is given by the following:

$$J_{affordance}(\theta) = \frac{1}{K}\sum_{i=1}^{K}(goal - \max_{a} Q_\theta(s,a))^2 \tag{3.14}$$

The total simple loss $J_{simple}(\theta)$ is the sum of the main simple neural network loss $J_{\iota n}(\theta)$ and the affordance loss $J_{affordance}(\theta)$:

$$J_{simple}(\theta) = (1-\lambda)J_{\iota n}(\theta) + \lambda J_{affordance}(\theta) \tag{3.15}$$

The λ parameter controls the effect between losses. The effects of removing the affordance loss are investigated in the next section. Algorithm 3. **IDDQN** is the implementation of DDQN in IDQN. The difference when compared with IDQN is in its $target_{double}$ equation given by the following:

$$\begin{aligned} target_{double} = r + \gamma Q_{\theta'}(s', \arg\max_{a'}[(Q_{\theta'}(s',a') + \\ |\min_{a'} Q_{\theta'}(s',a')|) \odot \iota(s') - \min_{a'} Q_\theta(s',a')]) \end{aligned} \tag{3.16}$$

Given $target_{double}$ is possible to obtain the main double neural network loss $J_{\iota nd}(\theta)$:

$$J_{\iota nd}(\theta) = \frac{1}{K}\sum_{i=1}^{K}(target_{double} - \max_{a} Q_\theta(s,a))^2 \tag{3.17}$$

The total double loss for IDDQN, $J_{double}(\theta)$, is the sum of the main double neural network loss $J_{\iota nd}(\theta)$ and the affordance loss $J_{affordance}(\theta)$:

$$J_{double}(\theta) = (1 - \lambda)J_{ind}(\theta) + \lambda J_{affordance}(\theta) \tag{3.18}$$

IDDQN differs from IDQN in the way the total loss is calculated. IDQN uses $J(\theta)_{simple}$, whereas IDDQN uses the double loss $J(\theta)_{double}$. For action selection, both algorithms use equation (3.10). IDQN and IDDQN use neural networks with single streams (Fig. 2.2 top image).

On the other hand, dueling architectures use the advantage stream to select an action during the training process. **IDuDQN** utilises neural networks with dueling streams (Fig. 2.2 bottom image) and the simple loss function $J(\theta)_{simple}$. However, the DuDQN action selection equation is given by the following:

$$a_t = \arg\max_a [(A_\theta(s, a) + |\min_a A_\theta(s, a)|) \odot \iota(s)] \tag{3.19}$$

IDDDQN utilises neural networks with dueling streams (Fig. 2.2 bottom image), the first for the advantage and the second for the Q-values. It also selects an action from the advantage stream using the equation (3.19). DDDQN uses the double loss $J(\theta)_{double}$.

IDQN, IDDQN, IDuDQN and IDDDQN require a buffer replay D^{replay}, a main neural network Q_θ, and a target neural network Q'_θ. The weights of the main neural network θ are copied to the target neural network weight θ' every n_{step}, where n_{step} is usually set to 100. The pseudo-code of Algorithm 3 explains how to implement the algorithms mentioned above.

3.3 Experimental Setup

This section describes the experimental setup used in this work to evaluate the performance of the IDQN, IDDQN, IDuDQN, and IDDDQN algorithms. For the experiments, five environments were utilised (Fig. 3.5). Each environment has distinctive characteristics that challenge the algorithms in multiple ways. The experiments are conducted in two stages. The first stage of experiments evaluates how IDQN performs against DQN and a random policy to prove that IDQN can learn from the CKFs' representation. The second stage of experiments compares the performance of IDQN, IDDQN, IDuDQN, and IDDDQN against their state-of-the-art variants. For both stages of experiments, the CKFs' representation is used as the input of the neural networks. The stable-baselines implementations of DQN, DDQN, DuDQN, DDDQN, Proximal Policy Optimisation (PPO) and Asynchronous Actor-Critic (A2C) were used to compare the performance of the proposed algorithms.

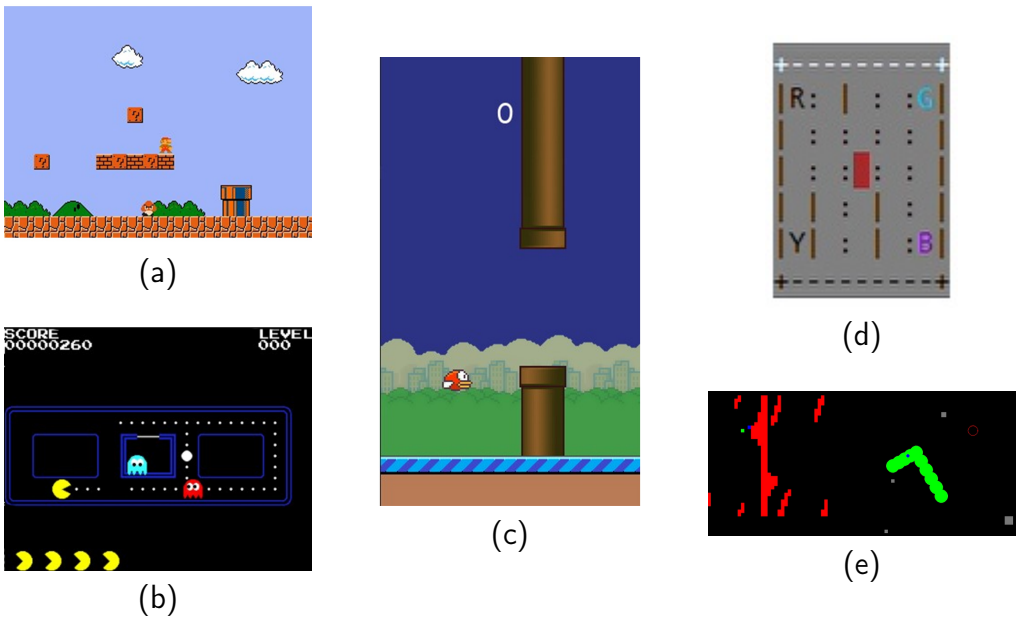


Figure 3.5: Game environments. (a) Mario. (b) Pacman. (c) FlappyBirds. (d) TaxiDriver. (e) ScaraRobot

In the first stage, the learning progress in every episode was measured by using the average reward and based only on the decisions of the neural network. In the

second stage, the progress was measured in epochs. The same network architecture and hyperparameters were used for all the experiments: an input layer which flattens the CKF, then two hidden layers of Rectified Linear Unit (ReLU) neurons and the output layer, which depends on how many actions the environment in turn has. The neural network architectures (single and double stream) are shown in Fig. 3.6. To this end, Table 3.2 shows the parameter values used for all the environments and experiments. The values of these hyperparameters were found to be the best by trial and error.

Table 3.2: Hyperparameters used during the experiments.

Parameter	Value
Learning rate (γ)	0.99
Mini-batch size	64
Episodes	2000
Epochs	200
Optimiser	Adam
Optimiser learning rate	0.0001
Loss function	*huber*
Training steps per epoch	400
Maximum steps per episode	3000
Target network update steps (τ)	100
Replay buffer size	50000
ϵ_{max}	0.9
ϵ_{min}	0.05
$\sigma_{episode}$ (ϵ decay per episode)	0.001
σ_{epoch} (ϵ decay per epoch)	0.01

3.3.1 Environments Description

Mario (Fig. 3.5a) is a game in which the free space available in the environment allows the agent to move almost everywhere, and this is a challenge. Moreover, negative rewards could take longer to propagate. For example, suppose there is a hole, and the jumping action was executed several states before. In that case, it will take several steps to make the agent understand that the action taken puts the

agent in a deficient state. The reward function of the Mario environment is given by:

$$r_{mario}(s,a) = \begin{cases} 10, & \text{finishing the game} \\ -10, & \text{dying} \\ 1, & \text{moving forward one square} \\ 0, & \text{otherwise} \end{cases} \tag{3.20}$$

Pacman (Fig. 3.5b) is a simple game where the agent's path is highly constrained. An action that immediately crashes with a wall or an enemy is easily identifiable in this environment. However, using the epsilon policy, the agent does not consider evident mistakes. On the other hand, a $\iota(s)$ function can easily avoid immediate mistakes and speed up the agent's learning process. The reward function of the Pacman environment is given by:

$$r_{pacman}(s,a) = \begin{cases} 10, & \text{finishing the game} \\ -10, & \text{dying} \\ 1, & \text{eating a pellet} \\ 0, & \text{otherwise} \end{cases} \tag{3.21}$$

Even though FlappyBirds (Fig. 3.5c) is a game with only two actions (fly and fall), their random selection under the same probability provokes the agent to stay at the top of the screen, since the fly action has a more biased behaviour than the fall action. The replay buffer will eventually fill with useless data that affect the agent's learning process. The reward function of the FlappyBirds environment is given by:

$$r_{flappybirds}(s,a) = \begin{cases} 10, & \text{finishing the game} \\ -10, & \text{dying} \\ 1, & \text{passing the pipes} \\ 0, & \text{otherwise} \end{cases} \tag{3.22}$$

TaxiDriver (Fig. 3.5d) and ScaraRobot (Fig. 3.5e) are environments that can only be solved by finishing two tasks: pick and drop. This characteristic complicates the exploration-exploitation process of the agent because the buffer fills more with demonstrations of the first task (pick) than the second one (drop). The reward function of the TaxiDriver environment is given by:

$$r_{taxidriver}(s,a) = \begin{cases} 10, & \text{picking in the right place} \\ 10, & \text{dropping in the right place} \\ 0, & \text{otherwise} \end{cases} \tag{3.23}$$

The reward function of the ScaraRobot environment is given by the following:

$$r_{ScaraRobot}(s,a) = \begin{cases} 10, & \text{picking in the right place} \\ 10, & \text{dropping in the right place} \\ 1, & \text{getting closer to the goal} \\ 0, & \text{otherwise} \end{cases} \tag{3.24}$$

For all the experiments, the average reward (produced from the actions of the neural network) is an indicator of learning progress. The set of rules of each environment is given in Table 3.3. Moreover, the actions that can be taken in every

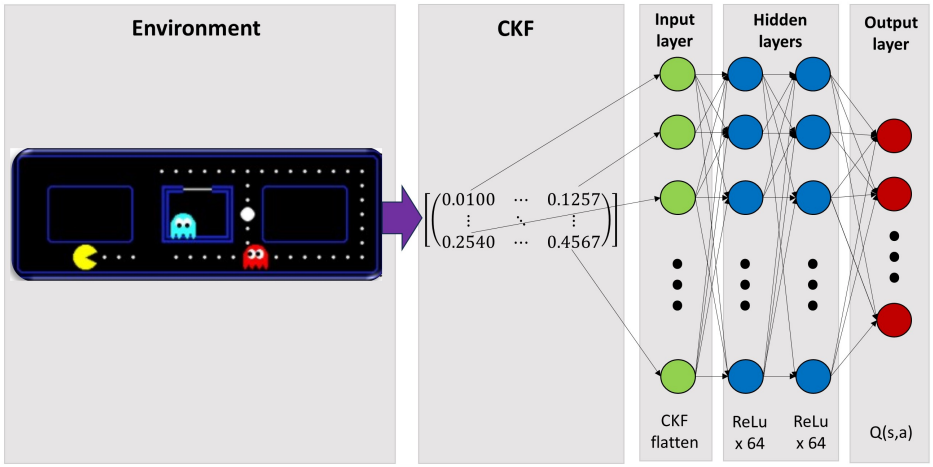
environment can be found therein the first tuple's element from their corresponding set of rules. For example, for the Mario environment, it is possible to take the actions $right$ and $jump$, whereas for Pacman, the available actions are $right$, $left$, up, and $down$.

Table 3.3: Set of rules for each game environment.

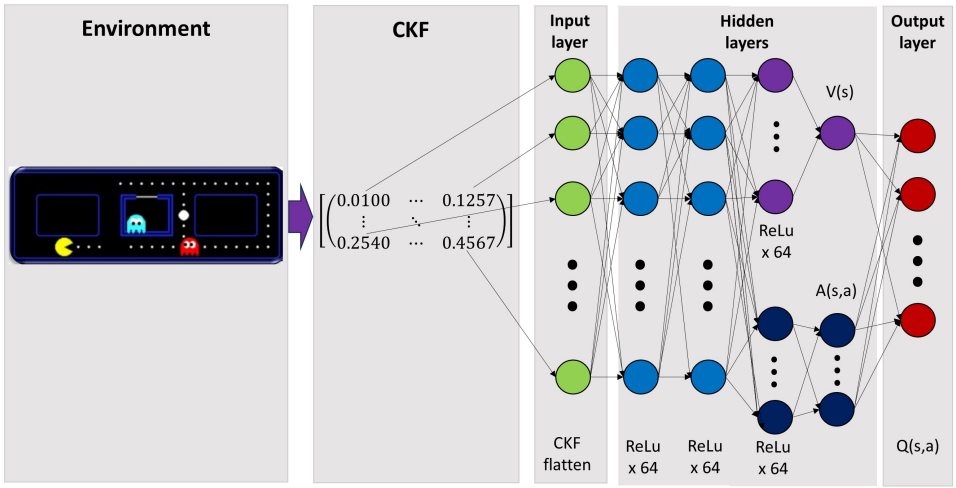
Environment	**Set of rules**
Mario	$\Psi_{Mario} = \{\langle right, pipe, 1, 0\rangle, \langle right, enemy, 2, 0\rangle, \langle right, hole, 2, 0\rangle, \langle right, block, 1, 0\rangle, \langle jump, empty, 0, -1\rangle\}$
Pacman	$\Psi_{Pacman} = \{\langle right, wall, 1, 0\rangle, \langle left, wall, -1, 0\rangle, \langle up, wall, 0, 1\rangle, \langle down, wall, 0, -1\rangle, \langle right, ghost, 1, 0\rangle, \langle left, ghost, -1, 0\rangle, \langle up, ghost, 0, 1\rangle, \langle down, ghost, 0, -1\rangle\}$
FlappyBirds	$\Psi_{FlappyBirds} = \{\langle fly, pipe_{up}, 3, 0\rangle, \langle fly, pipe_{up}, 0, 1\rangle, \langle fly, ceiling, 0, 1\rangle, \langle fall, pipe_{down}, 3, 0\rangle, \langle fall, pipe_{down}, 0, -1\rangle, \langle fall, floor, 0, -1\rangle\}$
TaxiDriver	$\Psi_{TaxiDriver} = \{\langle right, wall, 1, 0\rangle, \langle left, wall, -1, 0\rangle, \langle up, wall, 0, 1\rangle, \langle down, wall, 0, -1\rangle, \langle pick, wall, 0, 0\rangle, \langle pick, empty, 0, 0\rangle, \langle drop, wall, 0, 0\rangle, \langle drop, empty, 0, 0\rangle\}$
ScaraRobot	$\Psi_{ScaraRobot} = \{\langle right, obstacle, 1, 0\rangle, \langle left, obstacle, -1, 0\rangle, \langle up, obstacle, 0, 1\rangle, \langle down, obstacle, 0, -1\rangle, \langle pick, obstacle, 0, 0\rangle, \langle pick, empty, 0, 0\rangle, \langle drop, obstacle, 0, 0\rangle, \langle drop, empty, 0, 0\rangle\}$

3.3.2 First Stage of the Experiments

This experiment aimed to prove the neural network's capacity to learn from the CKFs. During this first stage of the experiments, IDQN was applied to solve the five environments. Additionally, the number of episodes was used as a common reference. Every episode is constrained to end if the agent reaches 3,000 steps, dies, or completes the game. In addition, three extra experiments are run for each environment. Thus, this could confirm that IDQN performs better than DQN and a full random policy. Moreover, DQN was performed using the same number of episodes IDQN took to learn.



(a)



(b)

Figure 3.6: Neural Network architectures used in the experiments. (a) shows the neural network setup used for IDQN and IDDQN. (b) shows the neural network setup used by IDuDQN and IDDDQN.

3.3.3 Second Stage of the Experiments

A problem with the first stage is that the incorporation of the affordance function $\iota(s)$ allows the agent to last longer during every episode. However, DQN makes more mistakes, and the number of training steps for the neural networks is significantly lower because every episode ends prematurely. The goal of this second stage is to deal with this unfair comparison. For this purpose, the measure of the learning

progress is carried out in epochs, where one epoch is conformed of 400 training steps. To this end, the influence of the affordance loss Eq. (3.14) is investigated by setting λ to 0, 0.5, 1, 5 and 10, respectively.

3.4 Results

This section describes the learning curves in Fig. 3.7 and Fig. 3.8 from the results of the first and second stages of the experiments, respectively. The affordance loss impact is also discussed at the end of this section [1].

3.4.1 Results of the First Stage of the Experiments

Fig. 3.7a to Fig. 3.7e show the learning curves in the first stage of the experiments for the five experimental environments: Mario, Pacman, FlappyBirds, TaxiDriver, and ScaraRobot. Both DQN and the random agents show similar performance within the Mario environment at the beginning of the training. The nature of the environment can explain this behaviour. While Pacman is highly constrained, Mario can take several actions within its open environment. Meanwhile, when there is an enemy or an obstacle, the $\iota(s)$ function provides actions that prevent the agent from dying or getting stuck. Consequently, IDQN feeds the replay buffer with better-quality data while reaching further states during the game.

In the Pacman environment, the efforts of DQN are impractical because every episode ends prematurely. Likewise, when using a random policy, the performance is poor because, most of the time, the agent can only go in two directions. Consequently, the equal probability of taking those actions leads the agent to a poor exploration of the environment. IDQN performs better because the $\iota(s)$ function avoids impractical actions such as crashing against a wall or an enemy. This behaviour may take more than 2,000 episodes for DQN to learn.

[1] Demo available at `https://youtu.be/Gqsud7KUZfM`

In the FlappyBirds environment, IDQN can easily outperform DQN because the stochastic nature of the ϵ-greedy policy exploration with equal probability indirectly biases the agent's behaviour. Therefore, the replay buffer fills with data related to crashes against the pipes, floor, or ceiling and rarely with passing through the pipes. Moreover, when randomly sampling a mini-batch from the replay buffer, the probability of picking a transition representing a positive reward when passing through the pipes is exceptionally low. Hence, the agent mainly trains with impractical data and rarely trains from transitions that represent the actual goal of the game. In contrast, IDQN can take decisions based on the obstacles around the bird. Thus, every episode lasts longer and fills the replay buffer with better-quality training data.

In the TaxiDriver and ScaraRobot environments, DQN presents difficulty in learning the drop action. In these environments, it is very easy to collide against obstacles due to the stochastic nature of DQN. This behaviour fills the buffer with useless data, and the episodes end prematurely. IDQN does not need to explore or learn when to pick or drop because that behaviour is already encoded in the affordance function $\iota(s)$.

Table 3.4: Average reward obtained in the second stage of the experiments.

Environment	IDQN	IDDQN	IDuDQN	IDDDQN	DQN	DDQN	DuDQN	DDDQN	PPO	A2C
Mario	26.24	25.87	30.24	23.69	23.24	20.28	27.06	23.88	23.94	−5.24
Pacman	45.77	45.03	39.71	41.16	23.02	14.63	20.36	26.05	25.68	6.55
FlappyBirds	15.18	15.18	14.16	13.49	−8.02	−8.26	−8.72	−8.49	−8.38	−8.83
TaxiDriver	14.51	16.22	16.54	18.1	1.11	1.11	1.24	0.9	7.05	0.22
ScaraRobot	690.35	715.17	679.27	647.82	239.26	258.12	182.97	133.04	108.48	7.83

3.4.2 Results of the Second Stage of the Experiments

For the second stage of the experiments, the learning curves in Fig. 3.8a to Fig. 3.8c show a comparison between the performance of the state-of-the-art algorithms DQN, DDQN, DuDQN, DDDQN, PPO, and A2C and the IECR variants IDQN, IDDQN, IDuDQN and IDDDQN. Table 3.4 summarises the results for this stage. The same number of training steps are applied to compare the algorithms fairly. The agent

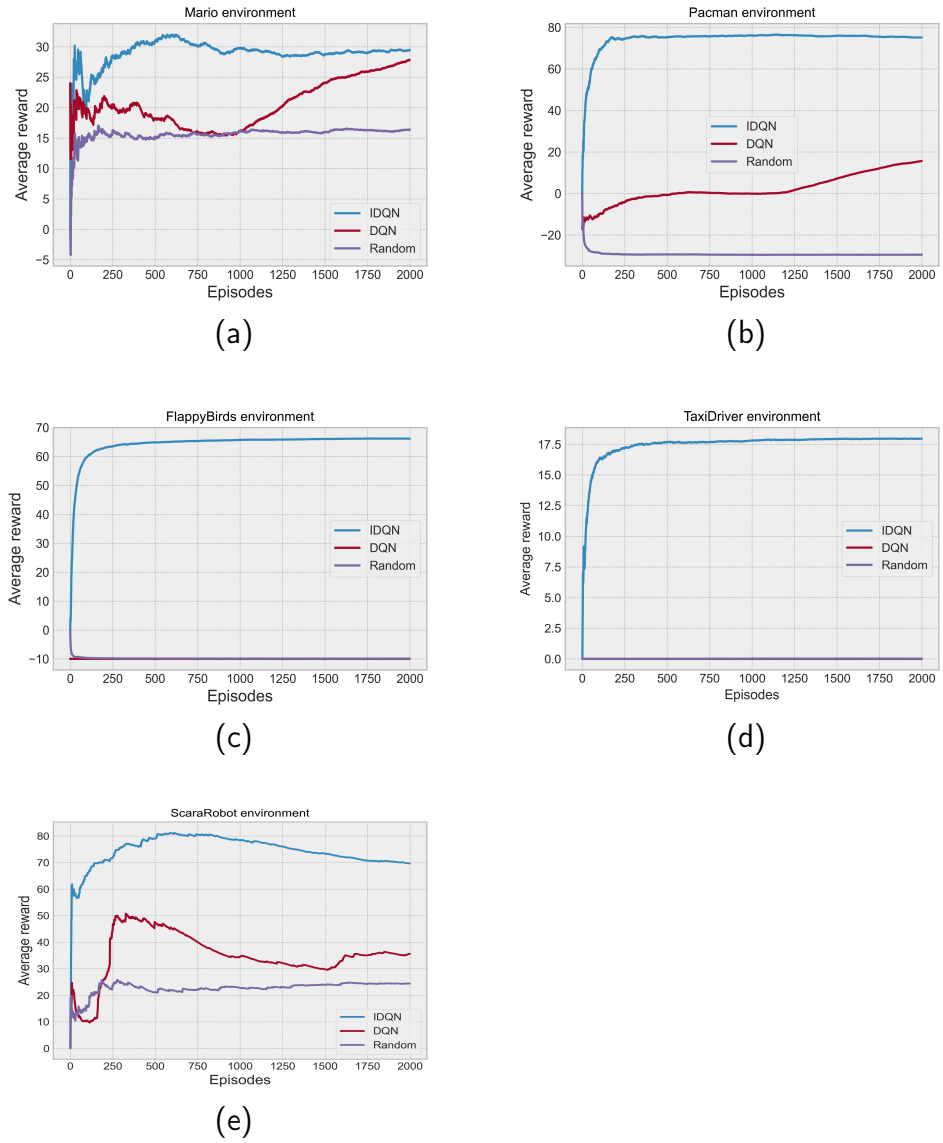


Figure 3.7: The earning curves above are the results of the first stage of the experiment in which the learning progress is measured in every episode: (a) Mario, (c) Pacman, (c) Flappybirds, (d) TaxiDriver and (e) ScaraRobot.

trained for 200 epochs, where every epoch is equivalent to 400 training steps. In the Mario environment, the DQN, DDQN, DuDQN, and DDDQN learning curves show more stable progress than in the first stage of the experiments. It can be observed better learning progress when $\iota(s)$ is involved than when it is not. In Mario's environment, the gap between state-of-the-art approaches is smaller than in Pacman and FlappyBirds because this environment has a huge open space where

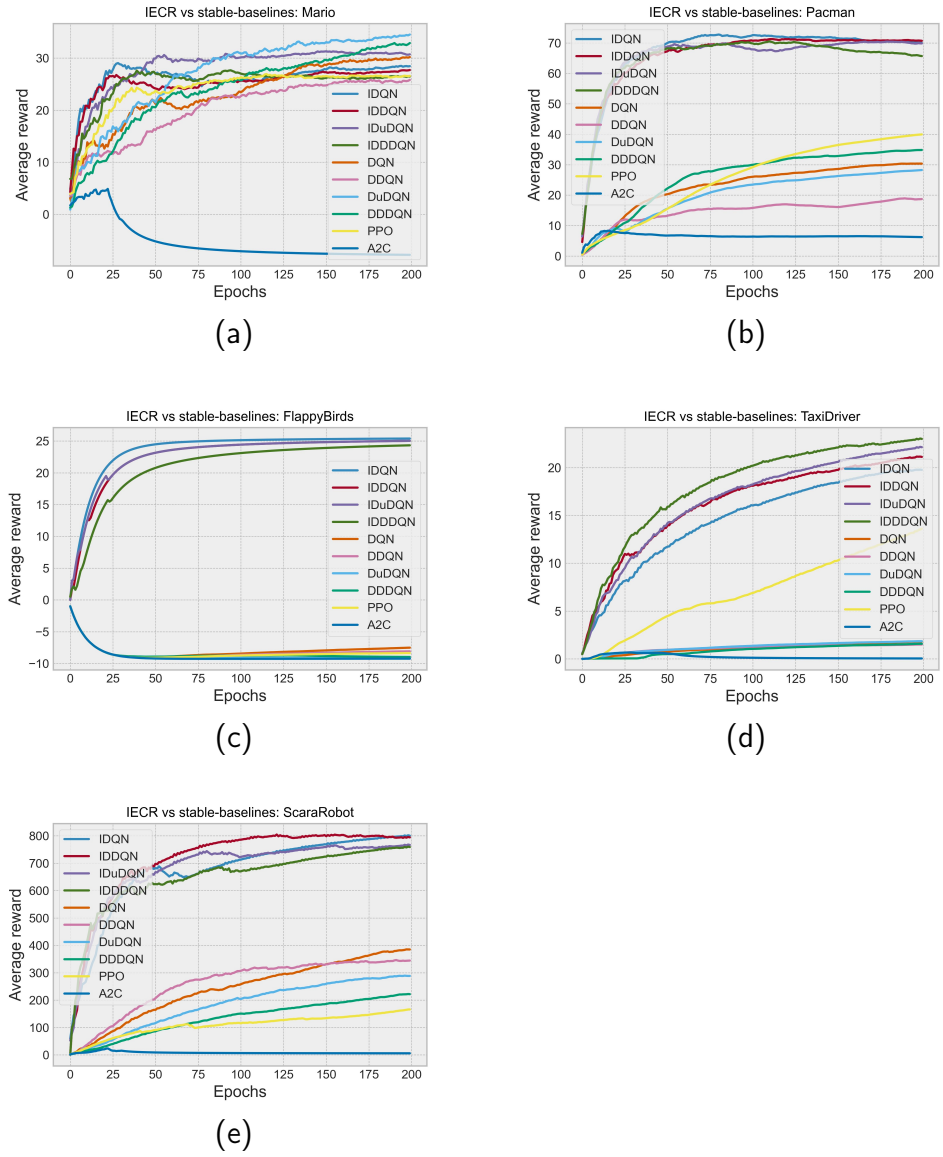


Figure 3.8: This figure displays the results of the second stage of the experiments, in which the progress is measured every epoch: (a) Mario, (c) Pacman, (c) Flappybirds, (d) TaxiDriver and (e) ScaraRobot. Every epoch is equivalent to 400 training steps. In the second stage, the learning progress of the state-of-the-art algorithms was less chaotic. However, IECR variants still outperform the state-of-the-art approaches.

the agent can decide where to go. Moreover, the $\iota(s)$ only has a notorious effect when enemies, pipes, or holes are nearby, and most of the time, all the actions are available. However, $\iota(s)$ assists in critical moments of decisions and allows the agent to explore further and fill the replay buffer with this information. Besides, PPO showed competitive learning progress, while A2C got stuck into a local minimum

(See Fig. 3.8a).

In the Pacman environment, $\iota(s)$ has a higher impact because environment obstacles such as walls and ghosts are always next to Pacman. While the state-of-the-art approaches use a stochastic policy for exploring the environment, provoking Pacman to crash against the walls or die by touching a ghost, $\iota(s)$ will avoid these states immediately.

In the FlappyBirds environment, the state-of-the-art approaches show difficulties in exploring the environment and finding high-value states. On the contrary, $\iota(s)$ explores and supports good decisions based on what is surrounding the bird. Consequently, the state-of-the-art approaches rarely train from adequate states, whereas $\iota(s)$ guides the agent into better decisions through every episode.

In the TaxiDriver environment, the affordances function $\iota(s)$ assisted the agent exploration efficiently because, once the taxi reached the passenger position, the only possible action according to the set of rules is picking. Therefore, the state-of-the-art algorithms must visit the same state several times to learn that picking is the only possible action at that state. In Fig. 3.8d, it can be noted that IECR variants learn faster due to the reason explained before.

The ScaraRobot environment behaves similarly to TaxiDriver. When the robotic arm reaches the picking or dropping positions, only one action is allowed (picking or dropping). However, the state-of-the-art approaches cannot produce enough high-quality data dependent on the action-selection process, which reverberates in their learning performance (Fig. 3.8a).

In summary, the second stage of experiments showed that all the algorithms which use contextual information significantly outperform the baseline algorithms by 77 % on average, according to the information in Table 3.4, in the environments used in this thesis. To calculate the improvement, the following equation is used:

Table 3.5: Performance comparison of the maximum reward obtained with IDQN, IDDQN, IDuDQN or IDDDQN from the learning curves of Fig. 3.8 and the stable baselines.

Environment (Max_{IECR})	**Algorithm**	$Max_{baseline}$	$P_{improvement}$
Mario (33)	DQN	30	10
	DDQN	25	25
	DuDQN	26	21
	DDDQN	27	19
	PPO	24	28
	A2C	-8	124
Pacman (70)	DQN	40	43
	DDQN	35	50
	DuDQN	30	47
	DDDQN	28	60
	PPO	19	27
	A2C	5	93
FlappyBirds (25)	DQN	-10	140
	DDQN	-9.5	138
	DuDQN	-9	136
	DDDQN	-8	132
	PPO	-7.5	130
	A2C	-7	128
TaxiDriver (23)	DQN	14	61
	DDQN	2	95
	DuDQN	2	95
	DDDQN	2	95
	PPO	2	95
	A2C	1	96
ScaraRobot (800)	DQN	400	50
	DDQN	350	56
	DuDQN	290	64
	DDDQN	210	74
	PPO	180	78
	A2C	1	99
Average			≈ 77

$$P_{improvement} = 100 \cdot (1 - \frac{Max_{baseline}}{Max_{IECR}}) \tag{3.25}$$

Here, $P_{improvement}$ represents the percentage of improvement of the *iota* algorithm Max_{IECR} with the highest reward compared to the maximum reward obtained by the baseline algorithms expressed as $Max_{baseline}$.

3.4.3 Affordances Loss Impact

The impact of the affordances loss was measured by calculating the percentage of improvement, with $\lambda = 0$ serving as a reference for all the environments. Fig. 3.9 summarises the results showed in Table 3.6. When $\lambda = 0.5$, IDQN and IDuDQN had an improvement of 7.5% and 6%, respectively. While IDDQN and IDDDQN

showed a deterioration in the performance of 1% and 14%. For $\lambda = 1$, IDQN and IDDQN increased their performance by 1.5% and 5.5%, whereas IDuDQN and IDDDQN performance reduced in 2.5% and 7.5%, respectively. The experiment results show that $\lambda = 5$ produced an increment in the performance of IDQN, IDDQN and IDuDQN of 6.8%, 1.8% and 0.5%, while IDDDQN performance dropped in 4.5%. Setting $\lambda = 10$ provoked that IDQN, IDDQN and IDuDQN increased their performance by 8.9%, 1.7% and 6.5%, respectively. While IDDDQN performance decreased 5.2%.

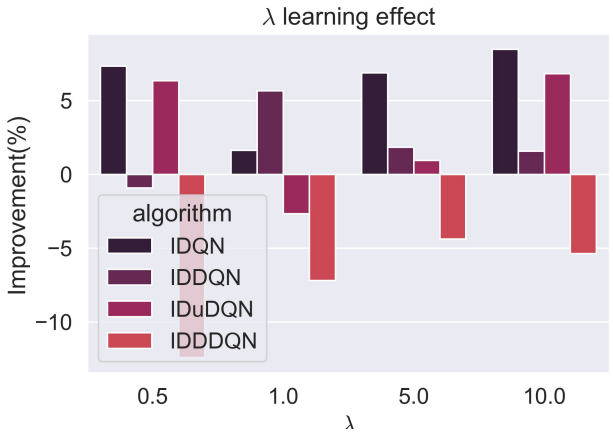


Figure 3.9: The results above show the effect of varying the value of λ in the algorithms.

3.5 Discussion

This chapter introduced the IECR framework, which takes advantage of the available information in the environment to accelerate the agent's learning process. This approach is first fed from a manually defined set of rules, much like what a human does. Then it uses the rules to generate an affordance function that seeks to reduce the exploration space and optimise the decision-making process.

In real-world tasks, humans understand the rules, so repeating the same mistake millions of times is unnecessary. IECR is a successful framework that uses this human capability to benefit from context and use it to make intelligent decisions.

Table 3.6: Affordance loss impact (average reward).

$\lambda = 0$	IDQN	IDDQN	IDuDQN	IDDDQN
Mario	24.91	25.01	26.75	36.36
Pacman	41.19	45.92	48.95	41.02
FlappyBirds	14.93	15.18	13.38	14.41
TaxiDriver	14.77	15.29	15.51	16.86
ScaraRobot	680.65	641.92	731.89	697.07

$\lambda = 0.5$	IDQN	IDDQN	IDuDQN	IDDDQN
Mario	25.65	26.81	23.46	23.81
Pacman	42.78	47.58	45.86	42.95
FlappyBirds	14.9	15.18	13.76	13.92
TaxiDriver	14.97	15.46	15.91	15.8
ScaraRobot	810.28	595.5	966.78	573.0

$\lambda = 1$	IDQN	IDDQN	IDuDQN	IDDDQN
Mario	26.24	25.87	30.24	23.69
Pacman	45.77	45.03	39.71	41.16
FlappyBirds	15.18	15.18	14.16	13.49
TaxiDriver	14.51	16.22	16.54	18.1
ScaraRobot	690.35	715.17	679.27	647.82

$\lambda = 5$	IDQN	IDDQN	IDuDQN	IDDDQN
Mario	26.42	24.54	24.79	25.75
Pacman	41.96	50.42	45.47	50.09
FlappyBirds	15.18	15.18	13.49	13.33
TaxiDriver	14.94	16.64	14.5	15.27
ScaraRobot	841.11	593.11	914.08	711.51

$\lambda = 10$	IDQN	IDDQN	IDuDQN	IDDDQN
Mario	25.14	26.12	24.31	29.34
Pacman	44.43	48.63	44.73	41.87
FlappyBirds	15.16	15.12	13.75	13.29
TaxiDriver	16.83	16.1	18.3	18.55
ScaraRobot	802.81	591.69	959.26	614.2

The fact that IECR variants outperform all these state-of-the-art approaches makes it clear how vital context is during the learning process of an agent.

In the FlappyBirds environment, the IECR variants IDQN, IDDQN, IDuDQN, and IDDDQN outperform state-of-the-art approaches. There may be better ap-

proaches than DQN, DDQN, DuDQN, or DDDQN. For example, a Q-table and QL could manage to solve this environment quickly. Since FlappyBirds is a highly repetitive environment, the number of states can be calculated. Another solution may be to use two replay buffers: one buffer for poor and average transitions and a second buffer exclusive for good transitions. It would be necessary to bias the sampling process of the data to collect good transitions so that the agent can learn and find a solution for the environment. Another solution may be to design a reward function that pushes the bird to the middle of the pipes. However, IDQN does not require such modifications, and a set of rules is enough to solve the environment.

The results in Table 3.6 show that λ and the two loss functions based on the affordance function ($J(\theta)_{simple}$ and $J(\theta)_{double}$) have a higher impact in IDQN, IDDQN and IDuDQN than in IDDDQN. It is believed that the reason for this is that the $J(\theta)_{double}$ loss function is connected to the Q-values stream of the target neural network. Hence, the weights of the main and target neural networks may provoke the possible actions in the advantage stream to differ from the ones in the Q-values stream.

Simply carrying out a stochastic exploration of the environment does not secure a good learning process for the agent. Therefore, the satisfactory results of IECR introduced in this chapter can be explained by the quality of the data produced with the assistance of the $\iota(s)$ function. Regarding the manual definition of the sets of rules, there are two possible approaches that can automate their generation. The first one would consist of logging what actions combined with certain elements produce negative rewards. However, this approach would not be able to deal with ambiguous cases such as the ones where the agent is stuck and not getting any reward. Besides that, it would again bring the problem of designing a reward that indirectly guides the learning process. The second alternative, which is more promising, consists of using Large-Language Models (LLMs) (Brown et al. 2020). A possible procedure to follow would be feeding the image with a description of the

game and samples of the actions taken by the learning agent, aiming to use the strong contextual capabilities of LLMs to design a human-like set of rules. Lastly, IECR has the potential to be extended to more baseline algorithms such as PPO and A2C. To achieve this, it would be necessary to design the corresponding loss functions involving $\iota(s)$ and integrate the exploration based on the affordances in the same way it was done for DQN and its variants.

3.6 Summary

This chapter has explored the integration of context in RL and opened the door to new challenges in RL. The framework developed in this chapter demonstrates a smooth integration of contextual data in DRL. In this chapter, it has been shown that the proposed framework and the resulting algorithms IDQN, IDDQN, IDuDQN, and IDDDQN improved the agent's performance by converging in the experimental environments in around 40,000 training steps. Since a manually defined set of rules and the concept of CKFs were included, an affordance function could be obtained. Consequently, the agent's learning profits from better quality data than it would do with a pure stochastic exploration. The contributions of this chapter include an intuitive framework that includes contextual information to improve RL and four new algorithms based on IECR. The results of the first stage of experiments showed IDQN's capacity to learn from CKFs representation. The second stage of the results shows that by using the same number of training steps (40,000) as a reference, IECR variants also outperform the state-of-the-art algorithms. IECR provides a solution to difficult states, where instead of learning after millions of interactions, the agent computes $\iota(s)$ and avoids getting stuck before continuing to explore the environment.

Chapter 4

Affordance-based Reinforcement Learning for Human-Robot Interaction

Chapter 3 demonstrated that it is possible to integrate contextual information into the learning process of a Reinforcement Learning (RL) agent. More concisely, the representation of the state in Contextual Key Frames (CKFs) combined with a set of rules has shown to be effective when training neural networks for solving discrete environments. However, the conditions are different in a real-world setup, such as a Human-Robot Interaction (HRI) scenario. For example, all the information regarding the current state is not as easy to extract in the real world as it is with a simulation. Besides that, the stochastic behaviour of a human user is not easy to deal with only with a pre-trained neural network due to the problem of generalisation discussed in Chapter 2. With this idea in mind, the current chapter aims to propose a solution by integrating contextual information into the Q-learning (QL) algorithm, a set of rules, and to represent the current real-world scenario with CKFs in an HRI setup. The rest of this chapter is structured as follows. It begins with Section 4.1, which motivates current problems of RL in HRI setups. Then, in Section 4.2, the

framework based on Contextual Q-learning (CQL) is formally introduced. This is followed by Section 4.3, which describes the experimental setup. Section 4.4 discusses the implementability of the framework in an HRI scenario and presents the results. Finally, the chapter concludes in Section 4.6.

4.1 Introduction

Robots with human-level intelligence to plan and adapt to dynamic environments may open the door to the full integration of robotics in industrial and domestic environments (Khan et al. 2020). In the context of HRI, RL has been applied to adapt the behaviour of the robot to the user in dynamic environments, such as finding optimal parameters in a robot arm impedance model (Modares et al. 2015), biped dynamic walking (J.-L. Lin et al. 2016) and allowing navigation among crowds (C. Chen et al. 2019; Everett et al. 2021). HRI is challenging and requires dealing with dynamic changes in the environment. A robot with the capacity to reprogram itself when necessary may lead to improve HRI (Ciou et al. 2018; Cruz et al. 2016b; Y. Gao et al. 2019; Lathuilière et al. 2018; W. Wang et al. 2019). This is particularly important in grasp, and release-related tasks, where combining RL capacities to adapt to dynamic environments and HRI is crucial (Andriella et al. 2020).

In the literature, several approaches exist for automating repetitive robot tasks based on HRI, such as LEarning from Demonstrations (LfD). Usually, LfD is executed employing three methods: kinesthetic, passive observation, and teleoperation (Ravichandar et al. 2020). In kinesthetic demonstrations, the user manually guides the robot by pulling or pushing the end effector (Stavridis et al. 2022; X. Yu et al. 2020). Passive observation is when the robot learns from the user through video streams (Hwang et al. 2020). With teleoperation, the user guides the robot by operating a teach pendant, a joystick, or a haptic device (Pareek and Kesavadas

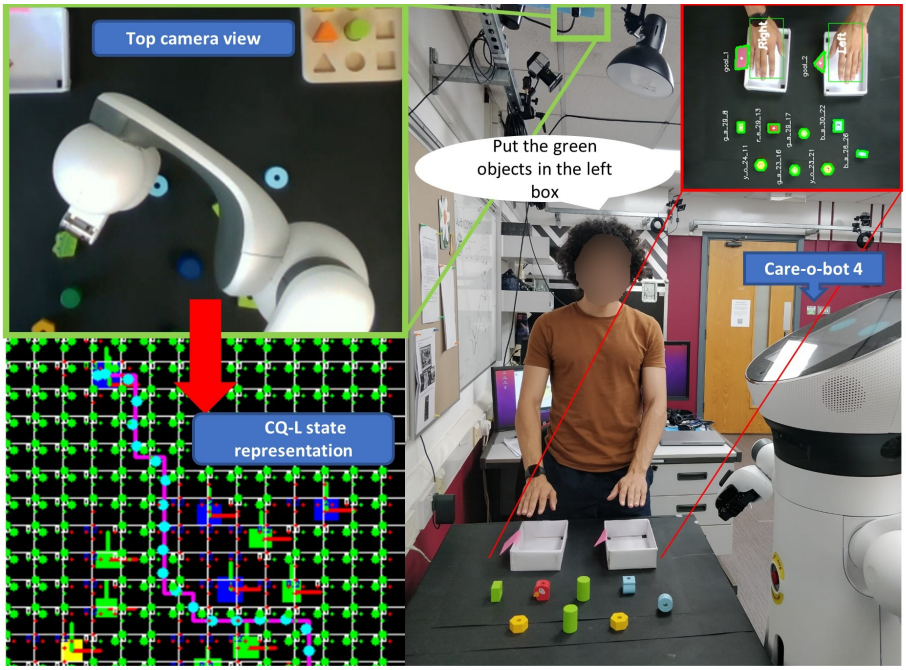


Figure 4.1: Experimental setup. The view of the camera and a visual representation of the state can be appreciated at the bottom left and the upper right of the image, respectively.

2019). A limitation of LfD approaches is that these are constrained to the solution shown by the user, and more optimal solutions are often discarded. Hence, RL offers a solution to this problem by encouraging the agent to explore further and find better solutions than the user's demonstrated one.

This chapter presents an affordance-based human-robot interaction framework whose novelty lies in its capacity to use contextual information (semantic information, affordances, and high-level goals) that enhances the exploration and learning process of the agent. The framework is based on a new algorithm called CQL. The algorithm aims to reduce the exploration space of the agent and the number of states required to represent it. This allows CQL to find a policy from the current observation in the real world and solve the Q-table in a short period of time. This fast learning capacity makes the framework suitable for HRI tasks. Our contributions are (i) CQL is introduced, allowing efficient learning in the context of active HRI, and (ii) a framework based on CQL that allows robots to perform HRI in the real world.

The problem domain to empirically validate the framework is an HRI scenario (Fig. 4.1). Here, the user first provides instructions and then actively interacts with the robot to manipulate objects. At the same time, changes that interfere intermittently with the robot's actions are produced. CQL is validated experimentally by comparing its performance against baseline algorithms such as classical QL (Watkins and Dayan 1992a), Deep Q-network (DQN) (Mnih et al. 2013), Proximal Policy Optimisation (PPO) (Schulman et al. 2017) and Asynchronous Actor-Critic (A2C) (Mnih et al. 2016) from the stable-baselines (Hill et al. 2018) implementations.

4.2 Affordance-based Human-Robot Interaction Framework

This section presents a framework that aims to solve the problem of performing grasp and release-related operations during HRI (Fig. 4.2). The framework is composed of three modules: voice-gestures, learning, and valid policy detector. The nomenclature used in this chapter is summarised in Table 4.1.

Symbol	Description
S	Set of states
A	Set of actions
G	Set of Goals
O	Semantic set
λ	Set of affordable actions
Ψ	Set of rules
I	Set of primary actions
E	Set of complementary actions
n_e	Number of elements
$\zeta(s,a)$	Affordances function of each state-action pair
$\xi(s,a,g)$	Initial optimistic values function
g_i	Name of the goal i-th goal
x_i	Coordinates of the i-th goal in the x axis
y_i	Coordinates of the i-th goal in the y axis
z_i	Coordinates of the i-th goal in the z axis
$s_{n,m}$	State in the form of a $n \times m$ matrix
k_i	Token of the i-th element
w_i	Width of the i-th element
l_i	Length of the i-th element
h_i	Height of the i-th element
$name_i$	Name of the i-th element
a_t	Maximum affordable action

Table 4.1: Nomenclature table for Chapter 4.

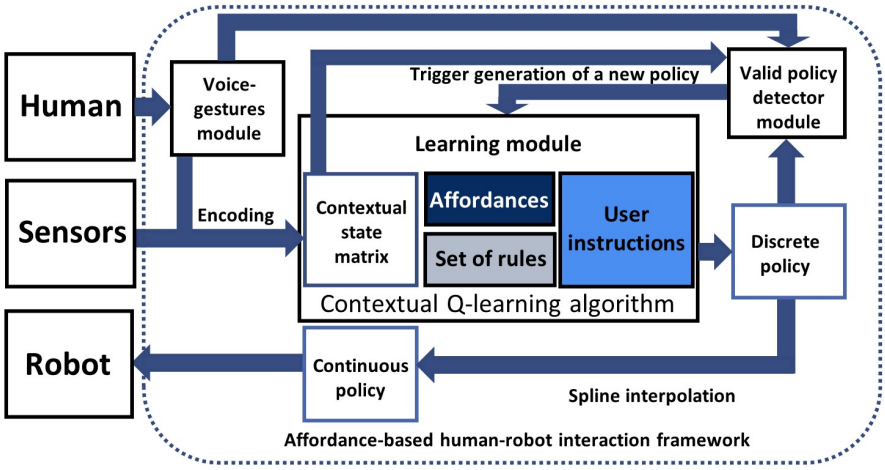


Figure 4.2: Proposed framework for HRI using RL.

4.2.1 Voice-gestures

With this module, to extract the sub-goals set Π, the robot acquires human instructions by using the Google speech-to-text API (Google n.d.) and the CVZone package (CVZone n.d.) for hand-tracking. When the instruction ι is ready, the algorithm 4 processes the string and fills Π. When the word “here" is in the instruction, the module tracks the hand position and finds the closest goal, which is stored in the set of goals G, given by:

$$G^4 = \{\langle g_i, x_i, y_i, z_i\rangle \,|\, x_i, y_i, z_i \in \mathbb{R}\}, \tag{4.1}$$

where g_i is the name of the goal. The terms x_i, y_i, and z_i are the coordinates of the ith goal in the x, y, z axis respectively. This allows the replacement of "here" with the goal's name. Consequently, algorithm 4 can be applied to relate the goals with the objects and split the task into sub-goals. For each sub-goal, CQL solves its respective contextual Q-table. When all the sub-goals are completed, or Π is empty, the task is considered finished.

Algorithm 4 Sub-goals extractor.

Input : Human instruction ι, Semantic set O, the phrase set $\rho = \{\}$
Result : Sub-tasks set Π
while$|\iota| > 0$
  for *word* in ι **do**
    $\rho = \rho \cup \{word\}$
    if $|\rho \cap G^1| > 0$ and $|\rho \cap V| > 0$
      $goal \in \rho \cap G^1$
      if $|\rho \cap \{all, every, the\}| > 0$ and $|\rho \cap \mathbb{Z}| = 0$
        $\Pi = \Pi \cap \{\langle obj, goal\rangle \,|\, obj \in O^1 \cap \rho\}$
      if $|\rho \cap \{some, the\}| > 0$ and $|\rho \cap \mathbb{Z}| = 0$
        $\Pi = \Pi \cap \{\langle obj, goal\rangle \,|\, obj \in H\}$
      if $|\rho \cap \mathbb{Z}| > 0$
        Set n to $|\rho \cap \mathbb{Z}|$;
        Fill set H by randomly pick n objects from
        $O^1 \cap \rho$;
        $\Pi = \Pi \cap \{\langle obj, goal\rangle \,|\, obj \in H\}$
      if $|\rho \cap \{a, an\}| > 0$
        $\Pi = \Pi \cap \{\langle obj, goal\rangle \,|\, obj \in_R O^1 \cap \rho\}$
      Remove ρ elements from ι and set $\rho = \{\}$
  end for
if $|\Pi| = 0$
  Send error: "Not executable instruction."
end

4.2.2 Learning

The learning module uses the proposed algorithm CQL to learn the set of actions that solve a task established by the user through the voice-gesture module. CQL uses a set of codes that numerically represents the state s of the environment. This set is used to create contextual Q-tables and to extract the affordances. Formally, a state s is a $n \times m$ matrix that contains semantic information of the state and $s_{n,m} \in \{0, 0.1, 0.2, 0.3, 0.4\}$, where 0 is an empty space, 0.1 is used to point an object as the target to manipulate, 0.2 stands for objects that become obstacles in the environment, 0.3 represents the goal, and 0.4 represents a hand. A number will be added to tokenise every new object type in s. The value of the tokens is in the range between zero and one to avoid state-of-the-art approaches to learning spurious correlations due to high values. This normalisation also assists with faster

convergence by avoiding large input values that may mislead the agent. The objects' semantic information is represented in the object's set O, given by:

$$O^8 = \{\langle k_i, x_i, y_i, w_i, l_i, h_i, name_i\rangle \mid i \in (0, 1, .., n), \\ x_i, y_i, w_i, l_i, h_i \in [0, \infty)\}, \tag{4.2}$$

where O represents the semantic set, $i \in (0, 1, ..., n)$, k_i is the ith of the element , x and y are the position coordinates in pixels, w is the width, l is the length, h is the height, and $name_i$ is the name of the object (e.g., "blue-square-1", "yellow-hexagon-2").

Algorithm 5 uses the semantic set O to create a matrix that represents the state s. The dimension of the states matrix s is given by $n \times m$. The set actions are defined as follows:

$$A = \{UP, DOWN, LEFT, RIGHT, GRASP, \\ DROP, GOAL\}, \tag{4.3}$$

where UP and $DOWN$ are the displacements along the y axis given by $\pm l_i$. $RIGHT$ and $LEFT$ are the displacements along the x-axis given by $\pm w_i$, $GRASP$ closes the gripper of the robot and controls its orientation according to l_i and w_i, $DROP$ opens the gripper, and $GOAL$ is the trajectory from the coordinates of the closest state of the target to the robot to the goal's position. The affordances Λ are given by:

$$\Lambda : A \times S \longrightarrow \mathbb{Z}_2^{|A|}, \tag{4.4}$$

Context is comprised of the affordances Λ, the semantic set O, the state s and a set of rules Ψ. Let:

Algorithm 5 State generator.

Input : Semantic set O, the number of elements n_e, the screen width sw, and the screen height sh
Result :s
$n \leftarrow \lceil \frac{sh}{h_1} \rceil$;
$m \leftarrow \lceil \frac{sw}{w_1} \rceil$;
Create an $n \times m$ matrix s filled with $zeros$;
for $i = 1$, n_e **do**
  for $r = 0$, w_i **do**
    for $c = 0$, h_i **do**
      if k_i is the target **do**
        $s_{(u_i+r,v_i+c)} = 0.1$;
      if k_i is an obstacle **do**
        $s_{(u_i+r,v_i+c)} = 0.2$;
      if k_i is the goal **do**
        $s_{(u_i+r,v_i+c)} = 0.3$;
      if k_i is a user's hand **do**
        $s_{(u_i+r,v_i+c)} = 0.4$;
    end for
  end for
end for

$$\begin{aligned}\Psi = \{&\langle UP, 0, 0, 1\rangle, \langle DOWN, 0, 0, -1\rangle,\\ &\langle LEFT, 0, -1, 0\rangle, \langle RIGHT, 0, 1, 0\rangle,\\ &\langle GRASP, 0.1, 0, 0\rangle, \langle DROP, 0.3, 0, 0\rangle,\\ &\langle GOAL, 0.3, \pm 1, \pm 1\rangle\},\end{aligned} \tag{4.5}$$

be the set of rules manually defined that contains which interactions among the actions and environment should not be performed by the robot. For example, picking an object when there is nothing to pick or moving to a place where the robot may collide. Ψ^3 is the horizontal exploration range and Ψ^4 is the vertical exploration range. The affordance ζ of each state-action pair is given by:

$$\zeta(s,a) = \begin{cases} 1, & |\{\langle a, s_{p,q}, u, v\rangle\} \cap I| > 1 \text{ and} \\ & |\{\langle a, s_{p+u,q+v}, u, v\rangle \,|\, a \in E\} \cap R| = 0 \\ 1, & |\{\langle a, s_{p,q}, u, v\rangle\} \cap E| > 1 \\ 0, & \text{otherwise} \end{cases} \tag{4.6}$$

Where $a \in A$, $p \in (0,n)$, $q \in (0,m)$, $u,v \in \{-1,0,1\}$, $I = \{UP, DOWN, LEFT, RIGHT\}$ is the primary actions set, and $E = \{GRASP, DROP, GOAL\}$ secondary actions set. With the affordances equation (4.5) and the current state, it is possible to generate a contextual Q-table and set optimistic initial values by:

$$Q(s,a) = \zeta(s,a) + \xi(s,a,g), \tag{4.7}$$

where $\xi(s,a,g) = 0.1$, when the action a points to the goal g, $\zeta > 0$ and $a \in I$, otherwise $\xi(s,a,g) = 0$. The optimistic initial values functionQ $\xi(s,a,g)$ bias the Q-values of the actions that point to the goal. Once the contextual Q-table is ready, it is necessary to find an optimal policy. To select the valid action given a state s, it is necessary to include the affordances function (4.6) by applying the Hadamard product operation:

$$a_t = \arg\max_a [(Q(s,a) + |\min_a Q(s,a)|) \odot \zeta(s,A)], \tag{4.8}$$

where a_t is the maximum possible action according to $\zeta(s,A)$, and $\forall a, s$, $Q(s,a) \neq 0$. Therefore, to update the Q-table by including the affordances function (4.6), the following equation is used:

$$Q(s,a) \longleftarrow Q(s,a) + \alpha[r + \gamma \max_{a'}[(Q(s',a') + |\min_{a'} Q(s',a')|) \odot \zeta(s',A) - \min_{a'} Q(s',a')]] \tag{4.9}$$

Algorithm 6 output is an optimal policy that produces a discrete set of actions, which is the trajectory that the robot must follow to reach the goal. These points are used to generate a discrete path. Since the robot is a continuous agent, it is necessary to smooth the discrete path by applying spline interpolation.

Algorithm 6 Contextual Q-learning.

Input : The goal g, and start position p
With equation (4.7), create a contextual Q-table
for n steps **do**
  if $\forall a, Q(s,a) = 0$
    Randomly select an action a_t
  else
    With probability ϵ select a valid action a_t with equation (4.8)
  Perform a_t and get reward r
  if terminal state
    $Q(s,a) \longleftarrow r$
  else
    Update Q-table with equation (4.9)
end for
Apply spline interpolation to the series of discrete actions of the optimal policy

4.2.3 Valid Policy Detector

Within each step of execution, changes in the environment are observed to decide if re-planning is required. Since the valid policy detector module is continuously observing the environment, when a change occurs (i.e., interruption of user hand or change in goal position), a negative reward will be returned such that the robot uses it as a trigger and reprograms itself using CQL to complete the manipulation task.

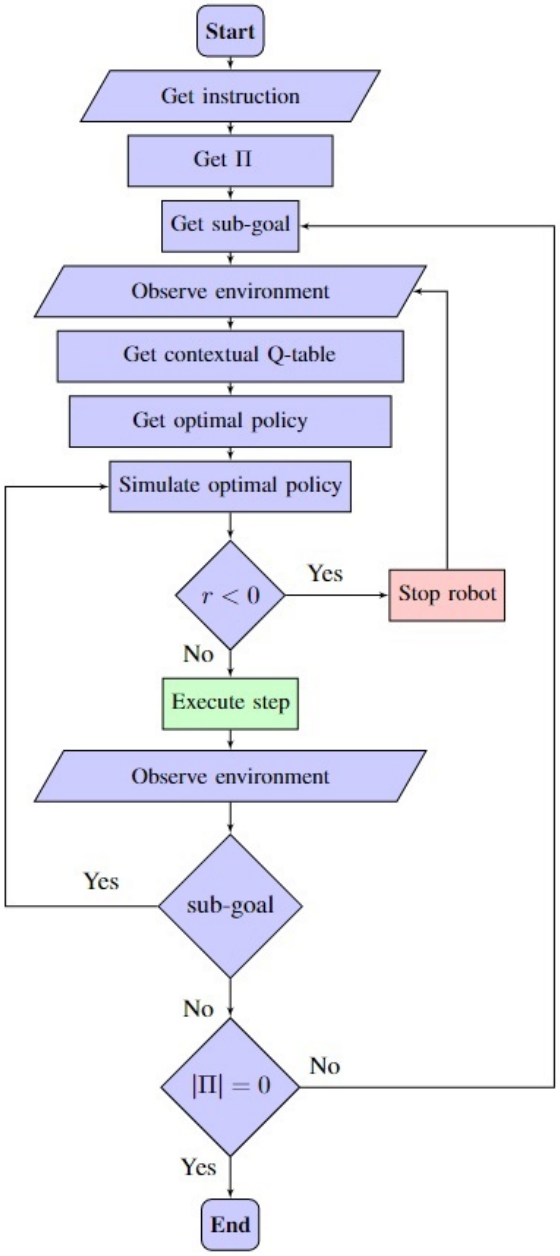


Figure 4.3: Flow chart of the HRI experiments with RL.

4.3 Experimental Setup

The problem domain to validate the framework empirically is an HRI scenario, where the user and robot manipulate objects. At the same time, the user provokes changes that interfere intermittently with the robot's actions. The experiment setup is comprised of a service robot (Care-O-Bot 4™), a table with a set of objects, and an Intel®RealSense™ camera mounted on the top of the table, as shown in Fig. 4.1.

The learning parameters used for CQL are $\alpha = 0.99$ and $\gamma = 0.05$ (it was found by trial and error that CQL converges faster when using these parameters). The following experiments were performed:

1. The performance of CQL is compared against the performance of QL, DQN, PPO, and A2C by solving the MDP for 100 objects' manipulation tasks with the robot. Every manipulation task is contained in a scenario configuration, each with a different initial position, goal destination, and different obstacles.

2. The user asks the robot for a certain task and puts pieces on the table while the robot executes the task.

3. A certain type of object is set as a sensible obstacle to test the robot's capacity to establish a safety perimeter around the obstacle.

4. The user moves the goal to test the robot's capacity to recognise a change in the environment that may lead to failing the current task.

5. The user obstructs the way of the robot with a hand to test the robot's capacity to react safely to the user's movements.

6. The framework is tested in a Kuka® LBR iiwa™ 14 robot.

Each experiment, in turn, aims to answer the following questions:

1. Does CQL perform better than QL, DQN, PPO, and A2C to suit the learning efficiency needed for HRI?

2. Can the robot understand the user's instructions and learn how to manipulate the objects on the table to complete the task using CQL?

3. Does CQL generate a series of optimal continuous actions that not only take an object from one place to another but also avoid collisions?

4. Can the simulation of the last optimal policy produce a negative reward trigger in real-time to re-plan the actions of the robot?

5. Can the robot safely react in real-time to a dynamic obstruction from the user?

6. Is the framework suitable for a different robot configuration?

For all the algorithms, the number of times the agent finds a solution for the 100 hundred manipulation tasks was counted, such that it is possible to calculate the proportion of successful attempts, which is referred to as the success rate [1].

4.4 Results

This section describes the results of the six experiments, providing an overview of the outcomes and insights garnered from each experiment. Fig. 4.4 shows a flowchart of the experiments.

4.4.1 Results of the First Experiment

In the first experiment, CQL, QL, DQN, PPO, and A2C were used for training over 100 different scenario configurations. For each scenario configuration, a new contextual Q-table is generated. This table contains all possible states given the current scenario configuration. A problem with Deep Reinforcement Learning (DRL) approaches is that they over-fit in simulation or the real world for a certain task, and it is difficult to make them work under different tasks or environments (Nguyen and La 2019). Hence, a change in the state related to perception or its configuration would affect the agent's performance. To avoid this issue, CQL design aims to generalise by learning from scratch given the current scenario configuration (Fig. 4.3). Despite changes in the environment, the algorithm always generates a new contextual Q-table for that scenario configuration. Once the contextual Q-table is solved, the set

[1] For a better understanding of the experimental setup, see: `https://youtu.be/raVeVjPv_Rc`

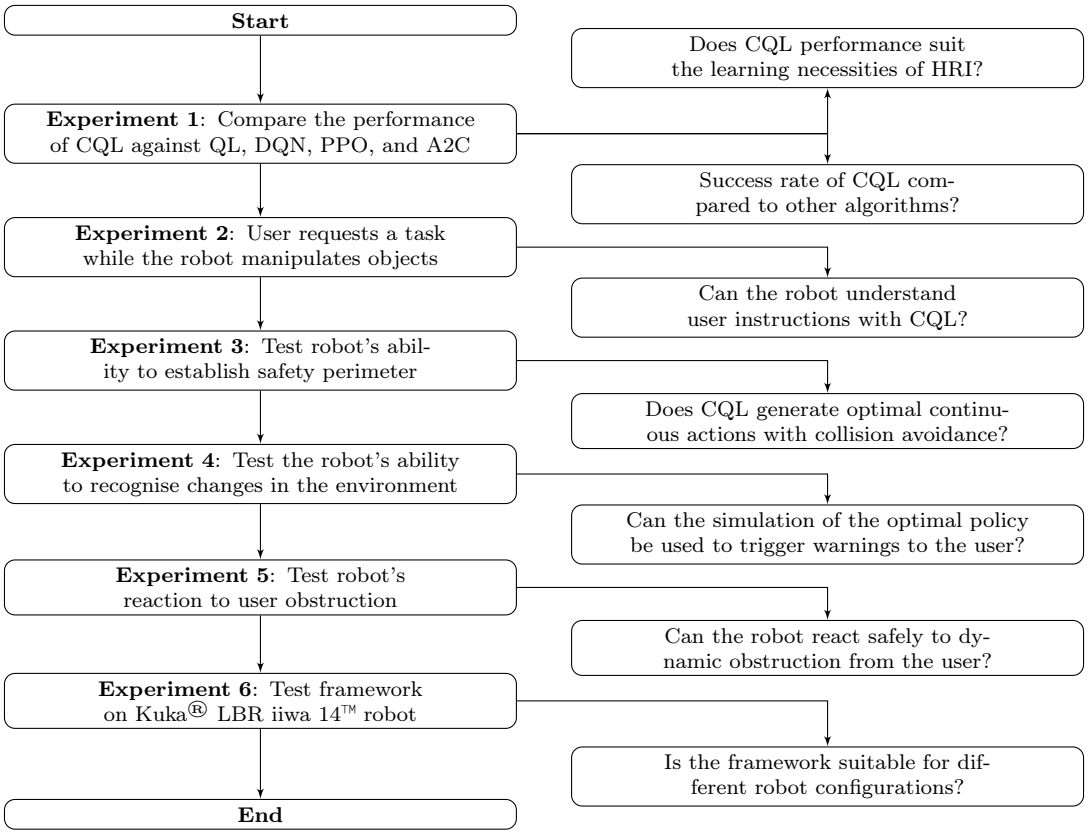


Figure 4.4: Flow chart of the experiments carried out to test different capabilities of the framework.

of actions can be safely transferred to the robot. In this context, the performances of CQL, QL, DQN, PPO, and A2C were compared. The CPU energy consumption of each algorithm is measured with the PyRAPL library (Rountree and Tsafrir n.d.).

Fig. 4.5 shows the learning curves of CQL, QL, DQN, PPO, and A2C after training over 100 different scenarios configurations. The results are summarised in Table 4.2. Where CQL had a success rate of 84%, took 67,631 training steps on average to converge in 2.7 seconds, and expended 61.76 J of energy. QL took 1.02 seconds on average to converge and consumed 26.9 J. However, the QL success rate is 38% and takes 67,631 steps on average to converge. DQN had the slowest learning rate, and it only succeeded in 17% of the scenarios with 39.9 seconds of learning time, an average number of steps to converge of 76,430, and an energy consumption of 886.83 J. Among the stable-baselines algorithms, PPO showed to be the best by succeeding in 68% of the cases in 50,098 average number of steps to converge.

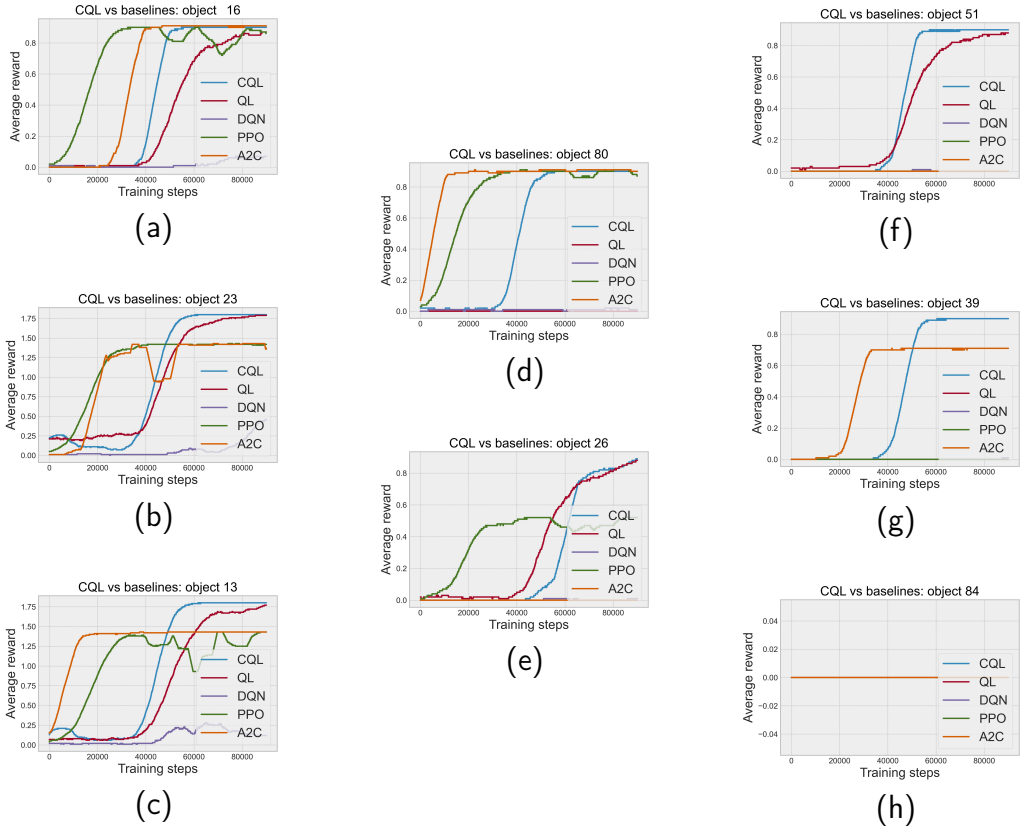


Figure 4.5: The learning curves above are eight out of 100 samples taken from the results of the experiments. All algorithms succeeded in (a) and (b), but DQN failed. In (c), QL and DQN fail to find a solution. In (d), A2C and DQN fail to find a solution while PPO and QL struggle to converge. In (e), only CQL and QL converge. In (f), CQL and A2C converge while the rest of the algorithms fail. In (g), PPO struggles to converge while CQL finds a solution. In (h), all the algorithms fail to find a solution.

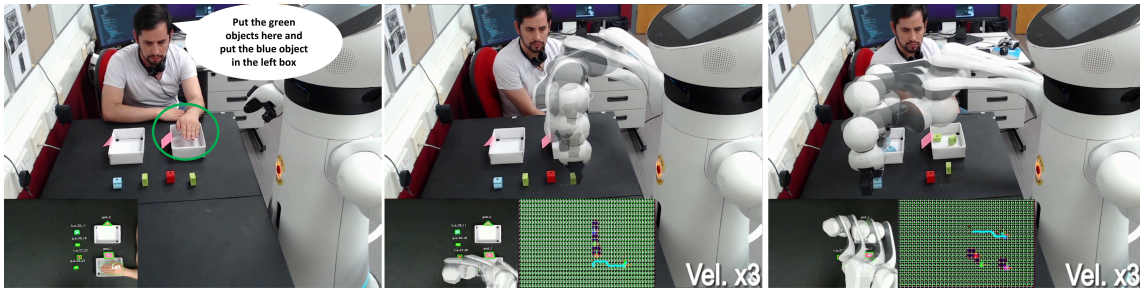


Figure 4.6: This figure shows a user speaking an instruction while pointing to the right box with his hand while the robot performs the instruction.

Nevertheless, PPO learning time is about 105 seconds and consumes 2662.49 J. In terms of less number of training steps to converge (Fig. 4.5c), A2C spends 30,915, but its success rate is only 38%. There were cases where any of the algorithms found

a solution, as shown in Fig. 4.5h. This experiment serves as evidence to show that CQL performs better than QL, DQN, PPO, and A2C to suit the learning efficiency needed for HRI.

Table 4.2: Results after running CQL, QL, DQN, PPO, and A2C in 100 different scenarios.

Algorithm	Success rate	Training time	Average number of steps to converge	Energy consumption
CQL	**85%**	2.7 s	60,103	61.76 J
QL	38%	**1.02** s	67,631	**26.9 J**
DQN	17%	36.9 s	76,430	886.83 J
PPO	68%	91.8 s	50,098	2122.52 J
A2C	39%	104.6 s	**30,915**	2662.49 J

The improvement in the performance of CQL compared to the baseline algorithms (Table 4.2) can be attributed to the reduction in exploration space based on the affordances. Furthermore, the effect of $\xi(s, a, g)$ for biasing the exploration towards the direction of the goal allowed CQL to finish the task faster than the baseline algorithms, which usually omit this information.

4.4.2 Results of the Second Experiment

The user provided visual and spoken instructions in the second experiment (Fig. 4.6). The tasks were divided into sub-goals, and a contextual Q-table was generated for each object that met the description of the instruction. For each contextual Q-table, CQL was applied to solve the MDP by obtaining a discrete optimal policy. Once the discrete policy was available, CQL applied spline interpolation to transform the discrete series of actions into a continuous one and send it to the Care-O-Bot 4™'s controller. Between every continuous action, the robot looked for relevant changes in the environment and could identify the extra pieces the user put on the table (Fig. 4.7). Therefore, the robot puts the objects into its corresponding box. This demonstration indicates that the robot understands the user's instructions and learns how to manipulate the objects on the table to complete the task using CQL.

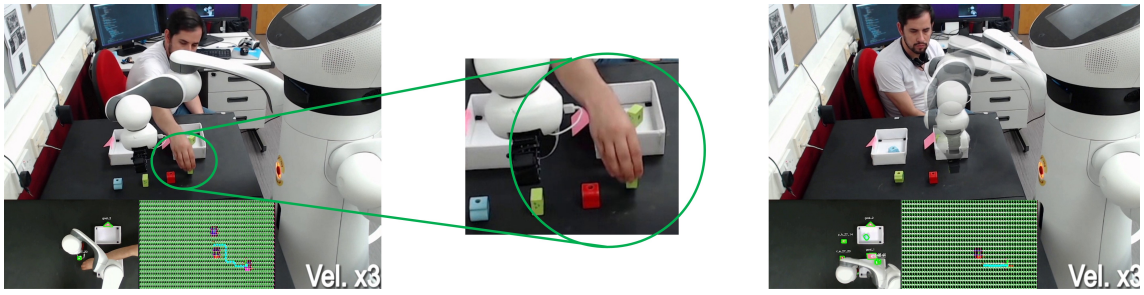


Figure 4.7: In this figure, the user puts extra objects on the table such that the robot identifies them and puts them into the box.

4.4.3 Results of the Third Experiment

In the third experiment (Fig. 4.8), the yellow objects were set as sensible obstacles such that CQL established a virtual security perimeter. The robot executed the series of continuous actions without colliding with the sensible obstacles. This demonstrated that CQL generates a series of optimal continuous actions that take objects from one place to another while avoiding collisions.

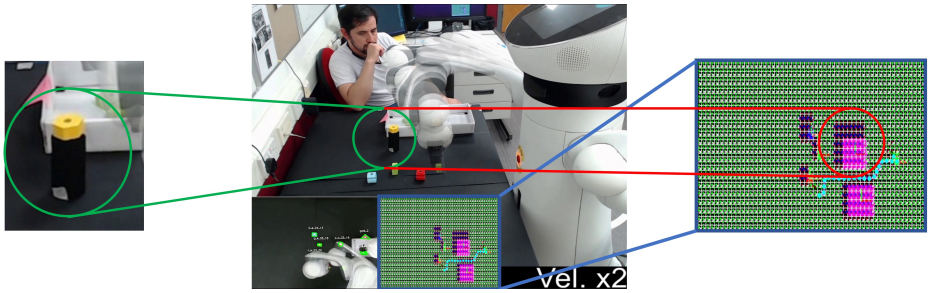


Figure 4.8: In this figure, the yellow objects are identified as sensible obstacles, and CQL adds a security perimeter while the robot successfully avoids those obstacles.

4.4.4 Results of the Fourth Experiment

In the fourth experiment (Fig. 4.9), the user moved the goal while the robot was holding an object. Consequently, the robot identified a relevant change in the environment by running the policy in the most recently observed environment and obtained a negative reward. In this case, the simulation of the policy dropped the object in a 0.0 (empty space) square instead of a 0.3 (goal) one. Hence, the robot could identify the type of error and react accordingly. The robot finished the task

by computing a new policy with CQL and dropping the object in the new goal. This demonstration shows that the simulation of the last optimal policy can produce a negative reward trigger in real time to re-plan the robot's actions.

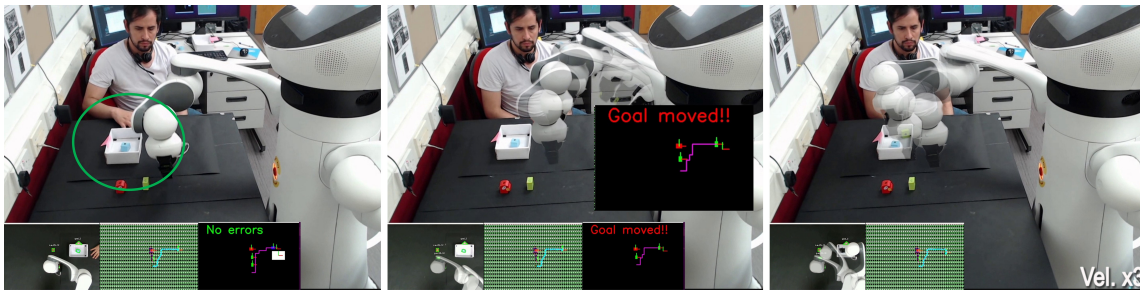


Figure 4.9: In this figure, the user moves the goal, and a negative reward trigger is returned such that the robot identifies that the user moved the goal and then re-planned its movements.

4.4.5 Results of the Fifth Experiment

In the fifth experiment (Fig. 4.10), the user placed his hand in the way of the robot. The robot identified this relevant change in the environment. In this experiment, the robot found a collision while simulating the set of discrete actions against a 0.4 (user's hand). The robot asked the user to be careful with his hand, and after the user moved his hand, the robot finished the task. Overall, the robot successfully reacts safely and in real-time to dynamic obstructions from the user.

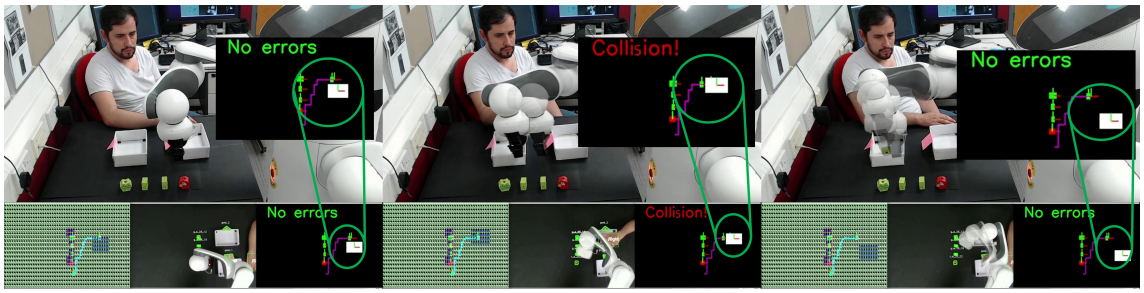


Figure 4.10: In this figure, the user puts a hand in the way of the robot's path. Consequently, a negative reward is returned, and the robot asks the user to move his hand.

4.4.6 Results of the Sixth Experiment

In the sixth experiment (Fig. 4.11), the framework was tested in a robot with a different configuration. Even though RL performs well in simulation, its implementation on real robots experiences several shortcomings, such as incomplete perception information and physical differences between the simulation and real-world scenarios. This may difficult the implementation of RL algorithms into different robot setups. One way to tackle this problem is by mixing real data with simulated data and training an agent with data from a buffer filled with transitions of synthetic and real-world images. This would allow the neural network to generalise better. Another possible solution is to define primitive actions and, in this way, reduce the risks that random exploration may cause in real-world setups. The use of several robots in parallel is also a well-known method to produce training data under real-world conditions.

However, using the framework proposed in this chapter, the Kuka® robot was able to carry out the same activities while showing the same capacities as Care-O-Bot 4™ during the experiments. To this end, the framework has been shown to be compatible with different robot setups. This highlights the novelty of this work, which has shown to be a solution to the drawbacks discussed before.

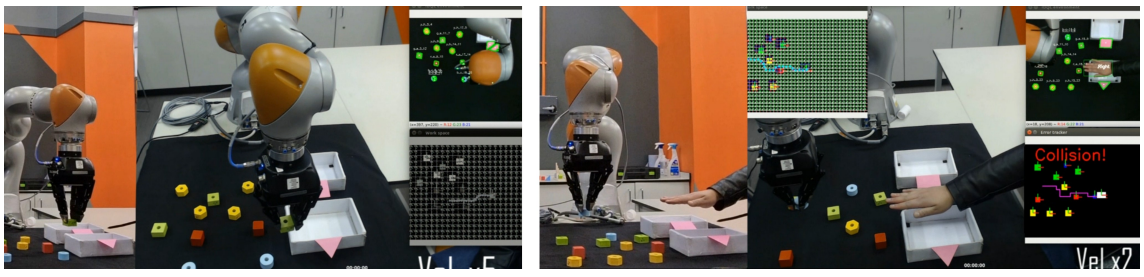


Figure 4.11: In this figure, the framework is tested in a different robot configuration.

4.5 Discussion

The outcomes of the experiments demonstrated the efficacy of CQL compared to baseline algorithms such as QL, DQN, PPO and A2C. The superior performance metrics presented in Table 4.2 of CQL CQL can be attributed to the reduction of the exploration space by the use of affordances as a filter of non-relevant actions. In this manner, the agent saves time, and at the same time, the algorithm biases the exploration towards the direction of the goal, which also reflects in reaching the goal faster than the baseline algorithms.

With the proposed framework, the robot showed an interesting "understanding" behaviour by following the instructions of the user and completing the tasks. This awareness of the robot is something that requires further investigation and improvement. Another important characteristic of the framework is that despite the robot configuration, it was still possible to implement it in a Kuka® robot. This shows that the framework can generalise, at least in terms of the robot being used. Nevertheless, further improvements are required, such as manipulating more complex objects and adding more robust object recognition strategies. These advancements would allow the framework to be used in more complex scenarios, such as repetitive processes in factories where a no robotics expert user can only teach the robot with spoken instructions and gestures, so the robot would not need a roboticist to reprogram it every time the process changes.

4.6 Summary

In this chapter, the affordance-based human-robot interaction framework was empirically validated. The experiments showed that the framework allows robots to understand instructions and execute manipulation during HRI based on an RL approach. This shows the impact of adding contextual information, such as semantics and affordances, to set initial optimistic values in a contextual Q-table. The 2.7

seconds of learning time allows CQL to generalise in different setups. Besides, it is possible to observe the environment while the robot executes a task within every step of execution. The robot using the framework was shown to be reliable while identifying relevant changes in the environment based on negative rewards and the semantic representation of the state.

The framework has the potential to be applied to navigation and collaborative robot problems and opens the door to robust real-time applications of RL learning in robotics. The current limitations of this chapter include tight shapes (e.g., sticks, pens, or pencils) that may lead CQL to create irregular squares to represent the states such that the agent's exploration could be affected, and the resulting set of actions may not be optimal. A future work direction would include the implementation of the framework in a more complex HRI scenario, including learning from the user.

Chapter 5

Learning to Bag with a Simulation-free Reinforcement Learning Framework for Robots

The frameworks presented in Chapter 3 and Chapter 4 demonstrated that the use of contextual information enhances the learning performance of Reinforcement Learning (RL) agents. However, in both previous chapters, it was possible to predict the next states, given the simulations and the nature of the rigid objects used in the Human-Robot Interaction (HRI) scenario. This chapter focuses on how to learn to manipulate a deformable object (a bag) in which the bag's next state is not easy to predict (i.e., if an action is taken, the physical configuration of the object changes drastically), difficult to model, and transferring from simulation to real world is a complex task. More concisely, the framework presented in this chapter allows robots to learn bagging in the real world, aiming to deal with the aforementioned shortcomings. The rest of this chapter is structured as follows. It begins with Section 5.1, motivating the problem of RL and manipulation of deformable objects. The problem formulation is described in Section 5.2. Then Section 5.3 formally introduces the framework, followed by Section 5.4, which describes the experimental setup. Sec-

tion 5.5 presents the results, while Section 5.6 discusses them. Finally, Section 5.7 concludes this chapter.

5.1 Introduction

Robots with human-level dexterity that can handle deformable objects may encourage a smoother integration of robots in daily activities. In practice, daily activities depend on more than manipulating rigid objects. In this context, the robots' capacity to manipulate deformable objects to operate in human environments is a necessity (J. Zhu et al. 2022). Among deformable objects, bags are used in several relevant tasks, such as transporting objects, packing, and shopping. Even though there have been studies on how to manipulate deformable objects, such as paper (Balkcom and Mason 2004, 2008; Elbrechter et al. 2011, 2012), fabrics (Borràs et al. 2020; Hoque et al. 2022; Jangir et al. 2020), ropes (Nair et al. 2017; Shi et al. 2022; Sundaresan et al. 2020), cables (Zhou et al. 2020; J. Zhu et al. 2019) and meat (Jørgensen et al. 2019), the problem of learning the *bagging* task is still underdeveloped.

Bagging is a complex task for robots because there exist challenges related to perception, occlusions, modelling of the bag's dynamics, ambiguity related to finding the opening, and how to grasp one or two layers of the bag depending on the current state of the task. RL has the potential to deal with the problems mentioned above. However, RL agents are commonly trained in simulation, which brings more challenges when implementing the agent in real-world tasks (Sharma et al. 2022). One challenge is that before simulating the bag, the model must be the most similar as possible to the real object (Yin et al. 2021). On the other hand, when switching from simulation to real-world, the agent must deal with occlusions, incomplete information, and noises. These are vital factors that make difficult the generalisation of RL (Nguyen and La 2019).

This chapter presents a new learning framework for robot-bagging tasks with

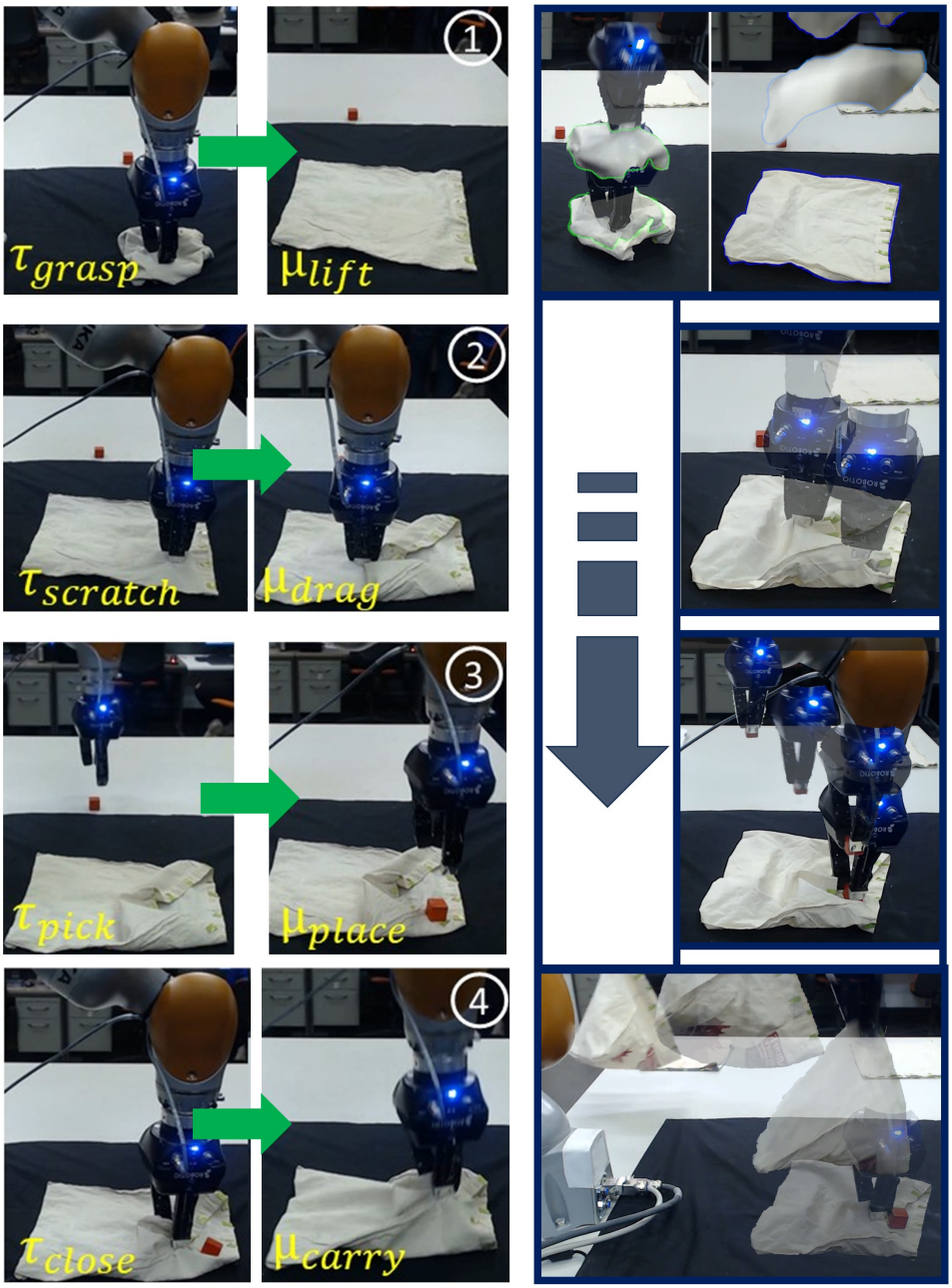


Figure 5.1: The robot, in four steps, performs the *bagging* task. In the first step, the robot unfolds the bag. In the second step, the bag is opened by the robot. The robot places the red cube in the bag's opening in the third step. In the fourth step, the robot carries the bag, completing the task.

compact state representations and primitive actions, aiming to efficiently train a robot to learn *bagging* in the real world[1]. The framework identifies five possible states and utilises eight primitive actions related to several grasping points on the bag. The task is solved in four steps (Fig. 5.1): unfolding, opening, placing the piece,

[1]Demo available at `https://youtu.be/3omyRbZJCZ0`

and carrying. The main contributions of this chapter are (i) a new RL algorithm is introduced, allowing robots to learn how to bag in the real world efficiently, (ii) a versatile state representation for the bagging task, and (iii) a framework that integrates reliable perception of the bag state and learning.

Symbol	Description
S	Set of states
A	Set of actions
Φ	Set of tuples of primary and complementary primitive action pairs
P_{s_j}	Set of pose points for primary primitive actions of s_j, where $s_j \in S$
D_{s_j}	Set of pose points for complementary primitive actions of s_j, where $s_j \in S$
Ψ	Set of rules
$\Lambda_{s_j,primary}$	Set of primary affordable actions for the state s_j
$\Lambda_{s_j,complementary}$	Set of complementary affordable actions for the state s_j
$R(s,a)$	Reward function
a	Action
s	State
A_{bag}	Area of the bag in pixels^2
A_{th}	Threshold of the bag in pixels^2
A_o	Area of the opening in pixels^2
A_{oth}	Threshold of the opening in pixels^2
A_{cube}	Area of the cube in pixels^2
$A_{b_{max}}$	Maximum area of the bag when it is unfolded in pixels^2
$A_{o_{max}}$	Maximum area of the bags' opening when is open in pixels^2
τ	Primary primitive action
μ	Complementary primitive action
τ_{grasp}	Grasping primitive action
$\tau_{scratch}$	Scratching primitive action
τ_{pick}	Picking primitive action
τ_{close}	Closing gripper primitive action
μ_{lift}	Lifting primitive action
μ_{drag}	Dragging primitive action
μ_{place}	Placing primitive action
μ_{carry}	Carrying primitive action
g	Number of pose points
ζ	Griding parameter
Π	State-action values table

Table 5.1: Nomenclature table for Chapter 5.

The problem domain to empirically validate the framework is conformed by a cotton bag and a red cube (Fig. 5.1). The framework first learns to perform the task through a different number of steps, as explained in Section 5.4. Then, the trained model is used to perform the task using two more bags with different sizes and positions to test the framework's generalisation capabilities. The nomenclature used in this chapter is summarised in Table 5.1.

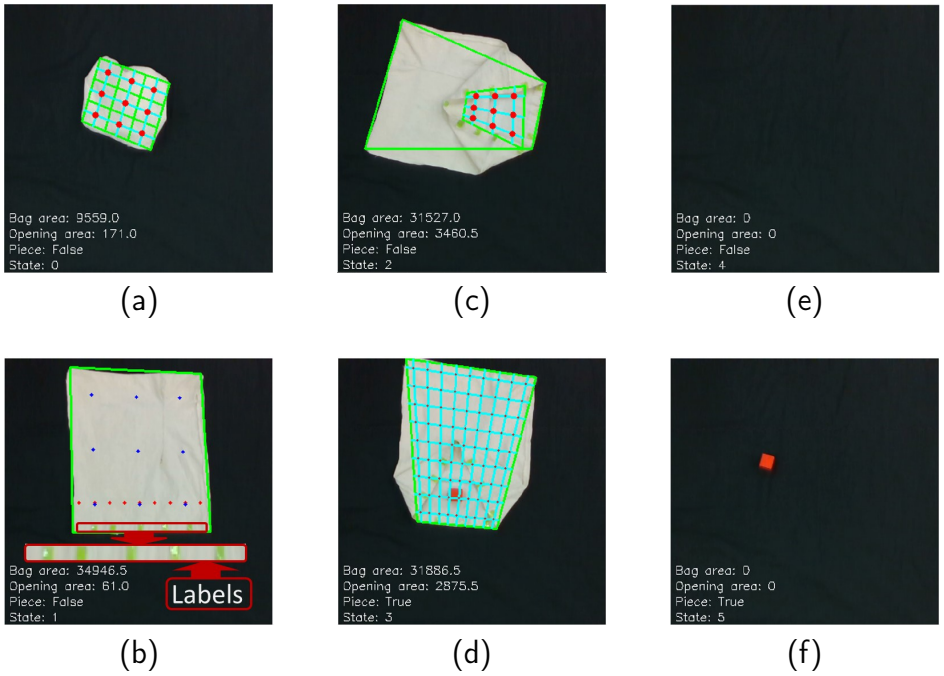


Figure 5.2: This figure illustrates the five states that conform the *bagging* task. The red and blue dots represent the grasping points the robot can select. In (a), the bag is folded such that its area is small, and the opening is not visible. In (b), the bag is unfolded, and the opening is visible. In (c), the bag's opening area is large enough to put an object inside. In (d), The object is in the bag's opening, distinguishing this state from the others. In (e), the task succeeded because no visible objects were left on the table, meaning the robot carried both the bag and the object. Lastly, (f) shows a failure case when the robot took the bag, but the red cube was still on the table.

5.2 Problem Formulation

The *bagging* task is formulated as a Markov Decision Process (MDP) (Sutton and Barto 1998) and is aimed to be solved by using a learning policy π. In an MDP, the agent executes a valid action a from the set of actions A in the current state s and transitions to a valid state s', where s and s' belong to the set of states S, according to the unknown dynamics of the bag. The environment provides a reward according to the reward function $R(s, a)$ upon transitioning to a new state. The set of states S is given by the following:

$$S = \{s_0, s_1, s_2, s_3, s_4, s_5\}, \tag{5.1}$$

where s_0 represents the folded bag, s_1 is the representation of the bag expanded, s_2 represents the bag opened, s_3 shows when the red object is on the bag, s_4 is the success state, and s_5 is the fail state. More specifically, the states are denoted as follows:

- For s_0: Folded bag.

  Condition: the bag's area is small, and the opening is not visible (Fig. 5.2a).

- For s_1: Expanded bag.

  Condition: The bag is unfolded, the opening is visible, and the green labels can be seen (Fig. 5.2b).

- For s_2: Opened bag.

  Condition: The opening area of the bag is large enough to accommodate an object (Fig. 5.2c).

- For s_3: Red object on the bag. The colour of the object helps the robot to distinguish it from the rest of the objects on the table.

  Condition: The object is in the opening of the bag, which distinguishes this state from the others (Fig. 5.2d).

- For s_4: Success state.

  Condition: No visible objects were left on the table, indicating that the robot carried both the bag and the object (Fig. 5.2e).

- For s_5: Fail state.

  Condition: The robot took the bag, but the red cube was still on the table (Fig. 5.2f).

Hence, the current state can be calculated as follows:

$$s = \begin{cases} s_0, & (A_{bag} < A_{th}) \wedge (A_o = 0) \wedge (A_{cube} = 0) \\ s_1, & (A_{bag} > A_{th}) \wedge (A_o > 0) \wedge (A_{cube} = 0) \\ s_2, & (A_{bag} > A_{th}) \wedge (A_o > A_{oth}) \wedge (A_{cube} = 0) \\ s_3, & (A_{bag} > A_{th}) \wedge (A_o > A_{oth}) \wedge (A_{cube} > 0) \\ s_4 & (A_{bag} = 0) \wedge (A_o = 0) \wedge (A_{cube} = 0) \\ s_5, & (A_{bag} = 0) \wedge (A_o = 0) \wedge (A_{cube} > 0), \end{cases} \tag{5.2}$$

where A_{bag} is the bag's area, A_{th} is a threshold value indicating a small area, A_o is the area of the opening, A_{oth} is the threshold value indicating a large enough opening and A_{cube} stands for the area of the cube. The robot can select among a set of primitive actions executed in pairs. Let:

$$\begin{aligned} \Phi = \{&\langle \tau_{grasp}, \mu_{lift} \rangle, \langle \tau_{scratch}, \mu_{drag} \rangle, \\ &\langle \tau_{pick}, \mu_{place} \rangle, \langle \tau_{close}, \mu_{carry} \rangle\}, \end{aligned} \tag{5.3}$$

be the set of tuples that contains the primitive actions τ that represents the primary primitive action executed by the robot and μ the complementary primitive actions (Fig. 5.4). The robot's primitive actions are defined as follows:

- **Grasping Action** τ_{grasp}: Involves the robot grasping two layers of the bag.
- **Lifting Action** μ_{lift}: Occurs when the bag is raised above the table and dropped to unfold it.
- **Scratching Action** $\tau_{scratch}$: Involves the robot grasping only one layer of the bag.
- **Dragging Action** μ_{drag}: Takes place when the robot moves a grasped layer

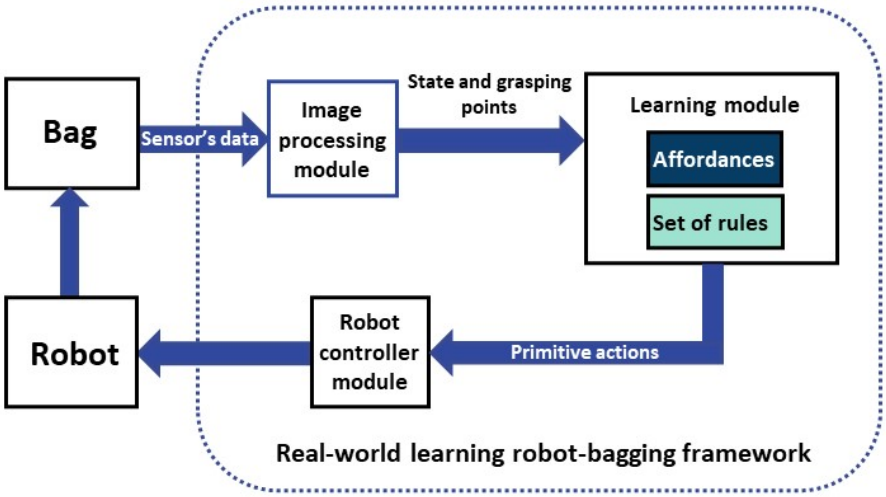


Figure 5.3: Proposed framework for learning to bag using RL.

point to a designated placing point.

- **Picking Action** τ_{pick} Refer to picking the red object.
- **Placing Action** μ_{place}: Places the red object.
- **Closing Action** τ_{close}: Grasps one layer of the bag when the object is placed in the opening.
- **Carrying Action** μ_{carry}: Lifts the bag to grab the object.

This chapter aims to find an optimal policy π^* that computes the correct sequence of primitive actions to transition through the states S till solving the bagging task.

5.3 Framework

This section presents a framework that aims to solve the problem of learning to bag in the real world with RL (Fig. 5.3). The framework comprises three modules: perception, learning and robot controller.

5.3.1 Perception

The perception module plays a crucial role in extracting relevant information from the bag, including its area, opening area, grasping points, and current state. It receives data from an Intel®RealSense™ camera, which provides RGB and depth images. The module utilises the OpenCV library to process this data. By filtering colours from the black background and obtaining the bag's contours, the module can calculate the A_{bag} value.

Algorithm 7 Opening area's calculator

Require: *points*: A list of 2D points
Ensure: A_o: The area of the opening

```
if CountPoints(points) < 3 then
    A_o ← 0
    return A_o
end if
triangles ← []
nodes ← CountPoints(points)
center ← GetCenter(points)
while True do
    pts ← points[]
    i, n ← FindClosestNode(center, pts)
    j, m ← FindClosestNode(n, pts)
    triangles.append([center, n, m])
    pts.pop(j)
    k, o ← FindClosestNode(n, pts)
    triangles.append([center, n, o])
    points.pop(i)
    if CountPoints(points) < 3 then
        break
    end if
end while
A_o ← 0
for each triangle in triangles do
    A_o = A_o + GetArea (triangle)
end for
return A_o
```

In order to assist with the automatic classification of bag states, green labels have been added around the bag's opening (refer to Fig. 5.2b). It is worth mentioning that perception is not the main focus of this work. Hence, the experiment configurations

were simplified in order to obtain the states from observations reliably. Specifically, green markers were used to facilitate the detection of the bag opening. Algorithm 7 is utilised to calculate the opening's area. This is achieved by providing an array of points containing the pair of coordinates obtained from the labels. Then through the generation of triangles, it is possible to sum all their areas and, in this way, obtain the opening's area A_o.

This information enables the module to determine whether the bag is on the table, folded, or unfolded. To differentiate the bag from the background, a black canvas is used as the background to make it easier to extract the bag, which is white. Similarly, the object for bagging is a cube in red. In brief, the primary functions of the perception module are:

- to determine the current state of the task;
- to provide pose points depending on the state;
- to measure the bag's current area; and
- to measure the current opening's area.

The perception module generates multiple pose points denoted by $g = \zeta^2$, where g represents the number of pose points and ζ is the griding parameter. For instance, Fig. 5.2 illustrates a configuration with $g = 9$ and $\zeta = 3$. Increasing the value of g results in more points available for grasping and lifting the bag, consequently leading to a larger set of actions to explore. Thus, the position of the grasping points can be determined.

The state s_0 (Fig. 5.2a) can be distinguished from the others because the opening is not visible, and the bag area is small compared to when it is unfolded. In this state, there are $g = 9$ grasping points (red dots in Fig. 5.2a) whose position can be obtained with the RealSense™ camera. Those points are stored in the set P_{s_0}. Before unfolding the bag and reaching the next state, the robot must explore which

one of these grasping points is the best to grasp the bag, lifting it at a given height stored in the D_{s_0} set and dropping it.

In the state s_1 (Fig. 5.2b), the opening is visible, and the bag's area has reached a feasible value for opening the bag. Besides, since the opening area is small, s_1 can be distinguished from the other states based on the criteria of the opening area. In this state, the perception module provides $g = 9$ grasping points close to the opening (red dots in Fig. 5.2b) which are stored in the set P_{s_1}, and g placing points (blue dots in Fig. 5.2b) stored in the set D_{s_1}. Consequently, the robot's goal in this state is to find which combination of actions over the grasping point maximises the opening area.

In the state s_2 (Fig. 5.2c), the opening's area serves as a trigger to identify this state. First, the perception module stores the object's position to be bagged in P_{s_2}. Then, the perception module sets $g = 9$ placing points (red dots in Fig. 5.2c) and stores them in the set D_{s_2}. The robot can place the object to be bagged on one of those placing points and get a reward depending on the closeness with the centre of the bag's opening.

In the state s_3 (Fig. 5.2d), the red object is placed in the opening. In this state, the robot is required to explore more grasping points. For this reason, the number of pose points is $g = 81$ (red dots in Fig. 5.2d). Then, the pose points are stored in the set P_{s_4} while the lifting poses corresponding to the complementary primitive action in this state are stored in the set D_{s_4}. Therefore, the robot can interact with the bag and decipher which grasping point allows it to finish the task. When the robot lifts the bag, and the module can not detect any object left, as shown in Fig. 5.2d, the state s_4 can be identified, and the task is finished. On the other hand, when there are still objects on the table, the module identifies the s_5 state, which is a failed attempt to bag the object (Fig. 5.2f).

5.3.2 Learning

The learning module seeks to find the optimal combination of grasping points on the bag and primitive actions from the robot. Additionally, this subsection aims to motivate the problems that experience current RL approaches and explain the functionality principle of the Π-learning algorithm (Table 5.2 summarises the main similarities and differences between Q-learning (QL) and Π-learning). Additionally, as previously described, this work proposes the use of several primitive actions, meaning that the action space is discrete.

Table 5.2: Differences between QL and Π-learning.

Aspect	**Q-learning**	**Π-learning**
Exploration Space	Explores all possible action-state pairs.	Reduces exploration space by defining rules and pairs of primitive actions.
Exploration Strategy	Uses ϵ-greedy policy.	Besides the ϵ-greedy polity, utilises affordances and predefined rules for action selection.
Next State Dependence	Dependent on the next state for updating the Q-values.	Independent of the next state for updating the Π-values.
State Transition Handling	May result in instability due to the state transitions of the bag.	Designed to deal with the state transitions of the bag.
Table Initialisation	Initialises Q-values to zeros.	Initialises Π-values to zeros.
Update Rule	Updates Q-values based on next state information.	Updates Π-values independently of the next state.
Optimal Policy Extraction	Extracts optimal policy based on the maximum Q-value.	Extracts optimal policy based on maximum Π-value.

In the context of state-of-the-art approaches, despite the popularity of QL and its successful implementation in multiple fields, when it comes to the problem defined in section 5.2, there exist several drawbacks. The first one is related to the size of the exploration space, where considering eight primitive actions, 81 pose points, and four possible states in the case of Fig. 5.2, it would be necessary to explore a total of 2592 primitive action-pose point pairs. This approach lacks practical significance because it would involve a long training time in the real world.

Moreover, the dependency of QL on the next state value would also involve significant exploration to achieve stable convergence. This is because of the dynamics

of the bag, which despite executing the best action, it would take repeating the same action until the bag's state transitions. More specifically, while for the best action a given the state s, the bag may transition to s', it may also stay in the same state s due to the manner the problem was defined (see Eq. (5.2)) and for the characteristics of the environment, which also include failures in the real world that could contaminate the training.

The following is proposed to solve the drawbacks. First, the exploration space is reduced by implementing affordances that define what primitive actions are suitable given the current state. The affordances are obtained from a manually defined set of rules Ψ. Then, the robot executes actions with probability ϵ, also known as ϵ-greedy policy (the value of ϵ controls the exploration of the environment), and gets a reward. The process is repeated for n steps till the training is completed. Let:

$$\begin{aligned}\Psi = \{&\langle s_0, \tau_{grasp}, \mu_{lift}\rangle, \langle s_1, \tau_{scratch}, \mu_{drag}\rangle,\\ &\langle s_2, \tau_{pick}, \mu_{place}\rangle, \langle s_3, \tau_{grasp}, \mu_{carry}\rangle\},\end{aligned} \tag{5.4}$$

be the set of rules that contains the tuples in which each state s is related to its valid actions. Then, the primary affordable actions of the states are:

$$\Lambda_{s_j,primary} = \Psi^2 \times P_{s_j}, \tag{5.5}$$

where $s_j \in \Psi^1$ and $j \in (0, |S|)$. The affordable complementary actions are given by:

$$\Lambda_{s_j,complementary} = \Psi^3 \times D_{s_j}, \tag{5.6}$$

where $s_j \in \Psi^1$. The set of affordable actions pairs is given by:

$$A = \Lambda_{s_j,primary} \times \Lambda s_j, complementary \tag{5.7}$$

Aiming to avoid confusion with the QL notation algorithm, Π is utilised. Al-

gorithm 8 has a function that calculates the state-action value Π, representing the quality value of an action a given a state s:

$$\Pi : S \times A \longrightarrow \mathbb{R} \tag{5.8}$$

At the beginning of the learning, all Π values are initialised to zero and stored in a Π-table. During the training process, it is updated with the following:

$$\Pi(s,a) \longleftarrow \frac{\Pi(s,a) + R(s,a)}{m}, \tag{5.9}$$

where m is the number of times that action-state pair $\Pi(s,a)$ has been selected by the agent, $m > 0$ and $a \in A$. The reward function used for the bagging task is given by the following:

$$R(s,a) = \begin{cases} \frac{A_{b_{max}}}{A_{bag}}, & s = s_0 \text{ or } s = s_1 \\ \frac{Ao_{max}}{A_o}, & s = s_2 \\ 1, & s = s_3, \text{ and the object is} \\ & \text{at the centre of the opening} \\ 1, & s = s_4 \\ -0.1 & s = s_5 \\ 0, & \text{otherwise,} \end{cases} \tag{5.10}$$

where $A_{b_{max}}$ is the maximum area of the bag when it is unfolded, A_{bag} is the area of the bag after executing an action, Ao_{max} is the maximum area that the bag's opening can reach, and A_o is the bag's opening area after executing an action.

In contrast to classical QL (Watkins and Dayan 1992b), which would require exploring a total of 2592 primitive action-pose point pairs (considering 8 primitive actions, 81 pose points, and 4 possible states in the case of Fig. 5.2), Π-learning is tailored to handle these conditions, particularly in the context of the bagging task. This is because many actions do not result in state changes due to factors such as incomplete unfolding in state 0 or failure to open the bag in state 1. As a result, the robot must repeat the same action until a transition occurs. To this end, the optimal policy is extracted from $\Pi(s, a)$ with:

$$\pi^*(s) = \arg\max_a [\Pi(s, a)] \tag{5.11}$$

Unlike QL, Π-learning incorporates equation (5.9), which is independent of the next state. It also reduces the exploration space by defining rules and pairs of primitive actions using equation (5.7). For instance, in the scenario depicted in Fig. 5.2, QL would require exploring 2592 actions, while Π-learning would only necessitate exploring 81 actions. This significant reduction in exploration space aims to reduce training time for Π-learning. Furthermore, the Π-table remains unaffected by the next state. This is crucial because the bag assumes different shapes after each action and may or may not transition into the next state. This behaviour may cause instability for QL because of its dependence on the next state information. Consequently, the perception module detects state transitions and allows Π-learning to concentrate on maximising the reward solely based on the current state.

Algorithm 8 takes the number of training steps n and the set of actions A as the input. First, a Π-table is generated. Then, after n steps of training, the algorithm returns the optimal policy $\pi^*(s)$. The Π-learning algorithm is specifically designed to enable learning with a reduced number of states while dealing with a wide range of actions.

Algorithm 8 Π-learning.

Require: Training steps n, set of actions A
Ensure: Optimal policy $\pi^*(s)$
1: Initialise a Π-table with zeros
2: **for** n steps **do**
3: With probability ϵ, select a valid action a from A
4: Perform a and calculate the reward with Eq. (5.10)
5: Update Π-table with Eq. (5.9)
6: **end for**
7: Extract the optimal policy from the Π-table with Equation (5.11)

5.3.3 Robot Controller

The robot controller module coordinates the robot's continuous actions in a precise manner. To accomplish this, the module inputs any primitive action and translates it into continuous actions that the robot can handle. Since the perception module provides the grasping points of the bag, and the learning module generates a sequence of primitive actions given a state, the robot controller module can make the robot interact with the environment. This information is managed through the Robot Operating System (ROS).

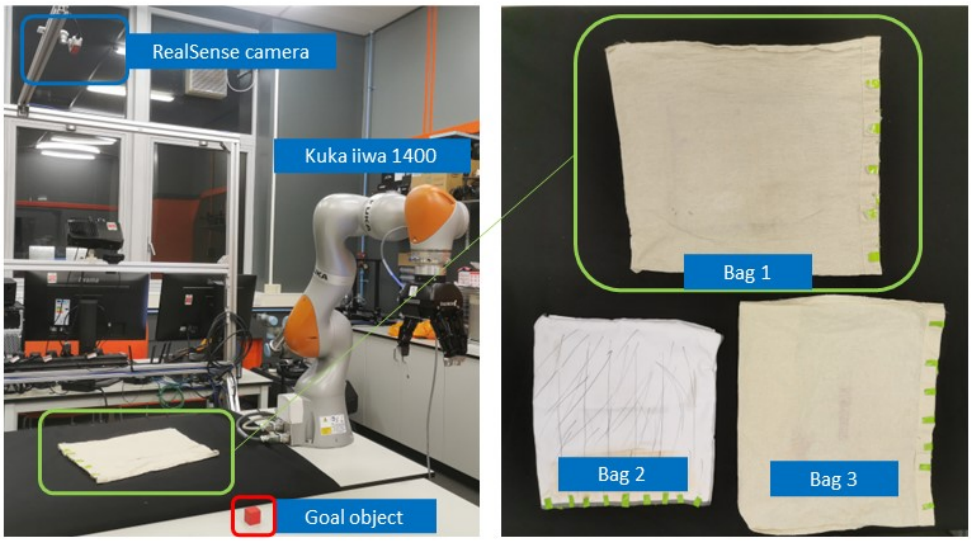


Figure 5.4: The left side of the figure illustrates the experimental setup comprising an object to be bagged (red cube), a Kuka® iiwa 1400™ robot, and an Intel® RealSense™. On the left side are the three bags used during the experiments.

5.3.4 Bagging task implementation

This subsection aims to provide a more exhaustive description of the bagging task, focusing on how primitive actions are implemented and how positions are determined within the perception module. The following is a point-by-point description of the processes that allow the robot to perform the bagging task:

- **The primitive actions** are steps taken by the robot to interact with the bag and achieve the overall task. The primitive actions are hard-coded commands and depend on the perception module's output, which interprets data from an Intel® RealSense™ camera and provides a set of possible grasping points. These actions are available as services running under ROS.

- **The grasping points** are necessary for the robot to securely hold the bag, depending on the current state. The perception module generates multiple pose points denoted by $g = \zeta^2$, where g represents the number of pose points and ζ is the griding parameter. These points are automatically positioned around the bag's opening and body. If more grasping points are required, the value of ζ must be increased. Depending on the state of the bag given by the perception module, the framework calls a ROS service that corresponds with the current state and feeds the grasping points as parameters.

- **The position determination of grasping points** is important for the execution of primitive actions. The algorithm presented in Algorithm 7 calculates the opening area A_o by generating triangles from an array of points obtained from green labels around the bag's opening. This information is used to determine the positions for grasping and placing points. When it comes to the body of the bag, the perception module retrieves the bag's body as a 4-sided geometry with sides F, G, H, and I (refer to Fig. 5.5). The objective is to evenly distribute ζ^2 points inside the described geometry based on the reference point (x_0, y_0), corresponding to the upper-right corner of the 4-sided

geometry surrounding the bag. Each side of the geometry can be expressed as follows:

1. Side F represented by the coordinates (p_{11}, p_{21}) and (p_{31}, p_{41}).
2. Side G represented by the coordinates (p_{12}, p_{22}) and (p_{32}, p_{42}).
3. Side H represented by the coordinates (p_{13}, p_{23}) and (p_{33}, p_{43}).
4. Side I represented by the coordinates (p_{14}, p_{24}) and (p_{34}, p_{44}).

The reference point (x_0, y_0) corresponds to the upper-right corner of the 4-sided geometry. The objective is to distribute ζ^2 points inside this geometry evenly. Hence, the i-th point on sides F, G, H, and Ican be represented as:

$$(x_i, y_i) = \left(x_0 + \frac{i}{\zeta} \cdot (p_{n1} - p_{n-3}), y_0 + \frac{i}{\zeta} \cdot (p_{n+1} - p_{n-1}) \right) \tag{5.12}$$

Here, i ranges from 0 to $\zeta - 1$, (x_i, y_i) represents the evenly distributed points and n represents the side of the 4-sided figure. If n is equal to 1, it corresponds to side F. If n equals 2, it corresponds to side G, and so on. With these points, a grid can be generated inside the geometry. Hence, the grasping points are at the centre of each grid cell.

The implementation of primitive actions relies on the perception module's analysis of RGB and depth images from the RealSense camera. Algorithm 7 contributes to determining the bag's opening area, and Eq. (5.12) determines the evenly distributed points on the sides of the 4-sided geometry for the body of the bag. With these points, a grid can be generated inside the geometry. Hence, the grasping points are at the centre of each grid cell. This integration of perception and primitive actions defined as services in ROS allows the robot to execute the bagging task defined in this paper[2].

[2]For further technical details, the reader is encouraged to visit: `https://github.com/FranciscoMunguiaGaleano/LearningToBag`

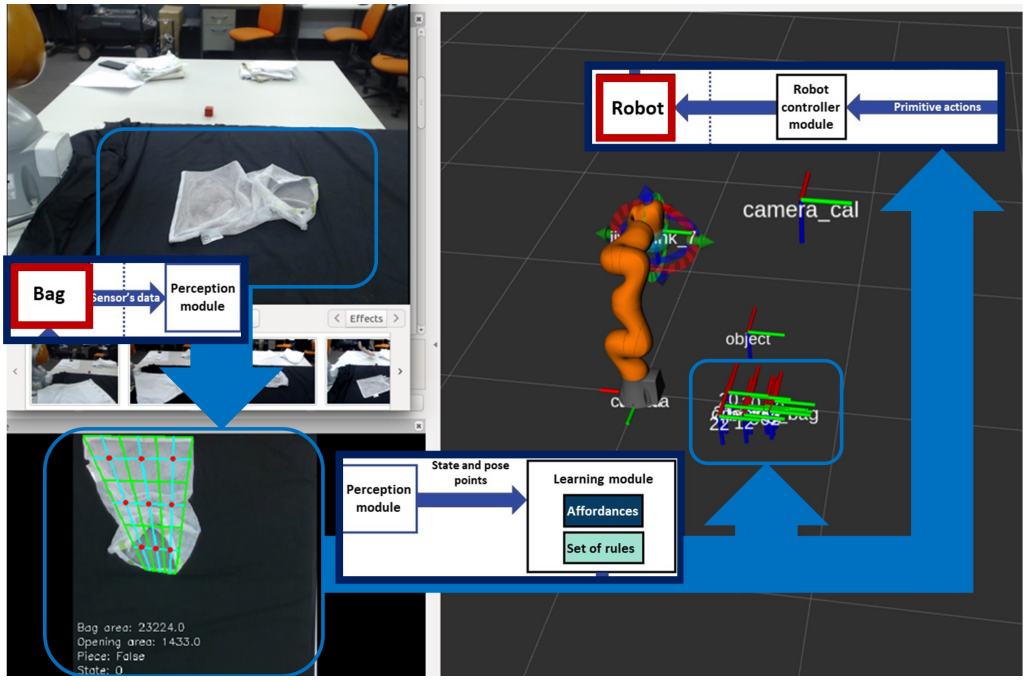


Figure 5.5: Implementation of the real-world learning robot-bagging framework.

5.4 Experimental Setup

The experimental setup to empirically validate the framework incorporates three bags (Table 5.3), the object to be bagged (red cube), a Kuka® iiwa 1400™ robot, and an Intel® RealSense™ camera (Fig. 5.4). The task's goal is to train the robot to learn to bag the object (red cube). The experiments start by training several agents to learn how to handle "Bag 1" in the real world for 10, 30, 50, and 100 training steps for each state of the task (unfolding, opening, placing the piece, and carrying), totalling 40, 120, 200 and 400 total training steps, respectively. Then, the performance of our framework is compared against the following state-of-the-art algorithms performance by using their implementations from the stable-baselines (Hill et al. 2018): Deep Q-network (DQN) (Z. Wang et al. 2016) and Asynchronous Actor-Critic (A2C) (Mnih et al. 2016) two robust and well-tested algorithms for discrete action spaces. After the training is completed, 10 attempts are executed for each agent, and the count of successful completion of the bagging task for each attempt is recorded. Additionally, 10 attempts are performed to calculate the algorithms' success rate when starting

from the unfolding and opening steps. The preceding experiment aims to discover if the framework can learn better than the baseline algorithms.

Table 5.3: Characteristics and parameters of the bags used in the experiments. The values of A_{th}, A_{oth}, $A_{b_{max}}$, $A_{o_{max}}$ are measured in pixels^2.

Name	**Opening length**	**Bag width**	**Material**	A_{th}	A_{oth}	$A_{b_{max}}$	$A_{o_{max}}$
Bag 1	30 cm	35 cm	Cotton	25000	150	34000	3900
Bag 2	25 cm	25 cm	Polyester	18000	50	28000	3200
Bag 3	33 cm	26 cm	Cotton	25000	150	34000	3900

The trained agent with the highest success rate is used to test the generalisation capacities of the framework. To evaluate the framework's proficiency in handling the task from a different starting position, the position and orientation of "Bag 1" were changed twice. Then, the framework is tested on "Bag 2" and "Bag 3" for 10 attempts.

5.5 Results

This section presents the results of the experiments. Firstly, the learning progress of the framework and the state-of-the-art algorithms for each step of the task (unfolding, opening, placing the piece, and carrying) is illustrated in Fig. 5.6. Secondly, Table 5.4 shows the total reward obtained by each algorithm, followed by Table 5.5, which contains the reward obtained by all the approaches for each step of the bagging task. Table 5.6 shows the success rates from performing the bagging task 10 times with "Bag 1" for each step of the task, as well as the success rates starting from step 1 (opening) and from step 2 (unfolding). Then, Fig. 5.7 shows the robot performing the task in different initial positions with "Bag 1". Fig. 5.8 illustrates the robot performing the bagging task with "Bag 2" and 'Bag 3". Lastly, the success rates of handling all the bags are summarised in Table 5.7.

The learning curves in Fig. 5.6a show the progress of the agent learning to unfold the bag, which consists of selecting a grasping point, lifting the bag, and dropping it

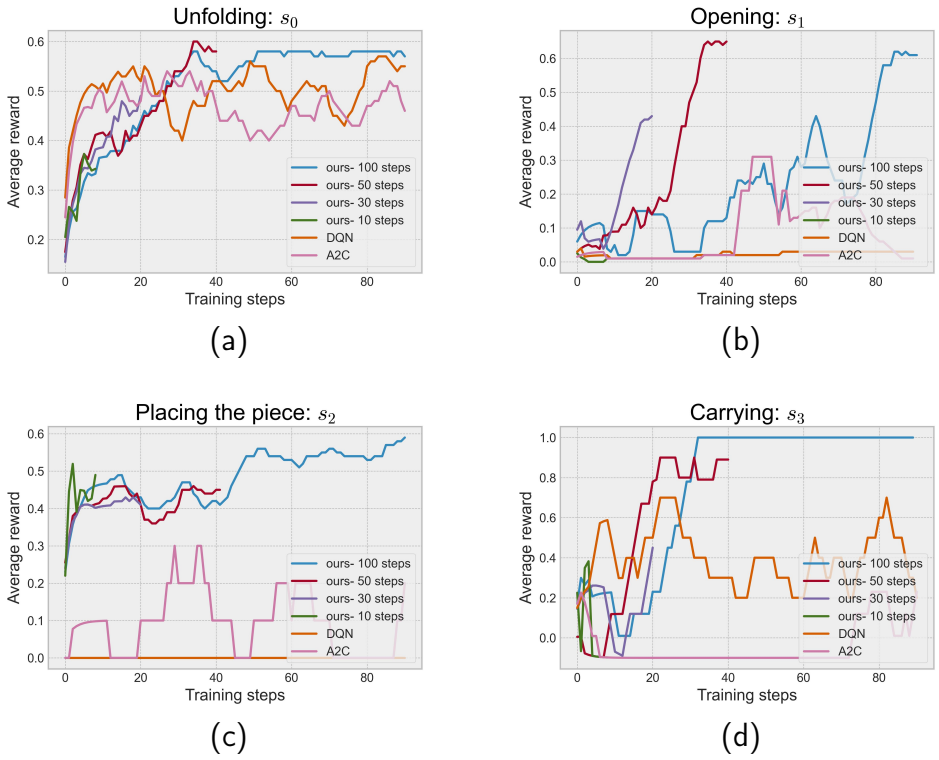


Figure 5.6: The learning curves above display the results of the experiments. First, (a) shows the progress of the learning curve of the unfolding step, demonstrating that the approach converges after 100 training steps while DQN and A2C struggle to do so. Then, in (b), the proposed framework was the only one to converge after training for 100 and 50 training steps. In (c), A2C and DQN failed to find a solution while the approach converged. Lastly, in (d), the approach trained for 100 steps and converged to the highest value.

till it is unfolded. The framework running for 100 training steps converged in around 50 training steps, while DQN and A2C demonstrated an unstable learning behaviour by struggling to converge. The framework running 10 and 30 training steps shows that the agent requires more exploration. Contrary, the framework learning for 50 training steps indicates that 50 is the minimum number of training steps required to find the best grasping and lifting positions for our approach.

The learning curves in Fig. 5.6b illustrate the learning progress of the agents for the opening step, which involves grasping one layer of the bag and dragging it to another point of the bag. The reward during this step is equal to the total area of the opening. The framework that ran for 100 training steps converged in approximately 80 training steps, while the one that ran for 50 training steps found the best solution

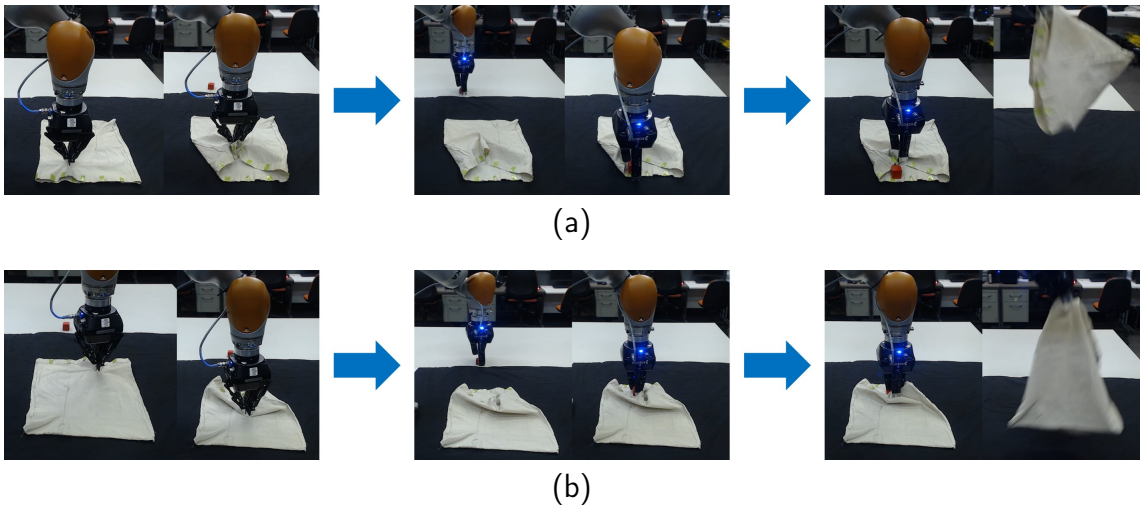


Figure 5.7: The robot performing the bagging task with two different bags. In (a), the bag's opening faces the camera's view. In (b), the bag's opening is facing the opposite direction of the camera's view. The robot successfully completed the tasks in both cases with different orientations of the bag.

in around 40 training steps. This is due to the stochastic nature of the exploration, which randomly found a better action in an early stage of the learning process for the 50 training steps experiment. DQN and A2C fell into a local minimum, and their learning could not progress. Our framework running for 10 and 30 training steps could not explore the environment enough to find an optimal policy.

The learning curves in Fig. 5.6c show the progress of the agent learning to place the goal object (red cube) in the bag's opening such that the closer the robot places the goal object, the higher the reward. The framework running for 10, 30, 50, and 100 training steps converged faster than the previous steps because this step is more straightforward than the previous ones by only including nine placing points in the bag's opening. The framework running for 10, 30, 50, and 100 could converge, while DQN and A2C could not find a solution.

The learning curves in Fig. 5.6d show the learning progress of the agents for the carrying task. In this step, if the robot executes the carrying action and the red cube is not on the table after that, the reward is equal to 1 and -0.1 otherwise. The framework that runs for 50 and 100 training steps could converge to a stable plateau

Table 5.4: Total reward obtained by the framework and the stable-baselines after training.

Approach	Total reward	Training time	Total training steps
Ours (100)	1.9608	173 min.	400 (100 for each step of the task)
Ours (50)	1.7000	95 min.	200 (50 for each step of the task)
Ours (30)	0.9900	62 min.	120 (30 for each step of the task)
Ours (10)	0.2475	18 min.	40 (10 for each step of the task)
DQN	1.0300	198 min.	400 (100 for each step of the task)
A2C	0.6488	186 min.	400 (100 for each step of the task)

while running for 10 and 30 could not result in the right grasping point for carrying the bag. DQN fell into a local minimum and could not find a solution. A2C could find a better grasping point to carry the bag. However, the learning curve shows that A2C struggles to converge.

Table 5.5: Reward obtained by the framework and the stable-baselines after training, categorised per step.

Approach	Reward step 1	Reward step 2	Reward step 3	Reward step 4
Ours (100)	0.500	0.241	0.490	0.730
Ours (50)	0.460	0.280	0.440	0.520
Ours (30)	0.390	0.248	0.430	0.220
Ours (10)	0.366	0.014	0.500	0.110
DQN	0.510	0.020	0.120	0.380
A2C	0.470	0.070	0.100	0.009

The success rates of all the approaches are summarised in Table 5.7, in which 10 attempts were performed for each step of the bagging task. For step 1, Ours(100) and Ours(50) reached the highest success rate with 70%, followed by DQN with 40%. The lowest success rates were achieved by Ours(30) and A2C with 0% and 20%, respectively. For step 2, A2C, DQN, and Ours(10) presented the lowest success rates, which demonstrates that step 2 is the most difficult to learn. For step 3, all the approaches reached a success rate equal to or superior to 90%, which demonstrates that this step is the easiest to learn. In the last step, Ours(10) had the lowest performance with 60% while the rest of the approaches reached a success rate equal to or superior to 90%. Additionally, 10 attempts were carried out from step 1 (unfolding) and step 2 (opening), which for Ours(100) resulted in 60% and 80% of

success rates, respectively. It can be observed that the low success rates of Ours(10), DQN, and A2C are because of getting stuck on step 2. Moreover, the difficulty increases when the task is started from step 1, which reflects in the performance of all the agents.

Table 5.6: Success rate of the framework and the stable-baselines after training.

Experiment "bag 1"	**Step 1**	**Step 2**	**Step 3**	**Step 4**	**Success rate from Step 1**	**Success rate from Step 2**
Ours (100)	7/10	9/10	9/10	10/10	6/10	8/10
Ours (50)	7/10	7/10	9/10	10/10	5/10	6/10
Ours (30)	2/10	4/10	10/10	10/10	1/10	4/10
Ours (10)	0/10	0/10	10/10	6/10	0/10	0/10
DQN	4/10	1/10	10/10	9/10	0/10	2/10
A2C	2/10	1/10	10/10	10/10	0/10	1/10

The total average reward obtained by all the approaches is summarised in Table 5.4, where the three approaches that collected the highest rewards are Ours(100), Ours(50), and DQN with 1.9608, 1.7 and 1.03, respectively. When it comes to the total average reward collected for each step, Table 5.5 shows that for step 1, DQN collected the highest reward of 0.51, followed by Ours(100), which collected 0.5. For step 2, Ours(50) collected the highest reward of 0.28, followed by Ours(100) with 0.241. In step 3, the highest reward was obtained by Ours(100). For step 4, the highest reward was obtained by Ours(100), followed by Ours(50) and DQN. In general, the stable-baselines DQN and A2C presented problems learning step 2 (opening), while the rewards collected for all the approaches in step 1 are almost the same.

Table 5.7: Success rate of the framework and stable-baselines after training per step.

Experiment	**Step 1**	**Step 2**	**Step 3**	**Step 4**	**Success rate from Step 1**	**Success rate from Step 2**
Bag 1	6/10	8/10	9/10	9/10	6/10	8/10
Bag 2	4/10	6/10	10/10	8/10	2/10	5/10
Bag 3	5/10	8/10	10/10	9/10	3/10	7/10

The last experiment tested the generalisation capabilities of the framework. The

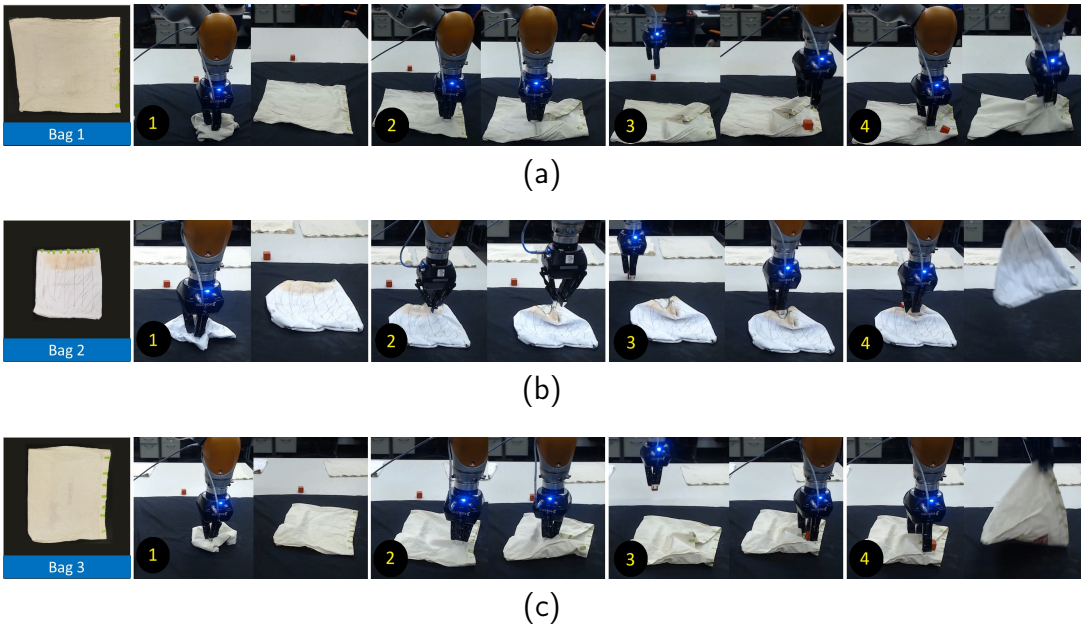


Figure 5.8: In (a), the robot performs the *bagging* task with "Bag 1", used for training with the framework. In (b), the robot using the framework after training performs the *bagging* task with "Bag 2", a smaller bag made of polyester. In (c), the robot using the framework after training performs the *bagging* task with "Bag 3", a bag made of cotton and also with a different size of "Bag 1".

success rates are summarised in Table 5.6. Fig. 5.7 illustrates the robot performing the bagging task with "Bag 1" starting from two different positions and orientations. Despite the change in the initial position, the framework was capable of finishing the task. This is because the proposed approach focuses on the state and grasping points with respect to the bag, which is independent of the pose of the bag in the global workspace frame. Fig. 5.8a shows the robot performing the bagging task with "Bag 2", which had a success rate of 20% when starting from step 1 and 50% when starting from step 2. The robot performing the bagging task with "Bag 3" had a success rate of 30% and 70% when starting from step 1 and step 2, respectively. Most of the failures are caused by the robot being unable to open the bag and not being able to unfold it.

5.6 Discussion

Prior to the current work, the DQN and A2C algorithms were implemented, aiming to solve the problem of learning bagging in the real world. However, low performance was observed during the bagging task with DQN and A2C (refer to Table 5.4), which can be attributed to the limited number of training steps (400 training steps in total). Achieving better results with these algorithms would likely require implementing them in a simulated environment. However, this approach presents challenges, such as the reality-to-simulation gap discussed in Chapter 2 and the need to simulate the physical properties of the bag accurately. Furthermore, DQN and A2C algorithms typically require thousands to millions of steps to achieve stable learning. For instance, Hester et al. (Hester et al. 2018) demonstrated that Deep Q-network from Demonstrations (DQNfD) required 1 million steps to achieve satisfactory scores in their experiments. While using demonstrations accelerated the learning process in their experiments, DQN took 84 to 85 million steps for similar performance in the same application. Using Deep Reinforcement Learning (DRL) algorithms in the real world is challenging due to the necessary number of iteration steps to train an agent. This problem led to the design of the Π-learning algorithm.

Additionally, after completing the learning phase with the proposed framework, the robot performed 100 attempts using each approach for the bagging task with "Bag 1" (refer to Fig. 5.8a). The success rates are summarised in Table 5.4, in which it can be appreciated that the main reason for the failures was the robot's inability to complete the unfolding (step 1) or opening of the bag (step 2). This highlights the need for further improvements, such as a more robust unfolding strategy and enhanced camera measurement accuracy. Improved accuracy of the camera's measurements would allow the robot's gripper to reach the surface of the bag's layer more precisely, as even a difference of 1 mm caused the robot to grasp both layers or fail to grasp any layer at all (The RealSense™ camera provokes these fluctuations).

The decision to not use continuous RL algorithms, such as Deep Deterministic

Policy Gradient (DDPG) (Lillicrap et al. 2015), or Soft Actor-Critic (SAC) (Haarnoja et al. 2018), is motivated by the aim of preventing dangerous behaviours of the robot in the experimental setup, particularly during the learning stage. Unlike continuous RL algorithms, which may lead to collisions and undesired behaviours due to their inherent exploration process, defining primitive actions and employing a discrete action-selection approach helped to prevent such incidents specifically for the task proposed in this chapter. To summarise, the following list encompasses the main limitations of the proposed framework and, therefore, potential challenges that require further research:

- **Unfolding difficulty:** The current approach faces challenges when it comes to unfolding the bag. Problems such as the bag being too large or the opening not being visible after following our approach prevent the framework from generalising to a wider set of bags.

- **Opening difficulty:** The robot encounters issues in opening the bag after the unfolding step. This is due to the resistance of the same bag's material to change its initial configuration because the robot grasps two layers instead of one.

- **Accuracy of the camera:** The camera's accuracy is not sufficient for the robot to distinguish between layers. Even a difference of 1 mm causes the robot to grasp both layers or fail to grasp any layer.

- **Automatically recognise the bag's opening with no markers:** For the cases presented in this paper, the bags had no handles, making it ambiguous and challenging to find the opening using only a camera. This process is difficult even for humans, who often need to rotate the bag several times before finding the opening.

Some potential solutions to the challenges listed above are proposed as follows:

- **Improved unfolding and opening strategy:** Adding a second arm or using a human-like gripper would allow the implementation of more robust strategies involving dexterity or simply adding more resources to the robot to hold the bag while attempting more actions.

- **Incorporating tactile sensing:** The camera's low accuracy limitations can be overcome by adding fingers with tactile feedback. This would allow the robot to localise the real position of the layers and, in this manner, know when the gripper has held one or two layers of the bag.

- **Enhanced exploration strategy:** To overcome the ambiguity regarding how to find the opening of the bag without markers, it would be necessary to implement a process that involves robustly unfolding the bag. Then, explore one side of the bag. If that side does not open, continue with the next side if the bag is still unfolded. Repeat these actions until the robot finds the opening of the bag.

5.7 Summary

This chapter presented an efficient learning framework for a robot manipulator to acquire the bagging task. The real-world learning robot-bagging framework has been empirically validated. Leveraging the Π-learning algorithm, this framework enables efficient learning of the bagging task in real-world scenarios. After training for a total of 400 steps, which took approximately three hours, the framework achieved a success rate of 60% and 80% when starting from the unfolding or opening step, respectively. Additionally, the framework demonstrated generalisation capabilities across different bags.

However, there are certain limitations to the proposed framework. For instance, the framework's applicability is restricted to bags made from specific materials, such as cotton bags, and may not be suitable for handling plastic bags. Furthermore, the

framework is designed to handle only one object that is smaller than the bag's opening. Additionally, there is room for improvement in the primitive actions of unfolding the bag (τ_{grasp}) and grasping only one layer of the bag ($\tau_{scratch}$), as these actions were the primary causes of the robot's failure in the bagging tasks. Despite these limitations, the framework exhibits the potential to tackle challenging problems such as deformable object manipulation to a wider degree.

Future work could focus on enhancing the unfolding and opening routines by incorporating bi-manual robotic manipulation techniques. This advancement would enable the bagging of multiple objects, expanding the framework's capabilities beyond a single object. This would help to overcome the limitation of using only red cubes. Adding more advanced vision techniques such as You Only Look Once (YOLO) (Redmon et al. 2016) would allow the robot to recognise a wider set of objects, and combined with the depth cloud produced by the RealSense™ camera, it is possible to find grasping points. Subsequently, finding the right orientation of the object would also influence the bagging task, and consequently, this information should also be part of the state space. Additionally, the integration of supervised learning methods could be explored since it has the potential to facilitate the generalisation of the perception module to a wider range of bag types. This extension would enhance the framework's versatility and applicability.

Chapter 6

Contributions, Conclusions and Future Work

This chapter concludes the study undertaken in this work. Section 6.1 focuses on highlighting the main contributions of this thesis. The findings and key takeaways are presented in Section 6.2, while recommendations for future research directions are discussed in Section 6.3. A roadmap of the frameworks and algorithms developed in this thesis is shown in Fig. 6.1.

6.1 Contributions

The contributions of this thesis are summarised as follows:

- Iota Explicit Context Representation (IECR), a framework that uses Contextual Key Frames (CKFs) as state representation to improve the learning and exploration processes of Reinforcement Learning (RL) agents.

- Four new algorithms based on IECR: Iota deep Q-network (IDQN), Iota double deep Q-network (IDDQN), Iota dueling deep Q-network (IDuDQN), and Iota dueling double deep Q-network (IDDDQN). These algorithms were

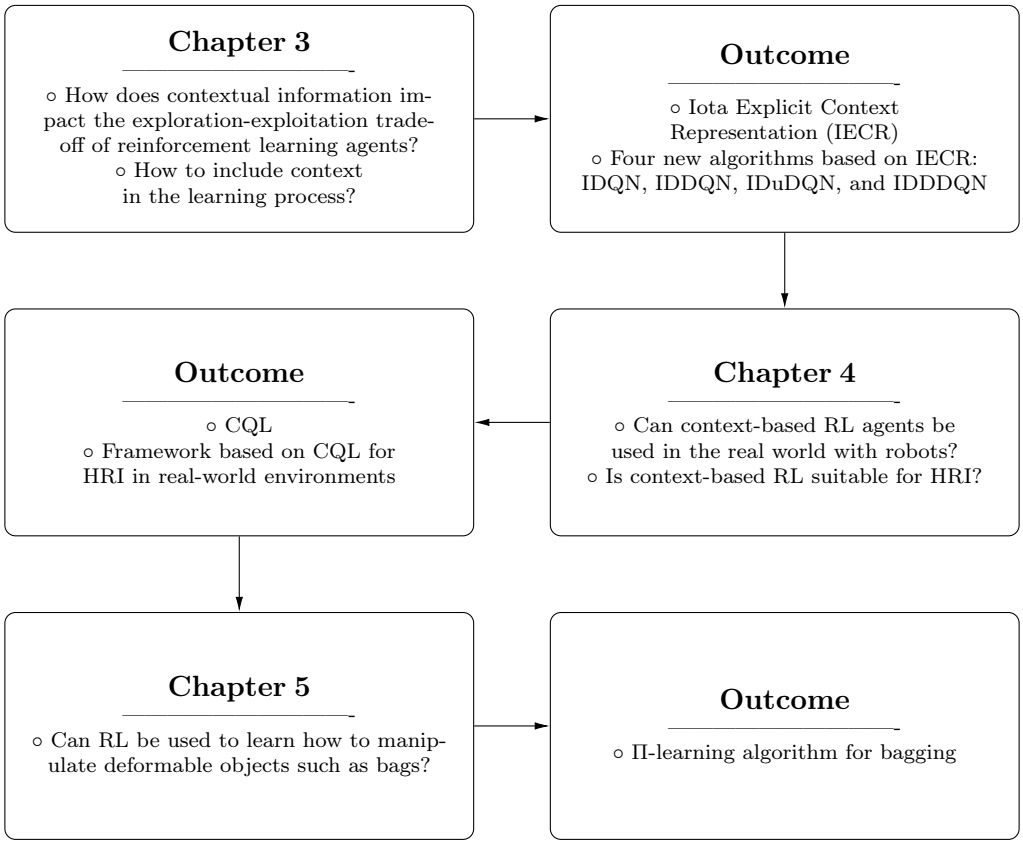


Figure 6.1: Roadmap of the techniques developed in this thesis.

demonstrated to be superior in terms of converging faster than their baseline equivalents in the environments used in this work.

- Contextual Q-learning (CQL) is introduced, an algorithm that allows efficient learning in the context of active Human-Robot Interaction (HRI).

- A framework based on CQL that allows robots to perform HRI in the real world while actively reacting to changes in the environment provoked by a human user.

- The Π-learning algorithm is introduced, allowing robots to learn how to bag in the real world.

- A framework for learning bagging that provides a reliable perception of the bag state, learning, and generalisation of the learned task.

6.2 Conclusions

In conclusion, this thesis has explored the impact of using contextual information, such as semantics and affordances, aiming to address the following research questions: How does contextual information impact the exploration-exploitation trade-off of reinforcement learning agents? How to include context in the learning process? Can context-based RL agents be used in the real world with robots? Is context-based RL suitable for HRI? And, can RL be used to learn how to manipulate deformable objects such as bags?

In order to answer the research questions mentioned above, the following objectives were pursued: (1) to investigate the effectiveness of using contextual information for training reinforcement learning agents in discrete environments, (2) to develop and test a robotic RL context-based system to perform HRI for rigid objects in the real world, (3) to develop and test a robotic RL context-based system for learning to manipulate deformable objects in the real world. By pursuing these objectives, this thesis' aim was to contribute to developing robust RL algorithms that can learn in simulation and the real world.

For the first objective, the findings of Chapter 3 shed light on the benefits of utilising CKFs representations and affordances to train neural networks to solve discrete environments. The results highlight the effectiveness of incorporating contextual information in training RL agents, leading to improved learning performance that surpasses current state-of-the-art baselines.

Moreover, by pursuing the second objective, this thesis has made significant contributions to the domain of rigid object manipulation during HRI based on an RL approach. The framework introduced in Chapter 4 shows the potential to solve real-world HRI tasks. It demonstrates how using contextual information enhances the learning capabilities of RL agents. Contextual information can be used to deal with problems not only for discrete environments such as games but also for robots running in the real world.

Additionally, Chapter 5 has explored the challenging domain of deformable object manipulation, aiming to address the third objective. Through the development and testing of an RL framework, a robot showed capable of learning to manipulate bags. All the learning process was carried out in the real world, which also shows the potential of using affordances to reduce the exploration space of the agent to the point that the robot can use an RL-based approach to learn the task in around 3 hours in the real world.

Overall, this thesis makes a valuable contribution to the RL field by proposing algorithms and frameworks that use contextual information to learn in a more human-like manner. This opens the door for further research and development in the domain of RL and robotics, aiming to create autonomous agents that can learn and operate in real-world scenarios as humans do.

6.3 Future work

The research conducted in this thesis has provided valuable insight into the fields of RL and robotics. Despite that, several challenges deserve further exploration, such as:

- Investigation of RL in the real world: While this study focused more on using simulations and less on learning in the real world, future work can explore methods that allow learning from data obtained in the real world as humans do. Despite the challenges, it is still necessary to gain a more comprehensive understanding of the way humans learn and how to transfer these skills to artificial agents. One of the reasons behind this problem is that Neural Networks cannot "remember" in the same manner a Q-table can just store the corresponding Q-value of a certain transition. This provokes the Neural Network to be fed with the same data several times, dramatically reducing its learning speed. An alternative to avoid such a problem would be to design method-

ologies to reduce the state space. In robotics, most of the tasks are related to picking and placing, so focusing on supervised learning to identify the grasping points of the most common objects humans have to deal with (e.g., door handles, drawers, or tools) combine it with classical planning methods and the powerful contextual capabilities of Large Language Models (LLMs) could simplify the state space. This approach can be used to generate datasets of robots executing diverse tasks. Then, the data set can be used to train a Neural Network with these demonstrations.

- Deformable objects manipulation: This thesis primarily examined the case of bagging, but there is a need to expand the research to a more diverse set of deformable objects. The investigation of methods that deal with deformable objects is still underdeveloped. Hence, conducting research in this domain is of great potential not only from the research point of view but also in terms of the wide range of applications that can benefit everyone's daily activities. One of the challenges robots encounter when interacting with deformable objects is the absence of feedback, making it challenging for them to determine the optimal sequence of actions to manipulate the object effectively. Tactile sensing has the potential to address this issue. In this context, developing tactile sensors that equate to the human sense of touch is paramount. Integrating tactile sensing data into RL algorithms would enhance the agent's understanding of the current state of the deformable object, potentially resulting in more robust learning and manipulation.

- Continuous RL algorithms with embedded contextual information: This thesis focused on discrete action spaces and then used several approaches, such as spline interpolation and primitive action definitions, to transform discrete policies into continuous ones. Future research could involve embedding contextual information into continuous action space RL algorithms. Algorithms such as

Proximal Policy Optimisation (PPO) and Deep Deterministic Policy Gradient (DDPG) can also benefit from affordances. An approach to enable this would involve utilising classical methods such as Probabilistic Roadmaps (PRM) to discard robot configurations that lead to collisions. With such information, it is possible to generate a probabilistic function that first reduces the likelihood of the ϵ-greedy policy selecting actions that lead to collisions. Second, if the neural network assigns a high probability to a certain range of actions that lead to collisions, the loss can be increased based on this information. This approach has the potential to accelerate the learning process, similar to the algorithms developed in Chapter 3.

Appendix A

External Resources

To access the code and packages necessary to reproduce this work, please visit the GitHub repositories:

- **Chapter 3:**

  `https://github.com/FranciscoMunguiaGaleano/Iota-Deep-Q-Network`

- **Chapter 4:**

  `https://github.com/FranciscoMunguiaGaleano/RLforHRI`

- **Chapter 5:**

  `https://github.com/FranciscoMunguiaGaleano/LearningToBag`

Bibliography

Adams, S., T. Cody, and P. A. Beling (2022). "A survey of inverse reinforcement learning". In: *Artificial Intelligence Review* 55.6, pp. 4307–4346.

Andriella, A., C. Torras, and G. Alenya (2020). "Short-term human–robot interaction adaptability in real-world environments". In: *International Journal of Social Robotics* 12, pp. 639–657.

Andrychowicz, M. et al. (2017). "Hindsight experience replay". In: *Advances in neural information processing systems* 30.

Andrychowicz, O. M. et al. (2020). "Learning dexterous in-hand manipulation". In: *The International Journal of Robotics Research* 39.1, pp. 3–20.

Ardón, P., È. Pairet, R. P. Petrick, S. Ramamoorthy, and K. S. Lohan (2019). "Learning grasp affordance reasoning through semantic relations". In: *IEEE Robotics and Automation Letters* 4.4, pp. 4571–4578.

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

Arulkumaran, K., M. P. Deisenroth, M. Brundage, and A. A. Bharath (2017). "Deep reinforcement learning: A brief survey". In: *IEEE Signal Processing Magazine* 34.6, pp. 26–38.

Bahety, A. et al. (2022). "Bag All You Need: Learning a Generalizable Bagging Strategy for Heterogeneous Objects". In: *arXiv preprint arXiv:2210.09997*.

Bai, Y., C. Jin, and T. Yu (2020). "Near-optimal reinforcement learning with self-play". In: *Advances in neural information processing systems* 33, pp. 2159–2170.

Balkcom, D. J. and M. T. Mason (2004). "Introducing robotic origami folding". In: *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004.* Vol. 4. IEEE, pp. 3245–3250.

— (2008). "Robotic origami folding". In: *The International Journal of Robotics Research* 27.5, pp. 613–627.

Bellemare, M., S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos (2016). "Unifying count-based exploration and intrinsic motivation". In: *Advances in neural information processing systems* 29.

Bellman, R. (1966). "Dynamic programming". In: *Science* 153.3731, pp. 34–37.

Benjamins, C., T. Eimer, F. Schubert, A. Biedenkapp, B. Rosenhahn, F. Hutter, and M. Lindauer (2021). "Carl: A benchmark for contextual and adaptive reinforcement learning". In: *arXiv preprint arXiv:2110.02102*.

Benjamins, C. et al. (2022). "Contextualize Me–The Case for Context in Reinforcement Learning". In: *arXiv preprint arXiv:2202.04500*.

Blondé, L. and A. Kalousis (2019). "Sample-efficient imitation learning via generative adversarial nets". In: *The 22nd International Conference on Artificial Intelligence and Statistics*. PMLR, pp. 3138–3148.

Borràs, J., G. Alenyà, and C. Torras (2020). "A grasping-centered analysis for cloth manipulation". In: *IEEE Transactions on Robotics* 36.3, pp. 924–936.

Brown, T. et al. (2020). "Language models are few-shot learners". In: *Advances in neural information processing systems* 33, pp. 1877–1901.

Browne, C. B. et al. (2012). "A survey of monte carlo tree search methods". In: *IEEE Transactions on Computational Intelligence and AI in games* 4.1, pp. 1–43.

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

Casgrain, P., B. Ning, and S. Jaimungal (2022). "Deep Q-learning for Nash equilibria: Nash-DQN". In: *Applied Mathematical Finance* 29.1, pp. 62–78.

Chai, R., H. Niu, J. Carrasco, F. Arvin, H. Yin, and B. Lennox (2022). "Design and experimental validation of deep reinforcement learning-based fast trajectory planning and control for mobile robot in unknown environment". In: *IEEE Transactions on Neural Networks and Learning Systems*.

Chalmers, E., E. B. Contreras, B. Robertson, A. Luczak, and A. Gruber (2017). "Learning to predict consequences as a method of knowledge transfer in reinforcement learning". In: *IEEE transactions on neural networks and learning systems* 29.6, pp. 2259–2270.

Chen, C., Y. Liu, S. Kreiss, and A. Alahi (2019). "Crowd-robot interaction: Crowd-aware robot navigation with attention-based deep reinforcement learning". In: *2019 international conference on robotics and automation (ICRA)*. IEEE, pp. 6015–6022.

Chen, C., A. Seff, A. Kornhauser, and J. Xiao (2015). "Deepdriving: Learning affordance for direct perception in autonomous driving". In: *Proceedings of the IEEE international conference on computer vision*, pp. 2722–2730.

Chen, L. Y., B. Shi, D. Seita, R. Cheng, T. Kollar, D. Held, and K. Goldberg (2022). "AutoBag: Learning to Open Plastic Bags and Insert Objects". In: *arXiv preprint arXiv:2210.17217*.

Chen, L. Y. et al. (2023). "Bagging by Learning to Singulate Layers Using Interactive Perception". In: *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 3176–3183.

Chen, X., M. W. Ulmer, and B. W. Thomas (2022). "Deep Q-learning for same-day delivery with vehicles and drones". In: *European Journal of Operational Research* 298.3, pp. 939–952.

Cheng, H. and M. Q.-H. Meng (2018). "A grasp pose detection scheme with an end-to-end CNN regression approach". In: *2018 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. IEEE, pp. 544–549.

Cheng, H., Y. Wang, and M. Q.-H. Meng (2022). "A Vision-Based Robot Grasping System". In: *IEEE Sensors Journal* 22.10, pp. 9610–9620.

Ciou, P.-H., Y.-T. Hsiao, Z.-Z. Wu, S.-H. Tseng, and L.-C. Fu (2018). "Composite reinforcement learning for social robot navigation". In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 2553–2558.

Cobbe, K., O. Klimov, C. Hesse, T. Kim, and J. Schulman (2019). "Quantifying generalization in reinforcement learning". In: *International Conference on Machine Learning*. PMLR, pp. 1282–1289.

Cruz, F., S. Magg, C. Weber, and S. Wermter (2016a). "Training agents with interactive reinforcement learning and contextual affordances". In: *IEEE Transactions on Cognitive and Developmental Systems* 8.4, pp. 271–284.

Cruz, F., G. I. Parisi, J. Twiefel, and S. Wermter (2016b). "Multi-modal integration of dynamic audiovisual patterns for an interactive reinforcement learning scenario". In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 759–766.

Cruz, F., G. I. Parisi, and S. Wermter (2018a). "Multi-modal feedback for affordance-driven interactive reinforcement learning". In: *2018 International Joint Conference on Neural Networks (IJCNN)*. IEEE, pp. 1–8.

Cruz, F., P. Wüppen, A. Fazrie, C. Weber, and S. Wermter (2018b). "Action selection methods in a robotic reinforcement learning scenario". In: *2018 IEEE Latin American Conference on Computational Intelligence (LA-CCI)*. IEEE, pp. 1–6.

CVZone (n.d.). *CVZone - OpenCV and MediaPipe Tools and Resources*. `https://github.com/cvzone/cvzone`. Accessed: May 2, 2023.

Deisenroth, M. and C. E. Rasmussen (2011). "PILCO: A model-based and data-efficient approach to policy search". In: *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pp. 465–472.

Do, T.-T., A. Nguyen, and I. Reid (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.

Dromnelle, R., B. Girard, E. Renaudo, R. Chatila, and M. Khamassi (2020). "Coping with the variability in humans reward during simulated human-robot interactions through the coordination of multiple learning strategies". In: *2020 29th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*. IEEE, pp. 612–617.

Dulac-Arnold, G., N. Levine, D. J. Mankowitz, J. Li, C. Paduraru, S. Gowal, and T. Hester (2021). "Challenges of real-world reinforcement learning: definitions, benchmarks and analysis". In: *Machine Learning* 110.9, pp. 2419–2468.

Elbrechter, C., R. Haschke, and H. Ritter (2011). "Bi-manual robotic paper manipulation based on real-time marker tracking and physical modelling". In: *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, pp. 1427–1432.

— (2012). "Folding paper with anthropomorphic robot hands using real-time physics-based modeling". In: *2012 12th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2012)*. IEEE, pp. 210–215.

Everett, M., Y. F. Chen, and J. P. How (2021). "Collision avoidance in pedestrian-rich environments with deep reinforcement learning". In: *IEEE Access* 9, pp. 10357–10377.

Fan, J., Z. Wang, Y. Xie, and Z. Yang (2020). "A theoretical analysis of deep Q-learning". In: *Learning for Dynamics and Control*. PMLR, pp. 486–489.

Finn, C., P. Abbeel, and S. Levine (2017). "Model-agnostic meta-learning for fast adaptation of deep networks". In: *International conference on machine learning*. PMLR, pp. 1126–1135.

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

Fujimoto, S., D. Meger, and D. Precup (2019). "Off-policy deep reinforcement learning without exploration". In: *International conference on machine learning*. PMLR, pp. 2052–2062.

Gao, C., Z. Li, H. Gao, and F. Chen (2023). "Iterative Interactive Modeling for Knotting Plastic Bags". In: *Conference on Robot Learning*. PMLR, pp. 571–582.

Gao, Y., E. Sibirtseva, G. Castellano, and D. Kragic (2019). "Fast adaptation with meta-reinforcement learning for trust modelling in human-robot interaction". In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 305–312.

Ghadirzadeh, A., X. Chen, W. Yin, Z. Yi, M. Björkman, and D. Kragic (2020). "Human-centered collaborative robots with deep reinforcement learning". In: *IEEE Robotics and Automation Letters* 6.2, pp. 566–571.

Gibson, J. J. (1977). "The theory of affordances". In: *Hilldale, USA* 1.2, pp. 67–82.

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

Google (n.d.). *Google Cloud Speech-to-Text*. `https://cloud.google.com/speech-to-text`. Accessed: May 2, 2023.

Gu, N., Z. Zhang, R. He, and L. Yu (2024). "ShakingBot: dynamic manipulation for bagging". In: *Robotica* 42.3, pp. 775–791.

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

Hallak, A., D. Di Castro, and S. Mannor (2015). "Contextual markov decision processes". In: *arXiv preprint arXiv:1502.02259*.

Hanna, J. P., S. Desai, H. Karnan, G. Warnell, and P. Stone (2021a). "Grounded action transformation for sim-to-real reinforcement learning". In: *Machine Learning* 110.9, pp. 2469–2499.

Hanna, J. P., S. Niekum, and P. Stone (2021b). "Importance sampling in reinforcement learning with an estimated behavior policy". In: *Machine Learning* 110.6, pp. 1267–1317.

Hansen, N. and X. Wang (2021). "Generalization in reinforcement learning by soft data augmentation". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 13611–13617.

Hasselt, H. (2010). "Double Q-learning". In: *Advances in neural information processing systems* 23.

Hester, T. et al. (2018). "Deep q-learning from demonstrations". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32.

Hill, A. et al. (2018). *Stable Baselines*. `https://github.com/hill-a/stable-baselines`.

Hochreiter, S. and J. Schmidhuber (1997). "Long short-term memory". In: *Neural computation* 9.8, pp. 1735–1780.

Hoque, R. et al. (2022). "Visuospatial foresight for physical sequential fabric manipulation". In: *Autonomous Robots*, pp. 1–25.

Hwang, P.-J., C.-C. Hsu, and W.-Y. Wang (2020). "Development of a mimic robot-Learning from demonstration incorporating object detection and multiaction recognition". In: *IEEE Consumer Electronics Magazine* 9.3, pp. 79–87.

Icarte, R. T., T. Q. Klassen, R. Valenzano, and S. A. McIlraith (2022). "Reward machines: Exploiting reward function structure in reinforcement learning". In: *Journal of Artificial Intelligence Research* 73, pp. 173–208.

Jain, S. et al. (2018). "Deep q-learning for navigation of robotic arm for tokamak inspection". In: *Algorithms and Architectures for Parallel Processing: 18th International Conference, ICA3PP 2018, Guangzhou, China, November 15-17, 2018, Proceedings, Part IV 18*. Springer, pp. 62–71.

Jangir, R., G. Alenya, and C. Torras (2020). "Dynamic cloth manipulation with deep reinforcement learning". In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 4630–4636.

Ji, M., L. Zhang, and S. Wang (2019). "A path planning approach based on Q-learning for robot arm". In: *2019 3rd International Conference on Robotics and Automation Sciences (ICRAS)*. IEEE, pp. 15–19.

Jørgensen, T. B., S. H. N. Jensen, H. Aanæs, N. W. Hansen, and N. Krüger (2019). "An adaptive robotic system for doing pick and place operations with deformable objects". In: *Journal of Intelligent & Robotic Systems* 94, pp. 81–100.

Kabra, A., A. Agarwal, and A. S. Parihar (2021a). "Cluster-based deep contextual reinforcement learning for top-k recommendations". In: *Proceedings of the International Conference on Computing and Communication Systems: I3CS 2020, NEHU, Shillong, India*. Springer, pp. 125–135.

— (2021b). "Potent Real-Time Recommendations Using Multimodel Contextual Reinforcement Learning". In: *IEEE Transactions on Computational Social Systems* 9.2, pp. 581–593.

Karaman, S. and E. Frazzoli (2010). "Incremental sampling-based algorithms for optimal motion planning". In: *Robotics Science and Systems VI* 104.2.

Khamassi, M., G. Velentzas, T. Tsitsimis, and C. Tzafestas (2018). "Robot fast adaptation to changes in human engagement during simulated dynamic social interaction with active exploration in parameterized reinforcement learning". In: *IEEE Transactions on Cognitive and Developmental Systems* 10.4, pp. 881–893.

Khan, M. A.-M., M. R. J. Khan, A. Tooshil, N. Sikder, M. P. Mahmud, A. Z. Kouzani, and A.-A. Nahid (2020). "A systematic review on reinforcement learning-based robotics within the last decade". In: *IEEE Access* 8, pp. 176598–176623.

Khetarpal, K., Z. Ahmed, G. Comanici, D. Abel, and D. Precup (2020). "What can I do here? A Theory of Affordances in Reinforcement Learning". In: *International Conference on Machine Learning*. PMLR, pp. 5243–5253.

Konar, A., I. G. Chakraborty, S. J. Singh, L. C. Jain, and A. K. Nagar (2013). "A deterministic improved Q-learning for path planning of a mobile robot". In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 43.5, pp. 1141–1153.

Kontoudis, G. P. and K. G. Vamvoudakis (2019). "Kinodynamic motion planning with continuous-time Q-learning: An online, model-free, and safe navigation framework". In: *IEEE transactions on neural networks and learning systems* 30.12, pp. 3803–3817.

Koppula, H. S. and A. Saxena (2015). "Anticipating human activities using object affordances for reactive robotic response". In: *IEEE transactions on pattern analysis and machine intelligence* 38.1, pp. 14–29.

Kormushev, P., S. Calinon, and D. G. Caldwell (2013). "Reinforcement learning in robotics: Applications and real-world challenges". In: *Robotics* 2.3, pp. 122–148.

Kostas, J., Y. Chandak, S. M. Jordan, G. Theocharous, and P. Thomas (2021). "High Confidence Generalization for Reinforcement Learning". In: *International Conference on Machine Learning*. PMLR, pp. 5764–5773.

Lathuilière, S., B. Massé, P. Mesejo, and R. Horaud (2018). "Deep reinforcement learning for audio-visual gaze control". In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 1555–1562.

LaValle, S. M. et al. (1998). "Rapidly-exploring random trees: A new tool for path planning. 1998". In: *URL http://citeseerx. ist. psu. edu/viewdoc/summary*.

Lele, A. S., Y. Fang, J. Ting, and A. Raychowdhury (2020). "Learning to walk: Spike based reinforcement learning for hexapod robot central pattern generation". In:

2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS). IEEE, pp. 208–212.

Levine, S., C. Finn, T. Darrell, and P. Abbeel (2016). "End-to-end training of deep visuomotor policies". In: *The Journal of Machine Learning Research* 17.1, pp. 1334–1373.

Levine, S., P. Pastor, A. Krizhevsky, J. Ibarz, and D. Quillen (2018). "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection". In: *The International journal of robotics research* 37.4-5, pp. 421–436.

Li, G., Y. Wei, Y. Chi, Y. Gu, and Y. Chen (2020). "Sample complexity of asynchronous Q-learning: Sharper analysis and variance reduction". In: *Advances in neural information processing systems* 33, pp. 7031–7043.

— (2021). "Sample complexity of asynchronous Q-learning: Sharper analysis and variance reduction". In: *IEEE Transactions on Information Theory* 68.1, pp. 448–473.

Li, H., Q. Zhang, and D. Zhao (2019). "Deep reinforcement learning-based automatic exploration for navigation in unknown environment". In: *IEEE transactions on neural networks and learning systems* 31.6, pp. 2064–2076.

Li, S., X. Xu, and L. Zuo (2015). "Dynamic path planning of a mobile robot with improved Q-learning algorithm". In: *2015 IEEE international conference on information and automation*. IEEE, pp. 409–414.

Li, X., H. Liu, and M. Dong (2021). "A general framework of motion planning for redundant robot manipulator based on deep reinforcement learning". In: *IEEE Transactions on Industrial Informatics* 18.8, pp. 5253–5263.

Li, X., L. Li, J. Gao, X. He, J. Chen, L. Deng, and J. He (2015). "Recurrent reinforcement learning: a hybrid approach". In: *arXiv preprint arXiv:1509.03044*.

Lillicrap, T. P. et al. (2015). "Continuous control with deep reinforcement learning". In: *arXiv preprint arXiv:1509.02971*.

Lin, H.-Y., S.-C. Liang, and Y.-K. Chen (2020). "Robotic grasping with multi-view image acquisition and model-based pose estimation". In: *IEEE Sensors Journal* 21.10, pp. 11870–11878.

Lin, J.-L., K.-S. Hwang, W.-C. Jiang, and Y.-J. Chen (2016). "Gait balance and acceleration of a biped robot based on Q-learning". In: *IEEE access* 4, pp. 2439–2449.

Littman, M. L. (1994). "Markov games as a framework for multi-agent reinforcement learning". In: *Machine learning proceedings 1994*. Elsevier, pp. 157–163.

Liu, D., Z. Wang, B. Lu, M. Cong, H. Yu, and Q. Zou (2020). "A reinforcement learning-based framework for robot manipulation skill acquisition". In: *IEEE Access* 8, pp. 108429–108437.

Liu, S., Y. Li, and W. Fu (2022). "Human-centered attention-aware networks for action recognition". In: *International Journal of Intelligent Systems*.

Liu, S., S. Wang, X. Liu, C.-T. Lin, and Z. Lv (2020). "Fuzzy detection aided real-time and robust visual tracking under complex environments". In: *IEEE Transactions on Fuzzy Systems* 29.1, pp. 90–102.

Liu, Z., Q. Liu, L. Wang, W. Xu, and Z. Zhou (2021). "Task-level decision-making for dynamic and stochastic human-robot collaboration based on dual agents deep reinforcement learning". In: *The International Journal of Advanced Manufacturing Technology* 115.11-12, pp. 3533–3552.

Low, E. S., P. Ong, and K. C. Cheah (2019). "Solving the optimal path planning of a mobile robot using improved Q-learning". In: *Robotics and Autonomous Systems* 115, pp. 143–161.

Lowe, R., A. Almér, P. Gander, and C. Balkenius (2019). "Vicarious value learning and inference in human-human and human-robot interaction". In: *2019 8th International Conference on Affective Computing and Intelligent Interaction Workshops and Demos (ACIIW)*. IEEE, pp. 395–400.

Lowe, R., Y. I. Wu, A. Tamar, J. Harb, O. Pieter Abbeel, and I. Mordatch (2017). "Multi-agent actor-critic for mixed cooperative-competitive environments". In: *Advances in neural information processing systems* 30.

Luebbers, M. B., A. Tabrez, and B. Hayes (2022). "Augmented Reality-Based Explainable AI Strategies for Establishing Appropriate Reliance and Trust in Human-Robot Teaming". In: *5th International Workshop on Virtual, Augmented, and Mixed Reality for HRI*.

Luis, S. Y., D. G. Reina, and S. L. T. Marín (2021). "A multiagent deep reinforcement learning approach for path planning in autonomous surface vehicles: the Ypacaraí lake patrolling case". In: *IEEE Access* 9, pp. 17084–17099.

Ma, X., D. Hsu, and W. S. Lee (2022). "Learning latent graph dynamics for visual manipulation of deformable objects". In: *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 8266–8273.

Mahler, J. et al. (2017). "Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics". In: *arXiv preprint arXiv:1703.09312*.

Maoudj, A. and A. Hentout (2020). "Optimal path planning approach based on Q-learning algorithm for mobile robots". In: *Applied Soft Computing* 97, p. 106796.

Matas, J., S. James, and A. J. Davison (2018). "Sim-to-real reinforcement learning for deformable object manipulation". In: *Conference on Robot Learning*. PMLR, pp. 734–743.

Millan-Arias, C. C., B. J. Fernandes, F. Cruz, R. Dazeley, and S. Fernandes (2021). "A robust approach for continuous interactive actor-critic algorithms". In: *IEEE Access* 9, pp. 104242–104260.

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

Mnih, V. et al. (2015). "Human-level control through deep reinforcement learning". In: *nature* 518.7540, pp. 529–533.

Mnih, V. et al. (2016). "Asynchronous methods for deep reinforcement learning". In: *International conference on machine learning*. PMLR, pp. 1928–1937.

Modares, H., I. Ranatunga, F. L. Lewis, and D. O. Popa (2015). "Optimized assistive human–robot interaction using reinforcement learning". In: *IEEE transactions on cybernetics* 46.3, pp. 655–667.

Munguia-Galeano, F., A.-H. Tan, and Z. Ji (2023a). "Deep reinforcement learning with explicit context representation". In: *IEEE Transactions on Neural Networks and Learning Systems*.

Munguia-Galeano, F., S. Veeramani, J. D. Hernández, Q. Wen, and Z. Ji (2023b). "Affordance-based human-robot interaction with reinforcement learning". In: *IEEE Access*.

Munguia-Galeano, F., J. Zhu, J. D. Hernández, and Z. Ji (2024). "Learning to bag with a simulation-free reinforcement learning framework for robots". In: *IET Cyber-Systems and Robotics* 6.2, e12113.

Naeem, M., S. T. H. Rizvi, and A. Coronato (2020). "A gentle introduction to reinforcement learning and its application in different fields". In: *IEEE access* 8, pp. 209320–209344.

Nair, A., D. Chen, P. Agrawal, P. Isola, P. Abbeel, J. Malik, and S. Levine (2017). "Combining self-supervised learning and imitation for vision-based rope manipulation". In: *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE, pp. 2146–2153.

Nair, A., B. McGrew, M. Andrychowicz, W. Zaremba, and P. Abbeel (2018). "Overcoming exploration in reinforcement learning with demonstrations". In: *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, pp. 6292–6299.

Ng, A. Y., S. Russell, et al. (2000). "Algorithms for inverse reinforcement learning." In: *Icml*. Vol. 1, p. 2.

Nguyen, H. and H. La (2019). "Review of deep reinforcement learning for robot manipulation". In: *2019 Third IEEE International Conference on Robotic Computing (IRC)*. IEEE, pp. 590–595.

Nikolaidis, S., R. Ramakrishnan, K. Gu, and J. Shah (2015). "Efficient model learning from joint-action demonstrations for human-robot collaborative tasks". In: *Proceedings of the tenth annual ACM/IEEE international conference on human-robot interaction*, pp. 189–196.

Osband, I., B. Van Roy, D. J. Russo, Z. Wen, et al. (2019). "Deep Exploration via Randomized Value Functions." In: *J. Mach. Learn. Res.* 20.124, pp. 1–62.

Pan, J., X. Wang, Y. Cheng, and Q. Yu (2018). "Multisource transfer double DQN based on actor learning". In: *IEEE transactions on neural networks and learning systems* 29.6, pp. 2227–2238.

Pareek, S. and T. Kesavadas (2019). "iART: Learning from demonstration for assisted robotic therapy using lstm". In: *IEEE Robotics and Automation Letters* 5.2, pp. 477–484.

Pinto, L. and A. Gupta (2016). "Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours". In: *2016 IEEE international conference on robotics and automation (ICRA)*. IEEE, pp. 3406–3413.

— (2017). "Learning to push by grasping: Using multiple tasks for effective learning". In: *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE, pp. 2161–2168.

Polydoros, A. S. and L. Nalpantidis (2017). "Survey of model-based reinforcement learning: Applications on robotics". In: *Journal of Intelligent & Robotic Systems* 86.2, pp. 153–173.

Quillen, D., E. Jang, O. Nachum, C. Finn, J. Ibarz, and S. Levine (2018). "Deep reinforcement learning for vision-based robotic grasping: A simulated comparative evaluation of off-policy methods". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 6284–6291.

Raffin, A., J. Kober, and F. Stulp (2022). "Smooth exploration for robotic reinforcement learning". In: *Conference on Robot Learning*. PMLR, pp. 1634–1644.

Raileanu, R. and R. Fergus (2021). "Decoupling value and policy for generalization in reinforcement learning". In: *International Conference on Machine Learning*. PMLR, pp. 8787–8798.

Ravichandar, H., A. S. Polydoros, S. Chernova, and A. Billard (2020). "Recent advances in robot learning from demonstration". In: *Annual review of control, robotics, and autonomous systems* 3, pp. 297–330.

Redmon, J., S. Divvala, R. Girshick, and A. Farhadi (2016). "You only look once: Unified, real-time object detection". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788.

Ren, Z., D. Dong, H. Li, and C. Chen (2018). "Self-paced prioritized curriculum learning with coverage penalty in deep reinforcement learning". In: *IEEE transactions on neural networks and learning systems* 29.6, pp. 2216–2226.

Ribeiro, C. (2002). "Reinforcement learning agents". In: *Artificial intelligence review* 17, pp. 223–250.

Rountree, B. and D. Tsafrir (n.d.). *pyRAPL: A Python interface for RAPL*. `https://github.com/berkeley-abc/pyRAPL`. Accessed: May 2, 2023.

Roy, S., E. Kieson, C. Abramson, and C. Crick (2019). "Mutual reinforcement learning with robot trainers". In: *2019 14th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*. IEEE, pp. 572–573.

Rummery, G. A. and M. Niranjan (1994). *On-line Q-learning using connectionist systems*. Vol. 37. University of Cambridge, Department of Engineering Cambridge, UK.

Russell, S. (1998). "Learning agents for uncertain environments". In: *Proceedings of the eleventh annual conference on Computational learning theory*, pp. 101–103.

Şahin, E., M. Cakmak, M. R. Doğar, E. Uğur, and G. Üçoluk (2007). "To afford or not to afford: A new formalization of affordances toward affordance-based robot control". In: *Adaptive Behavior* 15.4, pp. 447–472.

Salvato, E., G. Fenu, E. Medvet, and F. A. Pellegrino (2021). "Crossing the reality gap: A survey on sim-to-real transferability of robot controllers in reinforcement learning". In: *IEEE Access* 9, pp. 153171–153187.

Schaul, T., J. Quan, I. Antonoglou, and D. Silver (2015). "Prioritized experience replay". In: *arXiv preprint arXiv:1511.05952*.

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

Seita, D., P. Florence, J. Tompson, E. Coumans, V. Sindhwani, K. Goldberg, and A. Zeng (2021). "Learning to rearrange deformable cables, fabrics, and bags with goal-conditioned transporter networks". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 4568–4575.

Sengadu Suresh, P., Y. Gui, and P. Doshi (2023). "Dec-AIRL: Decentralized Adversarial IRL for Human-Robot Teaming". In: *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems*, pp. 1116–1124.

Shafti, A., J. Tjomsland, W. Dudley, and A. A. Faisal (2020). "Real-world human-robot collaborative reinforcement learning". In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 11161–11166.

El-Shamouty, M., X. Wu, S. Yang, M. Albus, and M. F. Huber (2020). "Towards safe human-robot collaboration using deep reinforcement learning". In: *2020 IEEE international conference on robotics and automation (ICRA)*. IEEE, pp. 4899–4905.

Shao, K., Z. Tang, Y. Zhu, N. Li, and D. Zhao (2019). "A survey of deep reinforcement learning in video games". In: *arXiv preprint arXiv:1912.10944*.

Sharma, S. et al. (2022). "Learning Switching Criteria for Sim2Real Transfer of Robotic Fabric Manipulation Policies". In: *2022 IEEE 18th International Conference on Automation Science and Engineering (CASE)*. IEEE, pp. 1116–1123.

Sheridan, T. B. (2016). "Human–robot interaction: status and challenges". In: *Human factors* 58.4, pp. 525–532.

Shi, L., M. Pantic, O. Andersson, M. Tognon, R. Siegwart, and R. H. Jacobsen (2022). "Reactive Motion Planning for Rope Manipulation and Collision Avoidance using Aerial Robots". In: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 3384–3391.

Silver, D. et al. (2016). "Mastering the game of Go with deep neural networks and tree search". In: *nature* 529.7587, pp. 484–489.

Sodhani, S., A. Zhang, and J. Pineau (2021). "Multi-task reinforcement learning with context-based representations". In: *International Conference on Machine Learning*. PMLR, pp. 9767–9779.

Sonar, A., V. Pacelli, and A. Majumdar (2021). "Invariant policy optimization: Towards stronger generalization in reinforcement learning". In: *Learning for Dynamics and Control*. PMLR, pp. 21–33.

Stavridis, S., D. Papageorgiou, and Z. Doulgeri (2022). "Kinesthetic teaching of bimanual tasks with known relative constraints". In: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 11796–11801.

Sundaresan, P. et al. (2020). "Learning rope manipulation policies using dense object descriptors trained on synthetic depth data". In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 9411–9418.

Sutton, R. S. and A. G. Barto (1998). *Introduction to reinforcement learning*. Vol. 135. MIT press Cambridge.

Tabrez, A. and B. Hayes (2019). "Improving human-robot interaction through explainable reinforcement learning". In: *2019 14th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*. IEEE, pp. 751–753.

Tabrez, A., M. B. Luebbers, and B. Hayes (2020). "A survey of mental modeling techniques in human–robot teaming". In: *Current Robotics Reports* 1, pp. 259–267.

Tai, C.-S., J.-H. Hong, D.-Y. Hong, and L.-C. Fu (2022). "A real-time demand-side management system considering user preference with adaptive deep Q learning in home area network". In: *Sustainable Energy, Grids and Networks* 29, p. 100572.

Taylor, M. E. and P. Stone (2009). "Transfer learning for reinforcement learning domains: A survey." In: *Journal of Machine Learning Research* 10.7.

Teng, T.-H., A.-H. Tan, and J. M. Zurada (2014). "Self-organizing neural networks integrating domain knowledge and reinforcement learning". In: *IEEE transactions on neural networks and learning systems* 26.5, pp. 889–902.

Thomaz, A. L. and C. Breazeal (2006). *Reinforcement Learning with Human Teachers: Evidence of Feedback and Guidance with Implications for Learning Performance*. AAAI.

Urtans, E. and A. Nikitenko (2018). "Survey of deep Q-network variants in PyGame learning environment". In: *Proceedings of the 2018 2nd International Conference on Deep Learning Technologies*, pp. 27–36.

Veeramani, S. and S. Muthuswamy (2022). "Hybrid type multi-robot path planning of a serial manipulator and SwarmItFIX robots in sheet metal milling process". In: *Complex & Intelligent Systems* 8.4, pp. 2937–2954.

Voss, V., L. Nechepurenko, R. Schaefer, and S. Bauer (2020). "Playing a strategy game with knowledge-based reinforcement learning". In: *SN Computer Science* 1.2, p. 78.

Wang, C., K. V. Hindriks, and R. Babuska (2013). "Robot learning and use of affordances in goal-directed tasks". In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, pp. 2288–2294.

Wang, K., B. Kang, J. Shao, and J. Feng (2020). "Improving generalization in reinforcement learning with mixture regularization". In: *Advances in Neural Information Processing Systems* 33, pp. 7968–7978.

Wang, W., C. Du, W. Wang, and Z. Du (2019). "A PSO-optimized fuzzy reinforcement learning method for making the minimally invasive surgical arm cleverer". In: *IEEE Access* 7, pp. 48655–48670.

Wang, X., J. Zhao, X. Jiang, and Y.-H. Liu (2022a). "Learning-based fabric folding and box wrapping". In: *IEEE Robotics and Automation Letters* 7.2, pp. 5703–5710.

Wang, X. et al. (2022b). "Deep reinforcement learning: a survey". In: *IEEE Transactions on Neural Networks and Learning Systems*.

Wang, Z., T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas (2016). "Dueling network architectures for deep reinforcement learning". In: *International conference on machine learning*. PMLR, pp. 1995–2003.

Watkins, C. J. and P. Dayan (1992a). "Q-learning". In: *Machine learning* 8, pp. 279–292.

— (1992b). "Q-learning". In: *Machine learning* 8, pp. 279–292.

Weiss, K., T. M. Khoshgoftaar, and D. Wang (2016). "A survey of transfer learning". In: *Journal of Big data* 3.1, pp. 1–40.

Wen, S., J. Chen, S. Wang, H. Zhang, and X. Hu (2018). "Path planning of humanoid arm based on deep deterministic policy gradient". In: *2018 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. IEEE, pp. 1755–1760.

Wong, C.-C., S.-Y. Chien, H.-M. Feng, and H. Aoyama (2021). "Motion planning for dual-arm robot based on soft actor-critic". In: *IEEE Access* 9, pp. 26871–26885.

Wu, H., Z. Zhang, H. Cheng, K. Yang, J. Liu, and Z. Guo (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., F. Zhu, Q. Liu, and P. Zhao (2021). "Improving exploration efficiency of deep reinforcement learning through samples produced by generative model". In: *Expert Systems with Applications* 185, p. 115680.

Yamanobe, N. et al. (2017). "A brief review of affordance in robotic manipulation research". In: *Advanced Robotics* 31.19-20, pp. 1086–1101.

Yan, C. and X. Xiang (2018). "A path planning algorithm for uav based on improved q-learning". In: *2018 2nd international conference on robotics and automation sciences (ICRAS)*. IEEE, pp. 1–5.

Yan, Y., G. Li, Y. Chen, and J. Fan (2022). "The efficacy of pessimism in asynchronous Q-learning". In: *arXiv preprint arXiv:2203.07368*.

Yang, Z., K. Merrick, L. Jin, and H. A. Abbass (2018). "Hierarchical deep reinforcement learning for continuous action control". In: *IEEE transactions on neural networks and learning systems* 29.11, pp. 5174–5184.

Yarats, D., D. Brandfonbrener, H. Liu, M. Laskin, P. Abbeel, A. Lazaric, and L. Pinto (2022). "Don't Change the Algorithm, Change the Data: Exploratory Data for Offline Reinforcement Learning". In: *ICLR 2022 Workshop on Generalizable Policy Learning in Physical World*.

Yin, H., A. Varava, and D. Kragic (2021). "Modeling, learning, perception, and control methods for deformable object manipulation". In: *Science Robotics* 6.54, eabd8803.

Yu, T., A. Kumar, Y. Chebotar, K. Hausman, S. Levine, and C. Finn (2021). "Conservative data sharing for multi-task offline reinforcement learning". In: *Advances in Neural Information Processing Systems* 34, pp. 11501–11516.

Yu, X., W. He, Q. Li, Y. Li, and B. Li (2020). "Human-robot co-carrying using visual and force sensing". In: *IEEE Transactions on Industrial Electronics* 68.9, pp. 8657–8666.

Yu, Y. (2018). "Towards Sample Efficient Reinforcement Learning." In: *IJCAI*, pp. 5739–5743.

Zakershahrak, M., S. R. Marpally, A. Sharma, Z. Gong, and Y. Zhang (2021). "Order matters: Generating progressive explanations for planning tasks in human-robot teaming". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 3751–3757.

Zeng, A. (2019). "Learning visual affordances for robotic manipulation". PhD thesis. Princeton University.

Zeng, A., S. Song, S. Welker, J. Lee, A. Rodriguez, and T. Funkhouser (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. et al. (2022). "Robotic pick-and-place of novel objects in clutter with multi-affordance grasping and cross-domain image matching". In: *The International Journal of Robotics Research* 41.7, pp. 690–705.

Zhang, A., R. McAllister, R. Calandra, Y. Gal, and S. Levine (2020). "Learning invariant representations for reinforcement learning without reconstruction". In: *arXiv preprint arXiv:2006.10742*.

Zhang, Y. and M. M. Zavlanos (2020). "Transfer reinforcement learning under unobserved contextual information". In: *2020 ACM/IEEE 11th International Conference on Cyber-Physical Systems (ICCPS)*. IEEE, pp. 75–86.

Zhang, Y., P. Sun, Y. Yin, L. Lin, and X. Wang (2018). "Human-like autonomous vehicle speed control by deep reinforcement learning with double Q-learning". In: *2018 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, pp. 1251–1256.

Zhang, Y., S. Li, K. J. Nolan, and D. Zanotto (2019). "Adaptive assist-as-needed control based on actor-critic reinforcement learning". In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 4066–4071.

Zhao, D., H. Wang, K. Shao, and Y. Zhu (2016). "Deep reinforcement learning with experience replay based on SARSA". In: *2016 IEEE symposium series on computational intelligence (SSCI)*. IEEE, pp. 1–6.

Zhao, K., Y. Wang, Y. Zuo, and C. Zhang (2022). "Palletizing Robot Positioning Bolt Detection Based on Improved YOLO-V3". In: *Journal of Intelligent & Robotic Systems* 104.3, p. 41.

Zhao, W., J. P. Queralta, and T. Westerlund (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.

Zhong, J., T. Wang, and L. Cheng (2021). "Collision-free path planning for welding manipulator via hybrid algorithm of deep reinforcement learning and inverse kinematics". In: *Complex & Intelligent Systems*, pp. 1–14.

Zhou, H., S. Li, Q. Lu, and J. Qian (2020). "A practical solution to deformable linear object manipulation: A case study on cable harness connection". In: *2020 5th International Conference on Advanced Robotics and Mechatronics (ICARM)*. IEEE, pp. 329–333.

Zhu, H., A. Gupta, A. Rajeswaran, S. Levine, and V. Kumar (2019). "Dexterous manipulation with deep reinforcement learning: Efficient, general, and low-cost". In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 3651–3657.

Zhu, J., B. Navarro, R. Passama, P. Fraisse, A. Crosnier, and A. Cherubini (2019). "Robotic manipulation planning for shaping deformable linear objects withenvironmental contacts". In: *IEEE Robotics and Automation Letters* 5.1, pp. 16–23.

Zhu, J. et al. (2022). "Challenges and outlook in robotic manipulation of deformable objects". In: *IEEE Robotics & Automation Magazine* 29.3, pp. 67–77.

Ziebart, B. D., A. L. Maas, J. A. Bagnell, A. K. Dey, et al. (2008). "Maximum entropy inverse reinforcement learning." In: *Aaai*. Vol. 8. Chicago, IL, USA, pp. 1433–1438.