LIÈGE université
Library

https://lib.uliege.be

matheO

https://matheo.uliege.be

Master thesis : Deep Reinforcement Learning for Robotic Grasping

Auteur : Fares, Nicolas

Promoteur(s) : Ernst, Damien; Sacré, Pierre

Faculté : Faculté des Sciences appliquées

Diplôme : Master : ingénieur civil en science des données, à finalité spécialisée

Année académique : 2021-2022

URI/URL : http://hdl.handle.net/2268.2/16288

Avertissement à l'attention des usagers :

Tous les documents placés en accès ouvert sur le site le site MatheO sont protégés par le droit d'auteur. Conformément aux principes énoncés par la "Budapest Open Access Initiative"(BOAI, 2002), l'utilisateur du site peut lire, télécharger, copier, transmettre, imprimer, chercher ou faire un lien vers le texte intégral de ces documents, les disséquer pour les indexer, s'en servir de données pour un logiciel, ou s'en servir à toute autre fin légale (ou prévue par la réglementation relative au droit d'auteur). Toute utilisation du document à des fins commerciales est strictement interdite.

Par ailleurs, l'utilisateur s'engage à respecter les droits moraux de l'auteur, principalement le droit à l'intégrité de l'oeuvre et le droit de paternité et ce dans toute utilisation que l'utilisateur entreprend. Ainsi, à titre d'exemple, lorsqu'il reproduira un document par extrait ou dans son intégralité, l'utilisateur citera de manière complète les sources telles que mentionnées ci-dessus. Toute utilisation non explicitement autorisée ci-avant (telle que par exemple, la modification du document ou son résumé) nécessite l'autorisation préalable et expresse des auteurs ou de leurs ayants droit.

Deep Reinforcement Learning for Robotic Grasping

This dissertation is submitted for the degree of

Master of Science in Data Science and Engineering

by

Nicolas FARES

Supervisors:
PROF. D. ERNST
PROF. P. SACRÉ

Internship advisor:
T. EWBANK

Jury:
PROF. D. ERNST
PROF. P. GEURTS
PROF. P. SACRÉ
PROF. L. WEHENKEL
T. EWBANK

Faculty of Applied Sciences
University of Liège

Academic year 2021-2022

Abstract

The development and deployment of robotic grasping systems in the industry help to improve the efficiency and productivity of one's production lines. Even though interesting for any industrial actor, those robotic systems require a significant upfront investment. This significant investment is composed of two primary types of costs: hardware and software. Thanks to recent developments in Deep Reinforcement Learning applied to robotic grasping through vision-based systems, IntegrIA is researching solutions that could reduce the software costs of robotic grasping applications focused on pick-and-place tasks.

Thus, this master's thesis implements a state-of-the-art reinforcement learning algorithm named QT-Opt and aims to compare it with IntegrIA's one. Both online and offline learning versions of QT-Opt are developed, resulting in three training algorithms to compare across three training datasets. Performances of resulting agents are quantitatively evaluated and qualitatively compared through metrics such as the normalised area under the success rate curve.

In the end, it is observed that this master thesis best agent trained on a dataset composed of 1,800 objects achieves a grasping success rate of 96.67% on previously unseen objects, against 97.32% for IntegrIA's agent. Even though it cannot outperform their implementation, it is interesting to observe that the best agent trained for this master's thesis achieves the 96% success rate from the original paper while being powered with a fraction of its resources.

Acknowledgements

Firstly, I would like to express my gratitude to Tom, who has been there from the beginning of this work, answering any questions I might have. He helped orientate my research and allowed me to integrate the IntegrIA team fully. This master's thesis would not have been possible without him. Secondly, I would like to thank my promoters, Damien and Pierre, for their precious pieces of advice and encouragement during the writing of this manuscript.

I would also like to thank the IntegrIA team members, Audrey and Loïc, who helped me daily during the work development.

Thanks to Julien and Maxence for their support and valuable feedback on the manuscript. Special gratitude to Hélène for reviewing it and the insightful ideas that emerged from our conversations.

Finally, I would like to thank Marion, my family and friends for their unconditional support and always allowing me to keep a balance between work and life.

Table of contents

List of figures ii

1 Introduction 1

1.1 Industry Context 1
1.2 IntegrIA 2
1.3 Research question 3
1.4 Thesis outline 4

2 Background 5

2.1 Reinforcement Learning 6
2.2 Robotic Grasping 12

3 Problem Settings & Algorithms 17

3.1 Approach to Robotic Grasping 17
3.2 Hardware Constraint 19
3.3 Problem Settings 22
3.4 Algorithms 27

4 Methodology 31

4.1 Object Database 31
4.2 Data Collection 32
4.3 Domain Randomisation 36
4.4 Reward Shaping 37
4.5 Network Architecture 38

5 Experiments 40

5.1 Detailed Training 40

5.2 DDQL versus QT-Opt 45
5.3 Discussion 54

6 Conclusion **56**
6.1 Further work 57

References **58**

Appendix A Additional Results from Experiments **63**
A.1 QT-Opt vs. DDQL 64

Appendix B Detailed Algorithms **73**
B.1 DDQL 73
B.2 QT-Opt 74
B.3 Cross-Entropy Method 75

List of figures

1.1 Pie chart showing the costs repartition of a robotic system. Source: IntegrIA . . . 2

2.1 Thorndike's Cat Box . . . 6
2.2 Interaction of a Learning Automata and its environment . . . 9
2.4 Illustration of AlexNet's architecture distributed on 2 GPUs (Krizhevsky et al., 2012). . . . 12
2.5 Interaction of an agent and its environment in Reinforcement Learning . . . 15

3.1 Vision module example . . . 18
3.2 Original data collection setup from QT-Opt (Kalashnikov et al., 2018). It can be seen that the surroundings of the robot as well as its model are perceived by the over-the-shoulder camera. . . . 19
3.3 Simulator of the environment . . . 24

4.1 Objects from the dataset . . . 33
4.2 Original image from depth camera and its preprocessed equivalent . . . 36
4.3 Network architecture . . . 38

5.1 Selected objects from the global dataset . . . 43
5.2 Success rate with and without forces measured against agents trained with DDQL, offline QT-Opt, and online QT-Opt on the total set of objects evolving over the number of training iterations . . . 47
5.3 Success rate with and without forces measured against agents trained with DDQL, offline QT-Opt, and online QT-Opt on the airplane object evolving over the number of training iterations. . . . 48

5.4 Success rate with and without forces measured against agents trained with DDQL, offline QT-Opt, and online QT-Opt on the semi-sphere object evolving over the number of training iterations. 49
5.5 Evolution of the Q-values and the discounted rewards measured with the area under the curve across timesteps for DDQL, offline QT-Opt, and online QT-Opt against the three training datasets. 52
5.6 Evolution of the Q-values and the discounted rewards measured with the coefficient of variation across timesteps for DDQL, offline QT-Opt, and online QT-Opt against the three training datasets. 53

A.1 Metrics from DDQL (blue lines), offline QT-Opt (red lines), and online QT-Opt (green lines) trained on the global set of objects against the number of training iterations. 64
A.2 Mean Q-values from DDQL (blue lines), offline QT-Opt (red lines), and online QT-Opt (green lines) trained on the global set of objects against the number of training iterations for each timestep. 65
A.3 Discounted rewards from DDQL (blue lines), offline QT-Opt (red lines), and online QT-Opt (green lines) trained on the global set of objects against the number of training iterations for each timestep. 66
A.4 Metrics from DDQL (blue lines), offline QT-Opt (red lines), and online QT-Opt (green lines) trained on the airplane object against the number of training iterations. 67
A.5 Mean Q-values from DDQL (blue lines), offline QT-Opt (red lines), and online QT-Opt (green lines) trained on the airplane object against the number of training iterations for each timestep. 68
A.6 Discounted rewards from DDQL (blue lines), offline QT-Opt (red lines), and online QT-Opt (green lines) trained on the airplane object against the number of training iterations for each timestep. 69
A.7 Metrics from DDQL (blue lines), offline QT-Opt (red lines), and online QT-Opt (green lines) trained on the semi-sphere object against the number of training iterations. 70
A.8 Mean Q-values from DDQL (blue lines), offline QT-Opt (red lines), and online QT-Opt (green lines) trained on the semi-sphere object against the number of training iterations for each timestep. 71

A.9 Discounted rewards from DDQL (blue lines), offline QT-Opt (red lines), and online QT-Opt (green lines) trained on the semi-sphere object against the number of training iterations for each timestep. 72

Chapter 1

Introduction

1.1 Industry Context

A robotic arm, also called an articulated arm or industrial robot, is often described as a mechanical arm. This mechanical device comprises several joints that allow it to move in multiple directions and rotate along multiple axes. These robots are controlled through computer interfaces, allowing industry actors to program them to perform specific tasks. Those tasks are often linked to the manufacturing and production industry with industrial applications.

They can be used in many environments since they are fast, reliable, and precise. They could, for example, be used in factories to automate the execution of repetitive tasks such as painting. They can also perform a task that requires much precision since sensors monitor every joint articulated motion. Object manipulation or welding are good examples of those tasks needing precision. Using robotic arms in the industry maximises efficiency and productivity, as they operate 24 hours a day, seven days a week (author at Intel, 2020). Finally, robotic arms allow companies to benefit from a competitive advantage in their respective markets.

In addition to this competitive advantage they provide, they also reduce the risks incurred by workers. Indeed, robotic arms can be used in hazardous environments during tasks showing potential danger to humans.

Indeed, using robotic arms in factories allows to improve productivity, reduce worker-related risks, and maximise profits. According to a 2015 report by The Boston Consulting Group (Harold L. Sirkin, February 2015), the average global labour-cost savings could reach 16% by 2025, or 9% savings in Belgium alone.

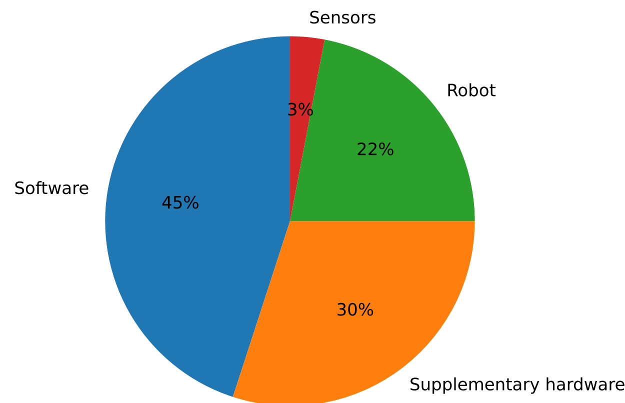


Fig. 1.1 Pie chart showing the costs repartition of a robotic system. Source: IntegrIA

However, deploying robotic arms is expensive and requires initial investments. A robotic system's costs are divided into hardware and software. The former accounts for 55% of the cost, while the latter represents 45% of it (fig. 1.1), according to data provided by IntegrIA.

A quick prospective analysis shows that thousands of euros are requested only for the robot. On top of that comes the purchase of a support on which the robot rests, an industry-grade computer necessary to control the robot, and finally, a suitable fixation is needed at the end of the arm for the application requested.

The software part of the budget represents the robot's development cost. Indeed, most of the industry uses traditional robots that require reprogramming for every new application on which they are deployed. This reprogramming is unavoidable as the system assumes the environment is deterministic, i.e., the repeated task is precisely the same at each iteration.

1.2 IntegrIA

IntegrIA is a research project composed of three research engineers from the University of Liège. The team led by Tom Ewbank agreed to have this master's thesis realised as part of an internship with them. This internship involves taking part in daily meetings and working with them in the university's offices.

IntegrIA's project focuses on facilitating the development and deployment of robotic arms specialised in the task of robotic grasping. To this end, IntegrIA is currently developing a robotic grasping agent using artificial intelligence that can generalise the task across objects of different shapes. The business goal is to reduce the software development cost linked to deploying a robotic system, which can be tackled from two different approaches. The first is to develop an efficient learning pipeline for well-defined robotic grasping problems. The second one develops an "off-the-shelf" agent reusable for each industry application around the task of robotic grasping. Both approaches solve the problem by minimising the time required to program the robotic system, decreasing the associated labour.

The learning pipeline would train a new model with the specifics of the problem for every new application. This type of learning system would be hyper-specialised for the problem, eventually solving it with very high accuracy. On the other hand, it would not be able to generalise to other use cases. The solution using an "off-the-shelf" agent consists of training a hyper-general model in the use cases it can work. This type of agent generally trains on a dataset of objects with various shapes and sizes, forced to learn a general behaviour abstracted from the specifics of objects. These two approaches to solving the business problem contrast between academic projects and industrial approaches. The former being interested with general and complex learning methods, while the latter aims only for higher success.

1.3 Research question

The IntegrIA team has based its work on a training algorithm named DDQL (van Hasselt et al., 2015). This master's thesis on the other hand builds upon DDQL a training algorithm named QT-Opt (Kalashnikov et al., 2018). Both of these algorithm are covered section 3.4.

The question fuelling this thesis is the following:

Does agents trained with the QT-Opt algorithm demonstrate better performance over agents trained with the DDQL training algorithm, when evaluated against multiple metrics ?

1.4 Thesis outline

The structure of the master's thesis is the following:

- **Chapter 2: Background.** This chapter briefly details the origins of reinforcement learning contrasted by the evolution of deep learning. Then presents how the combination of reinforcement learning and deep learning solves the problem of robotic grasping.
- **Chapter 3: Problem Settings & Algorithms.** This chapter mathematically describes the problem of robotic grasping, explains the choices for the selected approach and motivates them.
- **Chapter 4: Methodology.** This chapter covers all methods used during the experimentation as well as the main dataset on which the reinforcement learning algorithms were used.
- **Chapter 5: Experiments.** This chapter focuses on the experiment realised with the training algorithms, presents the results from quantitative assessments on the resulting agents, and discusses those results.
- **Chapter 6: Conclusions.** This chapter concludes this master's thesis work by summarizing its contributions, and then presents leads for future research on the problem defined by IntegrIA.

Chapter 2

Background

The field of reinforcement learning (RL) is concerned with methods that allow an agent to learn how to make decisions in an unknown environment. The agent gains the ability to accomplish a task in an environment that may be arbitrarily complex. Unlike humans, RL agents can learn from thousands of simultaneous interactions between themselves and their environment, limited only by the computing power available.

Two powerful examples of reinforcement learning techniques, both coming from the Google-funded company DeepMind, are their AlphaGo agent (Silver et al., 2016) and their solution to the protein folding problem (Jumper et al., 2021).

In this chapter, the development of both the knowledge and methods used nowadays in reinforcement learning applications is depicted. In particular, section 2.1 explains the development of reinforcement learning with a distinction of the work that was realised before the deep learning revolution and after (section 2.1.1, section 2.1.2, respectively). Finally, section 2.2 depicts different approaches to the task of robotic grasping.

2.1 Reinforcement Learning

2.1.1 Before the Deep Learning Era

Animal Learning

The origins of reinforcement learning are twofold, coming from animal learning on the one hand and optimal control on the other. Animal learning was initially introduced by the psychologist Edward Thorndike, who theorised trial-and-error learning, and defined the *law of effect* (Thorndike, 1911). This law suggests that responses that produce a satisfying effect in a particular situation become more likely to occur again in that situation, and responses that produce a discomforting effect become less likely to occur again in that situation. Furthermore, as the intensity of pleasure or pain increases, the pursuit or deterrence from the action increases as well. To demonstrate his theory, he used a puzzle box consisting of a door controlled by a lever placed inside it. He then placed a cat inside the box, which, to escape, had to activate the lever. To reward the cat when it correctly operated the lever to escape, food was placed outside the box as a reward. He finally noticed that the cat, once placed back in the box after having escaped, would operate the lever more and more quickly to get out (Thorndike, 1927).

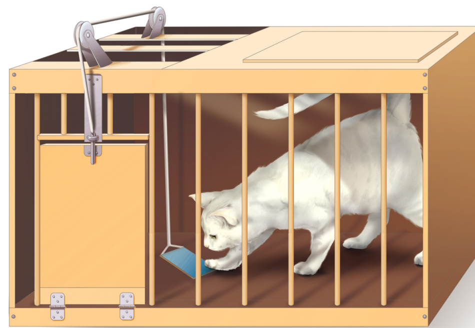


Fig. 2.1 Thorndike's Cat Box [1]

Optimal Control

Optimal Control was first developed as a branch of mathematical optimisation which defines optimisation methods to derive control policies for a system over an extended period of time, to achieve a desired goal (Pontryagin, 1987). Richard Bellman later introduced the Principle of

[1] https://terriermandotcom.blogspot.com/2012/05/thorndikes-cat-box.html

Optimality (definition 2.1.1.1 Bertsekas (2005)) as well as Dynamic Programming (DP) (Bellman, 1957). DP consists of algorithmic methods that solve Optimal Control problems.

Definition 2.1.1.1: Principle of Optimality

Let $\pi^* = \left\{\mu_0^*, \mu_1^*, \ldots, \mu_{N-1}^*\right\}$ be an optimal policy for the basic problem, and assume that when using π^*, a given state x_i occurs at time i with positive probability. Consider the sub-problem of being at state x_i at time i with goal to minimise the "cost-to-go" from time i to time N

$$E\left\{g_N\left(x_N\right)+\sum_{k=i}^{N-1} g_k\left(x_k, \mu_k\left(x_k\right), w_k\right)\right\}.$$

Then the truncated policy $\left\{\mu_i^*, \mu_{i+1}^*, \ldots, \mu_{N-1}^*\right\}$ is optimal for this sub-problem.

The Principle of Optimality states that, given an optimal policy for a problem (e.g. a sequence of roads to follow to go from Paris to Marseille), a sub-problem starting from a state in the optimal policy (going from Lyon to Marseille and Lyon is on the way from Paris to Marseille), then the optimal policy for this sub-problem is a truncated version of the original optimal policy (follow the same road from Paris to Marseille starting from Lyon).

Bellman Equation

Markov Decision Process (MDP) are a discrete stochastic version of the optimal control problem introduced by Howard (1960). It allows to represent the dynamical system with a discrete time-variable and actions are taken at every discrete time step, leading to a sequential decision-making problem. As rewards are evaluated at each time step, the goal is to maximise the long-term performance measured by the total reward cumulated over the course of the episode.

The research led to the functional equation known as the Bellman Equation:

$$V(x_i) = \max_{a_i}\{F(x_i, a_i) + \beta V(x_{i+1})\} \tag{2.1}$$

where

- V is the value function
- x_i is the state at time i

- a_i is the action performed at time i
- F is the reward function
- β is the discount factor

The Bellman Equation tells that, to be optimal according to some reward function F, the agent has to select the action a_i that maximises the value function V.

Learning Automata

During the early 1970s, learning automata have been studied as a model of learning (Narendra and Thathachar, 1974). A learning automata is the combination of an unknown environment and a stochastic automata as fig. 2.2 shows. The actions from the automata are the inputs to the environment. The responses from the environment are in turn the inputs to the automata and influence the updating of the action probabilities.

Different models of learning automata exists according to the input set of the automata:

- P-model: the input set is binary, e.g., $\{0,1\}$
- S-model: the input set is an interval, e.g., $[0,1]$
- Q-model: the input set is a finite collection of distinct symbols, e.g., obtained by quantisation

The reinforcement schemes applied to these learning automata are quite simple. Taking the P-model automaton and let zero be the non-penalty response and one the penalty response, if the learning automaton selects an action α_i at instant n and a non-penalty response occurs, the action probability $p_i(n)$ is increased and all other components of $p(n)$ are decreased. On the other hand, if a penalty response occurs, $p_i(n)$ is decreased and all other components of $p(n)$ are decreased (Narendra and Thathachar, 1974).

Temporal Difference

Richard Sutton first introduced Temporal Difference (TD) learning during his PhD dissertation in 1984 and then published in 1988 his paper (Sutton, 1988). The goal of TD learning, which draws its inspiration from mathematical differentiation, is to create precise reward predictions from delayed rewards. To forecast rewards, TD seeks to combine immediate rewards with predictions for the next time step. The most recent prediction is contrasted to what was anticipated with new knowledge when the following time step occurs. If there is a disparity, the algorithm computes the

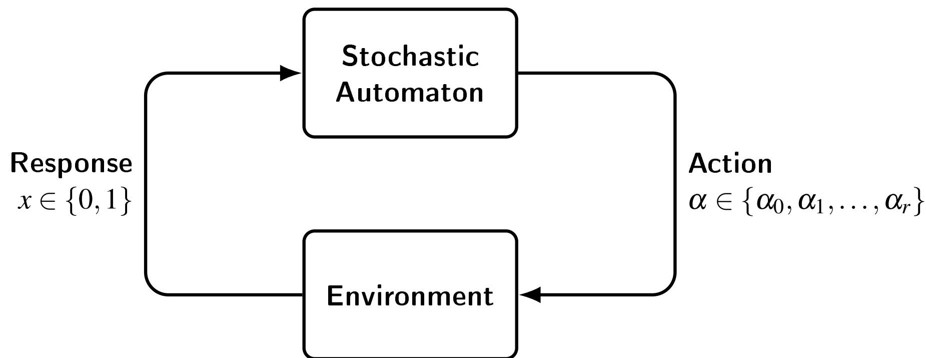


Fig. 2.2 Learning automata

error, or "temporal difference," to adjust the previous forecast towards the current one. To ensure that the entire chain of predictions gradually improves in accuracy, the algorithm seeks to bring the old and new predictions closer together at each time step.

TD Gammon

The goal of TD-Gammon (Tesauro, 1994) is to investigate the capacity of multilayer neural networks trained using TD(λ) to learn intricate nonlinear functions. The neural network that powers TD-Gammon is set up in a typical multi-layer perceptron (MLP) architecture. One way to think of the MLP architecture is as a universal approximator of nonlinear functions. A feed-forward flow of activation from the input nodes to the output nodes, passing via one or more layers of internal nodes referred to as hidden nodes, computes its output. Each connection in the network is parameterised by a real-valued weight.

Each of these nodes in the network produces a real number that is equal to the weighted linear sum of the inputs coming into it. It is then squashed by a nonlinear sigmoidal operation into the unit interval. The neural network may compute nonlinear functions of its input thanks to the squashing function's nonlinearity, and the specific function that is used relies on the weights' actual values.

Q-Learning

In 1989, Chris Watkins published his PhD thesis titled "Learning from Delayed Rewards" (Watkins, 1989), which introduced a model of reinforcement learning as incrementally optimising control of a markov decision process. In this thesis, Chris Watkins proposed Q-learning as a method to

learn optimal control directly without modelling the transition probabilities or expected rewards of the MDP. In 1992, Watkins and Peter Dayan offered a convergence proof. Given a state for an agent adhering to a policy, a Q-value function demonstrates the effectiveness of a particular action. Q-learning is the process of applying the Bellman Equation to iteratively update the Q-values for each state-action pair until the Q-function eventually converges to optimality (referred to as Q*). A model-free reinforcement learning method called Q-learning is capable of handling stochastic transitions and rewards without the need for modifications.

2.1.2 During the Deep Learning Era

Deep learning is a subset of machine learning (ML), which is a subset of artificial intelligence (AI) itself. The idea of artificial intelligence has been around since the 1950s, and its main objective is to enable computers to think and reason similarly to humans. ML is concerned with how to enable machines to learn without being explicitly taught as part of giving them the ability to think. By building increasingly intricate hierarchical models that imitate how people absorb new knowledge, deep learning goes beyond machine learning.

The Vanishing Gradient Problem

Glorot and Bengio (2010) explain in their paper the difficulty of training deep neural networks. They show that deep neural nets are hard to train because the features learned from the last layers do not flow back towards the first layers (see fig. 2.3). This problem leads to saturation in the neural net, making it hard and slow to train. They show that this phenomenon, known as the *vanishing-gradient problem*, is caused by two factors. First, a poor initialisation scheme of the weights of the neural network. Second, the use of non-linear activation functions, such as the hyperbolic tangent, the sigmoid function, or the softsign function, between hidden layers.

AlexNet

Deep convolutional neural networks (CNNs) are at the core of outstanding deep learning advancements. CNNs have been used to handle character recognition problems since the 1990s (Le Cun et al., 1997). However, their widespread use nowadays is owing to more recent research, in which a deep CNN (AlexNet) was used to outperform the state-of-the-art in the ImageNet picture classification challenge (Krizhevsky et al., 2012).

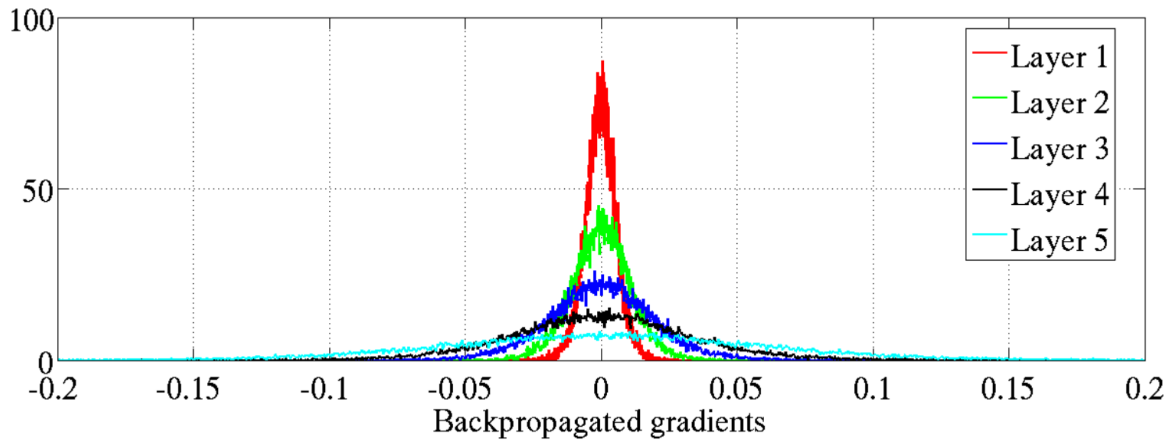


Fig. 2.3 Backpropagated gradients normalised histograms (Glorot and Bengio, 2010). Gradients for layers far from the output vanish to zero.

These networks, while being very effective at the task of computer vision, are notoriously hard to train because of their deep architecture and the vanishing gradient problem resulting from it.

As shown fig. 2.4, AlexNet is a deep neural network composed of multiple convolutional layers, pooling layers and dense layers. To train this network and get around the vanishing gradient problem, Krizhevsky et al. (2012) selected a specific activation function, the Rectified Linear Unit (ReLu) (Hahnloser et al., 2000) function, and initialised the weights of their network specifically to accelerate the early stages of learning.

It showed that arbitrarily deep neural networks were trainable thanks to this activation function that helps gradients flow back from output layers to input ones.

Deep Reinforcement Learning

Deep Reinforcement Learning (Deep RL) consist of reinforcement learning techniques applied to deep neural network. In 2013, DeepMind published a paper (Mnih et al., 2013) in which they studied the development of an agent trained with a variant of Q-Learning, called Deep Q-Learning (DQL). Their agent was benchmarked against seven Atari 2600 Games and it appeared that it outperformed all previous approaches on six of the games and surpasses a human expert on three of them (Mnih et al., 2013). They used deep CNNs to extract high-level features from raw video data in complex environments and train them to learn successful control policies.

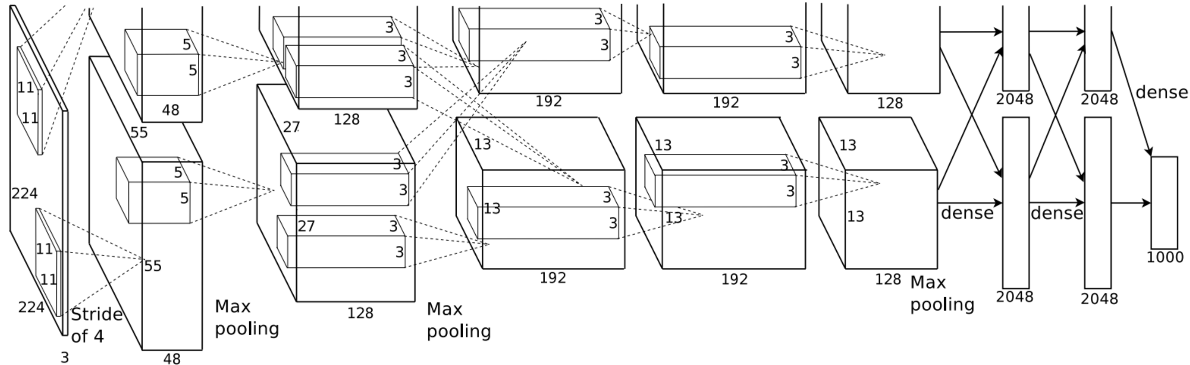


Fig. 2.4 Illustration of AlexNet's architecture distributed on 2 GPUs (Krizhevsky et al., 2012).

2.2 Robotic Grasping

The task of robotic grasping has been researched in literature for a long time. At first, the lack of proper sensors did not enable robotic arms to perceive their environment, thus requiring manual control over the mechanical arm and hands. As new types of sensors were being developed, robotic arms could grasp objects by mimicking the behaviour of human hands. However, this process still required human labour and the robots were not considered intelligent.

Thanks to the development of optical sensors, robotic arms could automatically grasp objects through a vision-based perception system (Du et al., 2019).

2.2.1 Approaches to Robotic Grasping

There exists different approaches to vision-based robotic grasping, depending on multiple criteria. Analytic approaches analyse the shape of the object to identify an optimal grasp pose. Data-driven approaches rely on machine learning and have made significant progress due to data availability, better computational resources and algorithmic techniques.

To further categorise data-driven approaches, one can identify model-based techniques and model-free techniques. The former takes advantage of specific knowledge about the target object's shape to solve the considered task. The latter aims to generalise to never-seen-before objects (Kleeberger et al., 2020).

Model-Based Robotic Grasping

Model-based robotic grasping techniques can be decoupled as a three-stage process: object poses are estimated, then a grasp pose is determined, and finally the path to pick the object is planned.

The object pose estimation process consists in estimating the translation and rotation from a reference frame of the target object. Formally, it can be expressed as a regression problem with six dimensions (3 degree of rotation, 3 degree of translation).

As the creation of annotated datasets is time-consuming, very tedious, and does not scale, the trend is to use synthetic datasets created from simulations. Indeed, they automatically provide flawless ground-truth annotations to train models with. A downside of training models in simulations is the necessity to assert that capability of transferring the application from simulation to real-world.

Model-Free Robotic Grasping

Model-free approaches, in contrast with model-based ones, do not use prior knowledge on the objects they work with. They therefore do not estimate the pose of the objects. These techniques are interesting because they generalise well to unseen objects, which can be desired depending on the use-case.

2.2.2 Supervised Learning for Robotic Grasping

Supervised learning techniques aim to define a mapping based on the dataset on which they train. Once again, two categories define the approaches. They can be discriminative if the grasp configuration is the input or generative if it is the output. The first approach consists of sampling grasp candidates and ranking them using a neural network. The system then executes the highest-scoring candidate. The second approach generates a grasp configuration directly, assumed to be optimal, without comparing it to others.

Dex-Net

Dex-Net (short for Dexterity Network) is a contribution from Berkeley Autolab[2] to the problem of robotic grasping through a vision-based system. Their first publication, Dex-Net 1.0 (Mahler et al.,

[2] https://autolab.berkeley.edu/projects

2016), presents a dataset including 10.000 unique 3D object models and 2.5 million parallel-jaw grasps associated with their algorithm.

Dex-Net 1.0 uses a novel deep learning method called Multi-View Convolutional Neural Network (MV-CNN) to classify 3D objects. This neural network finds prior 3D objects and grasps data stored in the Dex-Net dataset similar to the one processed by the model. Using a generated set of candidate grasps for the new object, a Multi-Armed Bandit model evaluates each grasp combined with the prior data retrieved from the dataset to predict the probability of force closure P_f. In their work, Mahler et al. consider the different sources of uncertainty to improve their predictions by defining those as random variables following gaussian distributions. The parameters of those random variables are then updated during the training of their model. Overall, the system takes advantage of the computing power provided by 1500 virtual cores hosted by the Google Cloud Platform.

Dex-Net 2.0 (Mahler et al., 2019) builds on top of Dex-Net 1.0 to improve the evaluation of grasp candidates. Using the Dex-Net 1.0 dataset and its associated algorithms to evaluate gasp candidates, they created a new dataset composed of 3D models, point clouds, grasp and analytical metrics to train a Grasp Quality Convolutional Neural Network (GQ-CNN). This GQ-CNN evaluates a candidate's grasp associated with a depth image to produce a probability of success under uncertainty.

2.2.3 Reinforcement Learning for Robotic Grasping

Deep reinforcement learning is promising and proves to be influential in grasping objects. Policies can learn how to effectively push or shift objects to ensure that they are graspable in a clutter setting (Zeng et al. (2018), Berscheid et al. (2019)).

QT-Opt

QT-Opt (Kalashnikov et al., 2018) is a deep reinforcement learning framework developed to solve the problem of learning vision-based dynamic manipulation skills with a scalable approach. In their paper, they formalise the dynamic system as an MDP. At each time step, the policy observes the state of the system through an RGB image from an over-the-shoulder camera. The policy then selects an action that the robotic arm will realise. If the agent holds the object above a certain

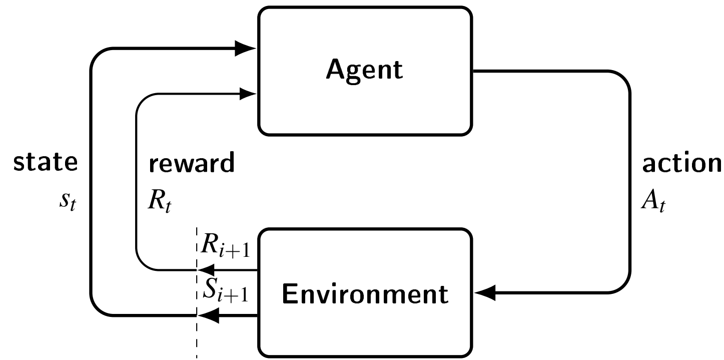


Fig. 2.5 The agent–environment interaction in reinforcement learning (Sutton and Barto, 2018).

height, it observes a reward of one, zero otherwise. The interaction between the agent and its environment is illustrated fig. 2.5.

The approach leveraged over 580k real-work grasp attempts, collected with seven robots over four months, for 800 robot hours. Over 1.2 million parameters compose the deep neural network representing the Q-function. It achieved a grasp success rate of 96% to previously unseen objects, which demonstrates that the algorithm from Kalashnikov et al. can generalise the features learned.

Learning policies for the problem of robotic grasping enabled by vision-based perception is a challenging task. On-policy algorithms are not an option since generalisation requires diverse data, but recollecting experience on a variety of objects after each policy update is impractical. To answer this challenge Kalashnikov et al. (2018) developed a scalable off-policy reinforcement learning system based on continuous Q-learning generalisation.

In opposition to other work (Lillicrap et al., 2016), QT-Opt (Kalashnikov et al., 2018) does not use a second network to sample actions from a state. Instead, they use a stochastic optimisation method over the actions with the Q-network as the objective function. The optimisation method selected is the Cross-Entropy Method (CEM), which is robust to local optima and easy to parallelise (Rubinstein and Kroese, 2004).

Seven robots collected transitions from the real world to scale up the algorithm. A distributed replay buffer database stores these transitions alongside historical off-policy transitions loaded from the disk. A thousand running jobs called "Bellman updater" are continuously labelling the data with the target Q-values by performing the CEM optimisation.

As explained, reinforcement learning in vision-based robotic grasping is challenging. To mitigate this, Kalashnikov et al. have done some experiments to improve learning stability. They

found that applying the Cross-Entropy Loss (Zhang and Sabuncu, 2018) (CE) to train the neural network is better than the standard Mean Square Error (MSE) as the total rewards are bounded between $[0,1]$. The QT-Opt (Kalashnikov et al., 2018) system employs two target networks, updating the first by Polyak-Averaging (Polyak and Juditsky, 1992) technique and the second being a lagged version of the first.

Chapter 3

Problem Settings & Algorithms

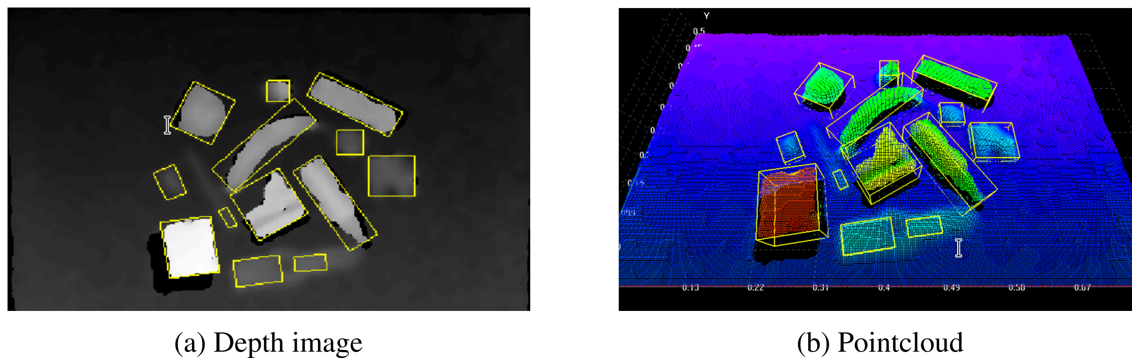
This section provides a detailed description of the robotic graping problem and the general approach IntegrIA uses to solve it. More specifically, section 3.1 introduces the modules composing the overall system. Section 3.2 presents the hardware available for this master's thesis and explains how it drives the learning framework. Section 3.3 mathematically defines the robotic problem and its corresponding MDP. Finally, section 3.4 details the algorithm used to train robotic grasping policies.

3.1 Approach to Robotic Grasping

The choices made for this master's thesis problem settings directly result from the previous work and choices done by the IntegrIA team. Future applications of their project focus on pick-and-place tasks, in which a robot grasps an object (e.g. on a conveyer belt) and then places it somewhere. A three-step process defines the pick-and-place task.

1. Object detection
2. Object grasping
3. Object placement

The IntegrIA project is focused only on the first and second steps, which they choose to tackle independently with two modules, the vision module and the grasping module.



(a) Depth image (b) Pointcloud

Fig. 3.1 Example the Easy3DObject library with the Intel RealSense D435 camera. Detected objects are surrounded by a yellow bounding box

The vision module is responsible for detecting and localising the target object. This module provides segmentation masks of the objects it detects to the grasping module. The task this module tackles is known as "category-agnostic instance segmentation". The goal is to define subsets of pixels belonging to the same physical object without classifying the object type. This choice enables the vision module to work with unseen objects, thus being robust to future applications. Once the vision module has detected the target object, the robotic arm is positioned on top of it for the grasping module to take over.

IntegrIA initially tested two detection modules to fulfill this task: the Easy3DObject detection module (fig. 3.1) from Euresys (S.A.) and the Synthetic RGB-D Fusion Mask-RCNN (SF Mask-RCNN) (Back et al., 2020) neural network. Unfortunately, the Easy3DObject module does not work well in cluttered environment, which is required for future applications. Hence, the SF Mask-RCNN neural network was selected to build the vision module.

The grasping module is based on a closed-loop system to avoid challenges from the open-loop setting. Indeed, open-loop grasping techniques determine a grasp candidate and the robotic arm executes the movement towards that candidate. It requires a precise calibration process between the camera and the robot as well a path-planning module to compute the motion of the robot. Finally, open-loop systems are not able to dynamically react to perturbations occurring during the motion of the arm.

Closed-loop systems on the other hand, are able to dynamically react to any change that may occur thanks to the feedback flow of information. Furthermore, these systems does not require a calibration process between the camera and the arm. Finally, it is possible to define closed-loop

Fig. 3.2 Original data collection setup from QT-Opt (Kalashnikov et al., 2018). It can be seen that the surroundings of the robot as well as its model are perceived by the over-the-shoulder camera.

interaction with an environment from a policy learned with RL techniques, which is the main study of this master's thesis.

3.2 Hardware Constraint

To solve the challenge of robotic grasping from vision-based perception, the IntegrIA team uses a depth camera mounted on the wrist of the robotic arm. This choice is motivated by the robustness it gives to future applications. Indeed, the system remains independent of the type of surroundings and the robotic arms themselves. The original setup for the data collection task by QT-Opt (fig. 3.2) shows that the over-the-shoulder camera captures the surroundings and the robotic arm of the systems. Since trained agents on that setup capture information related to the surroundings and specifics of that setup, it then becomes specific to it. In other words, the same agent deployed on a new application would probably not demonstrate the same behaviour as previously. On the other hand, this problem does not impact wrist-mounted cameras with a close-enough view of the object (fig. 3.1, fig. 4.2). The only specific the camera would perceive is the model of the gripper.

The IntegrIA team chose to wrist-mount a depth camera instead of an RGB camera as Kalashnikov et al. did. The first reason is that a depth camera improves knowledge transfer from simulation to reality by abstracting the textures and luminosity. Second, depth cameras give more information about the scene than RGB ones.

The training environment used in reinforcement learning has multiple components that necessitates different type of resources. The amount of available resources is therefore a decision factor in the architecture of the overall system. To explain the challenges created by this choice, it is necessary to detail the different components.

The replay buffer is the memory of the system, storing passed transitions to later redeliver them to be reprocessed again. The sampler interacts with the environment by collecting episodes to be stored in the replay buffer. Note that samplers can either be implemented in simulation with CPU cores, or in real world with robotic arms. The actions performed by the sampler in the environment are selected by the policy being trained at the same time. The system eventually uses samplers to evaluate the policy, however those episodes are not stored in the replay buffer. Finally, the trainer pulls transitions from the replay buffer, estimates the Q-values with its internal parameters and updates them according to the target Q-values from the transitions.

Each component requires a different type of resources. The trainer uses GPU computing power to compute and update the gradients of the error with respect to its internal parameters. CPU cores and robotic arms power samplers to collect episodes in parallel to each other, respectively in simulation or the real world. Finally, the replay buffer requires RAM memory to store episodes.

Two distinctions exist in reinforcement learning: offline learning and online learning. The former takes advantage of previously collected transitions stored in a replay buffer by iteratively sampling a batch of transitions and training a Q-function with them. In addition, the latter collects new transitions with its trained policy and replaces stored ones in the replay buffer according to an eviction scheme. The online learning algorithm then repeats this process.

Once again, each setup requires different resources and have advantages. Online learning necessitates samplers to continuously collect new episodes while offline learning does not. On one hand, the latter is interesting for systems with limited number of computing power as it requires less of it. On the other hand, the former provides a more thorough type of learning since the policy train on itself.

Three sources of computing power are available for the realisation of this master's thesis:

- NIC5: a cluster of CPUs managed by the CÉCI[1].
- Alan: a cluster of both CPUs and GPUs managed by a team from the Montefiore Institute[2].

[1] https://www.ceci-hpc.be/clusters.html
[2] https://github.com/montefiore-ai/alan-cluster

- Quadros: a machine hosted by the IntegrIA team.

Note that NIC5 and Alan are shared among researchers. A table presenting the hardware of each cluster is available at table 3.1.

One can understand the architectural choices made by IntegrIA in the training system by contrasting the computational resources available at Uliège with those used to develop QT-Opt and Dex-Net. On one hand, Kalashnikov et al. developed QT-Opt mixing offline and online learning. Their replay buffer was filled with transitions collected by seven robots in the real world but also with previously collected transitions, stored in an offline database. They also used a thousand CPU cores to update target Q-values before processing them with the GPU trainer. On the other hand, Mahler et al. trained Dex-Net 1.0 entirely in simulation with 1,500 CPU cores.

The learning framework developed by IntegrIA is run completely in simulation. Indeed, collecting real-world episodes necessitates constant monitoring to reboot the system in case of failure and ensure that the robots do not damage the setup or themselves. Furthermore, IntegrIA do not possess enough robots to perform data collections in a reasonable amount of time.

As explained before, both GPU and CPU computing power are necessary to train policy and collect transitions, respectively. However, the proportion of time used to update the policy over the time necessary to collect transitions is very low. Hence, IntegrIA chose to perform offline learning by collecting episodes with multiple CPU cores on NIC5 and train policies with GPUs, e.g. from Alan. This iterative process of collecting and training is interesting because it does not monopolise GPU resources that, most of the time, would not be used.

As shared resources (NIC5 and Alan) are highly solicited among researchers, a long waiting time can exist, justifying the necessity of IntegrIA to host a cluster (Quadros), providing a consistent source of GPU computing power. Finally, IntegrIA possess a UR5-e series robotic arm from Universal Robots[3] to evaluate trained policies in real world. this master's thesis however is realised purely in simulation as it is assumed that IntegrIA posses efficient sim-2-real transfer techniques.

[3]https://www.universal-robots.com/fr/

Table 3.1 Hardware information of clusters

Cluster	GPU type	CPU type		CPU count	RAM/node
NIC5	None	AMD EPYC 7542	2.9GHz	4672 (73x64)	256Gb. . . 1Tb
Alan	20 x GeForce RTX 1080Ti	Intel Xeon E5-2620 v4	2.10GHz	160 (10x16)	252Gb
	39 x GeForce RTX 2080Ti	Intel Xeon Silver 4116	2.10GHz	192 (8x24)	256Gb
		Intel Xeon Silver 4216	2.10GHz	64 (2x32)	256Gb
	16 x Quadro RTX 6000	Intel Xeon Gold 5218	2.30GHz	128 (4x32)	503Gb
	4 x Tesla V100	Intel Xeon Gold 6248	2.50GHz	80 (2x40)	503Gb
Quadros	4 x Quadro RTX 5000	Intel Core i9-10980XE	3GHz	36 (1x36)	188Gb

3.3 Problem Settings

This master's thesis aims at tackling the problem of performing a robust planar parallel-jaw grasp on a set of objects of various shapes individually placed on a table through vision-based perception. The algorithms developed in this master's thesis train a function that inputs an action and a depth image combined with information about the state to predict the observed reward, bounded between 0 and 1. The same stochastic optimisation process from QT-Opt (Kalashnikov et al., 2018) retrieves the policy learned.

3.3.1 Markov Decision Process

Let us consider discrete-time dynamics. Note that the first technical report done by the IntegrIA team inspired the definition of this system, as they defined the problem's constraints.

Let $T \in \mathbb{N}$ be the time horizon of the system, i.e., the number of steps in each episode. The system is defined through the following objects:

- A state space S
- An action space A
- A disturbance space W
- A transition function $f : S \times A \times W \to S$
- A bounded reward function $\rho : S \times A \times W \to [0, B_r] \subset \mathbb{R}$

- A conditional probability distribution P_w giving the probability $P_w(\cdot|s_t,a_t)$ of drawing a disturbance $w_t \in W$ when taking an action $a_t \in A$ while being in state $s_t \in S$.

A probability measure P_0 yields the probability $P_0(s_0)$ of each state $s_0 \in S$ being the initial one.

At each timestep $t \in \{0,\dots,T-1\}$, the system evolves from state s_t to state s_{t+1} with $s_t, s_{t+1} \in S$, under the action $a \in A$ and random disturbance $w \in W$ drawn with probability $P_w(w_t|s_t,a_t)$, according to the transition function f:

$$s_{t+1} = f(s_t, a_t, w_t) \tag{3.1}$$

The reward function associates a reward r_t with every transition done:

$$r_t = \rho(s_t, a_t, w_t), \quad |r_t| \leq B_r \in \mathbb{R} \tag{3.2}$$

The set $(S, A, W, f, \rho, P_w, P_0, T)$ defining this MDP is referred to as the environment.

Let Π be the set of all policies. The policy $\pi \in \Pi$ is defined as a function mapping the state s_t to a probability distribution over A at each timestep $t \in \{0,\dots,T-1\}$. This policy is stochastic and used by the agent to select action $a_t \sim \pi(\cdot|s_t,t)$ based on the information s_t provided at each timestep. Let $h = (s_0, a_0, w_0, \dots, s_{T-1}, a_{T-1}, w_{T-1})$ be the trajectory from instant time 0 to $T-1$. The cumulative reward signal $R(h)$ over trajectory h is defined as:

$$R(h) = \sum_{t=0}^{T-1} \gamma^t r_t \tag{3.3}$$

where $0 \leq \gamma \leq 1$ is a discount factor, $r_t = \rho(s_t, a_t, w_t)$ and $s_{t+1} = f(s_t, a_t, w_t)$.

The expected cumulative reward (a.k.a. expected return) of a policy $\pi \in \Pi$ starting from the initial state $s_0 \in S$ is given by

$$V^\pi(s_0) = \mathop{\mathbb{E}}_{\substack{a_t \sim \pi(\cdot|s_t,t), \\ w_t \sim P_w(\cdot|s_t,a_t)}} \left[\sum_{t=0}^{T-1} \gamma^t r_t \right] \tag{3.4}$$

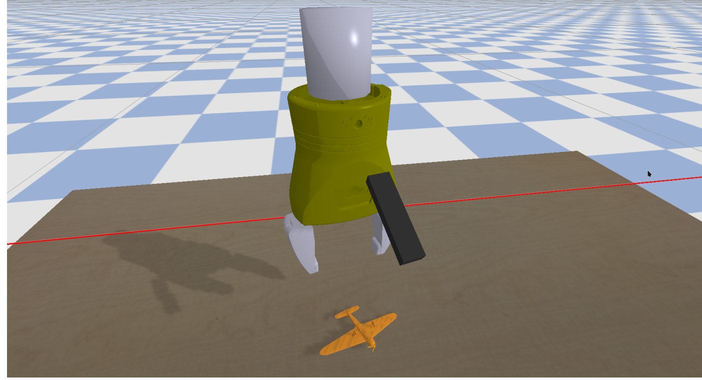


Fig. 3.3 Simulator of the environment. The black rectangle is the wrist-mounted depth camera.

An optimal policy $\pi^* \in \Pi$ is a policy such that

$$\pi^* \in \arg\max_{\pi \in \Pi} V^{\pi}(s), \quad \forall s \in S \tag{3.5}$$

3.3.2 Problem Statement

A robotic arm with a two-finger parallel jaw gripper as an end effector composes the grasping system. This grasping system uses a depth camera mounted on the wrist of the arm to perceive its environment. The robotic arm's joint dynamics are abstracted to simplify the simulations and to learn policies agnostic to the robotic arm. The end-effector is modelled as a "floating" gripper, as can be seen fig. 3.3.

The goal of the system is to grasp the object positioned on the table. This object comes from a dataset of objects that can take various shapes and poses on the table to learn general policies. The IntegrIA team assumes that the vision module, responsible for detecting objects on the table, correctly performs the task and computes an initial position for the gripper a few centimetres above the object. The agent then has to perform the remaining moves to grasp the object in as few steps as possible.

Partially Observable Markov Decision Process

Let the environment defined in section 3.3.1,

$$(S, A, W, f, \rho, P_w, P_0, T)$$

This environment is partially observable for the agent, requiring the use of a Partially Observable Markov Decision Process (POMDP). In opposition to classical MDPs, the agent does not have all the information about the current state s_t. It only has a perception of it through an observation $o \in O$ drawn from the observation space O with a sensor model $o \sim P_o(\cdot|s_t)$.

The objective is to find an agent with a behaviour modelled by a stochastic policy $a_t \sim \pi_\theta(\cdot|s_t)$, with parameters $\theta \in \Theta$, maximising the expected cumulative reward. The mathematical expression of the objective is:

$$\theta^* \in \underset{\theta \in \Theta}{\arg\max}\, V(\theta), \quad \forall \theta \in \Theta \tag{3.6}$$

$$V(\theta) = \underset{\substack{s_0 \sim P_0(\cdot), \\ a_t \sim \pi_\theta(\cdot|s_t,t), \\ w_t \sim P_w(\cdot|s_t,a_t)}}{\mathbb{E}} \left[\sum_{t=0}^{T-1} \gamma^t r_t \right] \tag{3.7}$$

Definitions

State space S All information necessary to solve the optimisation problem defines the states$s_t \in S$ at time t. The state representation contains the pose of the gripper's end-effector $p_{EE,t}$ and the opening of the gripper. It also contains the target object obj with its relative pose $p_{obj,t}$ to $p_{EE,t}$[4].

Action space A Let continuous action $a_t = [dx, dy, dz, d\phi, g, e_{term}] \in A$ where:

- $[dx, dy, dz] \in [-1, 1]^3$ is the linear Cartesian displacement of the end-effector.
- $d\phi \in [-1, 1]$ is the rotation of the gripper around the z-axis.
- $g \in \{0, 1\}$ is the gripper opening command with 0 to close the gripper and 1 to open it.
- $e_{term} \in \{0, 1\}$ is the episode termination command.

The displacements are expressed in the reference frame of the gripper. The rotation and the translation of the gripper are performed concurrently. The continuous values corresponding to displacements are normalised between -1 and 1 to improve the agent's training. The factors v and w allow getting back the actual displacement in meters for translations and radians for rotation from their corresponding actions. These factors define the maximum amplitude of translation and

[4] The pose of an object is defined as $p_{obj,t} = [\mathbf{t}_{obj,t}, \mathbf{q}_{obj,t}]$ where $\mathbf{t}_{obj,t} = [x, y, z]$ is the 3D position of the center of gravity of the object and $\mathbf{q}_{obj,t} = q_w \langle q_x, q_y, q_z \rangle$ is the quaternion representing the 3D rotation of the object around its center of gravity.

rotation that a single move can perform.

A safety mechanism constantly measures the force applied to the object by the gripper and interrupts the move if a force greater than a threshold F_{safety} is measured. If the mechanism is triggered, the gripper performs a slight upward displacement dz_{safety}. Furthermore, the agent is not allowed to change the opening of the gripper while realising a displacement in the gripper position. If both commands are detected, the gripper opening command has priority over the displacement one. Finally, e_{term} allows the agent to terminate the episodes before reaching the maximum number of steps. If e_{term} equals zero, the episode continues, and the system performs the actions. Otherwise, it ignores all actions and transitions towards a terminal state.

Disturbance space W **and probability distribution** P_w The disturbance is linked to the simulated environment, which is prone to computational errors and unexpected behaviours. Another source of error is the depth camera, which does not capture with perfect accuracy the depth of the objects. For more convenience however, IntegrIA chose to assume the system to be deterministic.

Observation space O As the environment is only partially observable, the environment presents only limited information about its state to the agent. The agent percepts the state s_t at time t through the observation o_t drawn from the sensor model $P_o(\cdot|s_t)$. This information vector contains a snapshot image $i_t \in I$ (I being the image space) of the setup, taken from the wrist-mounted camera. In addition, a binary variable representing the opening of the gripper is given, 0 if the gripper is closed and 1 otherwise. Finally, the agent also gets information about the current timestep t.

Initial state distribution P_0 The simulation randomly draws an object obj from a set of objects at the beginning of each episode. The initial pose of the object $p_{obj,0}$ may vary, but its location is guaranteed to be on the table, in line of sight of the vision module. The initial pose of the end effector $p_{EE,0}$ depends on the initial state drawn from P_0. The end-effector is centred above the object, with a distance H_0 above its highest point, ensuring its visibility from the end-effector's camera.

Transition function f Let $s_t \in S$ be a non-terminal state at time t, a_t be the action at time t. If $t = T - 1$, then the transition function leads to a terminal state s_{term}. Otherwise, the transition function leads to the state $s_{t+1} \in S$ according to eq. (3.1). Finally, if the value of the action

command e_{term} is 1, for all $t \in \{0, \ldots, T-1\}$, the rest of the action is ignored and the state is updated to a terminal state s_{term}.

Reward function ρ The reward function returns the null value from a transition leading to a non-terminal state. If the transition from s_t to $s_{t+1}, \forall t \in \{0, \ldots, T-1\}$ by performing action a_t leads to s_{t+1} equals to s_{term}, then the reward function evaluates the environment of the gripper to decide whether to attribute a reward or not. The simulation performs an upward displacement of the gripper without changing its opening. If the parallel fingers hold the object after the vertical displacement, then a reward of one is attributed. Otherwise, the reward is null.

Constants The values for the constant parameters of the POMDP are the following:

- Maximum horizon: $T = 10$
- Displacement scale factor: $v = 4\text{cm}$
- Rotation scale factor: $w = 1.5\text{rad}$
- Initial height above the object: $H_0 = 5\text{cm}$
- Force threshold for the safety mechanism: $F_{safety} = 50\text{N}$
- Upward displacement after safety stop: $dz_{safety} = 2\text{cm}$

3.4 Algorithms

The following section summarises the theory of the RL training algorithms used in this master's thesis. These algorithms are the Deep Double Q-Learning (DDQL) (van Hasselt et al., 2015) implemented by IntegrIA and the version of QT-Opt (Kalashnikov et al., 2018) realised for this master's thesis. The motivations behind DDQL come from papers Quillen et al. (2018) & Kalashnikov et al. (2018) which demonstrated that DDQL performs well on robotic grasping tasks and is relatively robust with respect to hyperparameters.

The objective of solving the problem of robotic grasping is to find an optimal policy π^* for the robotic agent to use. This policy can be recovered from the optimal Q-function $Q^{\pi^*}(s,a)$ defined

as the maximum expected return when being in state s and performing action a. The Bellman optimality condition defines the Q-function:

$$Q^{\pi^*}(s_t, a_t) = \underset{w_t \sim P_w(\cdot|s_t, a_t)}{\mathbb{E}} \left[r_t + \gamma V^{\pi^*}(f(s_t, a_t, w_t)) \right] \tag{3.8}$$

where

$$V^{\pi^*}(s) = \underset{a' \in A}{\arg\max} Q^{\pi^*}(s, a') \tag{3.9}$$

is also known as the value-function.

Reinforcement learning techniques estimate the optimal Q-function Q^{π^*} through a parametrised Q-function Q_θ with parameters $\theta \in \Theta$ from the parameter space.

3.4.1 DDQL

The Deep Double Q-Learning algorithm (van Hasselt et al., 2015) tackles the problem of overestimation of the Q-values that exist in the original Q-learning algorithm (Mnih et al., 2013). To estimate Q^{π^*}, DDQL minimises the Bellman error, i.e., the difference between the target Q-values Q_T and the learned Q-values Q_θ:

$$B(\theta) = \underset{(s_t, a_t, s_{t+1}) \sim U(D)}{\mathbb{E}} \left[Q_\theta(s_t, a_t) - Q_T(s_t, a_t, s_{t+1}) \right] \tag{3.10}$$

where D is the offline replay buffer composed of tuples (s_t, a_t, r_t, s_{t+1}), $U(D)$ is a uniform distribution over D and

$$Q_T(s_t, a_t, s_{t+1}) = \underbrace{\rho(s_t, a_t)}_{r_t} + \gamma \underset{a' \in A}{\arg\max} Q_{\theta'}(s_{t+1}, a') \tag{3.11}$$

Note that it is not the main network Q_θ that evaluates the target Q-values Q_T but a delayed version of it $Q_{\theta'}$. This second network (called the target network) allows to reduce the correlation between learned Q-values and their estimation, resolving the overestimation problem. DDQL updates the target network's parameters θ' with the main network's parameters θ after a predefined number of gradient updates. The details of the algorithm are presented algorithm 1.

3.4.2 QT-Opt

The QT-Opt algorithm uses components that allow for more regular and stable training (Kalashnikov et al., 2018). To learn the optimal parameters θ^*, the algorithm minimises the following Bellman error:

$$B(\theta) = \underset{(s_t,a_t,s_{t+1})\sim U(D)}{\mathbb{E}} [\Delta(Q_\theta(s_t,a_t), Q_T(r_t,s_{t+1}))] \tag{3.12}$$

Kalashnikov et al. chose to apply the cross entropy function as a divergence metric Δ between Q_θ et Q_T. The motivations are that the rewards of each episode are bounded between 0 and 1 and the fact that they found empirically that this measure of divergence brought more stability during training. Moreover, Q_T is evaluated through two target networks θ'_1, θ'_2 which are delayed versions of the main network θ. θ'_1 is the exponential moving average version of θ with an averaging constant of 0.9999, while the second version θ'_2 is a delayed version of θ'_1 by 6000 gradient steps.

The target Q-values Q_T are then computed according to:

$$Q_T(s_t,a_t,s_{t+1}) = \underbrace{\rho(s_t,a_t)}_{r_t} + \gamma V_{\theta'_1,\theta'_2}(s_{t+1}) \tag{3.13}$$

where

$$V_{\theta'_1,\theta'_2}(s_{t+1}) = \min_{i=1,2} Q_{\theta'_i}(s_{t+1}, \underset{a'\in A}{\arg\max} Q_{\theta'_1}(s_{t+1},a')) \tag{3.14}$$

This corresponds to a combination of Polyak Averaging (Polyak and Juditsky, 1992) and clipped double Q-learning (Fujimoto et al., 2018).

Online Learning and Offline Learning

Two versions of QT-Opt are developed for this master's thesis: an online and an offline one. Offline learning minimises the Bellman error, eq. (3.12), where the transitions (s_t,a_t,s_{t+1}) are uniformly sampled from the replay buffer D:

$$(s_t,a_t,s_{t+1}) \sim U(D)$$

This replay buffer contains a fixed set of transitions that have been previously collected and does not evolve over time.

On the other hand, online learning minimises the same Bellman error where the replay buffer does not contain a fixed set of transitions anymore. On the contrary, online learning uses the

newly trained policy to collect episodes and store them in the replay buffer. As its size is fixed, some transitions are randomly deleted to free space for the new ones when the buffer is full.

The details of the offline learning version of the QT-Opt algorithm are presented in algorithm 2. The online version's details add parallel workers that sample episodes according to their policy version, update the replay buffer with those transitions and then update their policy's parameters.

Chapter 4

Methodology

This chapter covers the specifics of the training framework. It is divided in five parts. First, section 4.1 presents the database of 3D model objects. Second, section 4.2 explains the data collection process with its predefined policies and why it is necessary. Section 4.3 and section 4.4 detail processes used to engineer the rewards and facilitate the shift from simulation to reality. Finally, section 4.5 provides the architecture of the network.

4.1 Object Database

The database of objects used to train the agents combines multiple sources of 3D objects:

- Random object dataset from Pybullet (Coumans and Bai, 2016–2021) containing 1,000 objects with random shapes procedurally generated.
- Dex-Net 1.0 dataset (Mahler et al., 2016) containing thousands of ordinary objects built from various sources. The IntegrIA team selected only the following objects from Dex-Net 1.0: 13 synthetic adversarial object meshes from Thingiverse, 1,371 synthetic object meshes from the 50 category subset of 3DNet, and 129 laser scans from the KIT Object Database.
- EGAD dataset (Morrison et al., 2020) composed of over 2,000 generated objects to train and evaluate vision-based perception robotic grasping systems. The objects constituting EGAD are various, from simple to complex shapes and from easy to hard to grasp. This particularity makes it unique compared to other datasets built for robotic grasping, which are sometimes limited in size or by the number of objects for each class.

- ABC dataset (Koch et al., 2019) is a collection of a million Computer-Aided Design (CAD) models built for research of geometric deep learning methods and applications.
- FabWave dataset (Binil Starly et al., 2019) intended to spur innovations in product design and manufacturing research, including AI-based techniques and analytical methods, contains 100,000 CAD models.

The object database was built by the IntegrIA team and used throughout their experiments. Hence, this thesis uses their database to train and evaluate policies. It comprises 2,388 objects: 446 objects from the Pybullet random dataset, 1,154 from the Dex-Net 1.0 dataset, 446 from the EGAD dataset, 343 from the ABC dataset and 199 from the FabWave dataset. When training an agent, two datasets divide the object dataset. The training algorithm uses the first to train the agent and the second to evaluate its capacity to generalise to previously unseen objects. The two datasets are composed of 1,800 and 588 objects, respectively. A sample of objects from the evaluation dataset is shown fig. 4.1

4.2 Data Collection

Training an agent through reinforcement learning techniques is challenging, and the more complex the environment is, the more challenging it is. One early challenge is getting the agent to realise successful episodes. In robotic grasping, the challenge is to perform the first successful grasp of the target object to accumulate successful experiences. It is almost impossible to achieve this goal while performing complete random actions.

Custom-made policies collect episodes that will later be loaded into the replay buffer to train an agent to answer this challenge. It is equivalent to showing the agent how to achieve the first steps and then letting it train independently from experiences it accumulates.

4.2.1 Data Collection with a Heuristic Policy

The behaviour of the heuristic policy is to go towards the target object with an open gripper ($g = 1$), then eventually close it ($g = 0$) when deemed appropriate and go back up. Formally, the policy guides the gripper to move towards a random point of the object until it reaches a certain height $h_{thresh} \sim \mathbb{U}[0, h_{obj}]$ randomly sampled at the beginning of the episode. The coordinates (x_{target}, y_{target}) of the specific point are randomly sampled from the segmentation mask at each

(a) Car

(b) Bottle

(c) Airplane

(d) Cup

(e) Light bulb

(f) Industrial piece

Fig. 4.1 Objects from the dataset

timestep.
The heuristic returns the following actions until reaching h_{thresh}:

$$\begin{cases} dx = \left(x_{\text{target}} - x_{\text{gripper}}\right)/v \\ dy = \left(y_{\text{target}} - y_{\text{gripper}}\right)/v \\ dz \sim \mathbb{U}[-1,0] \\ d\phi \sim \mathbb{U}[-1,1] \\ g = 1 \\ e_{\text{term}} = 0 \end{cases} \tag{4.1}$$

Let suppose the gripper reaches the height h_{thresh} at timestep t. The following actions returned by the heuristic until the end of the episode $T-1$ consist of closing the gripper, moving upward, and terminating the episode. The randomly sampled step $t^* \sim \mathbb{U}[t+1, T-1]$ is the termination step during which the value of e_{term} is set to 1.
The behaviour of the heuristic from step t to the end of the episode is then:

$$\begin{cases} dx = dy = d\phi = 0 \\ dz \sim \mathbb{U}[0,1] \\ g = 0 \\ e_{\text{term}} = 1 \text{ if } t = t^* \text{ otherwise } 0 \end{cases} \tag{4.2}$$

4.2.2 Data Collection with a Random Policy

The random policy allows for exploration of the environment and collects unsuccessful episodes, which helps to understand what kind of behaviour is unproductive. The gripper opening g is selected randomly at each step. The episode will either terminate if $t = T$ or at random step t^* between $[0, T-1]$ by setting $e_{term} = 1$. Components dx, dy, d_ϕ of the action are random while dz is biased so that the robot moves downwards until reaching $h_{tresh} \sim \mathbb{U}[0, h_{object}]$, sampled at the beginning of each episode. Once the gripper reaches the threshold, dz is biased to move upwards.

4.2.3 Data Preprocessing

The episodes collected with the different heuristics are preprocessed before being used to train an agent. Initially, IntegrIA tested different types of preprocessing methods such as background subtraction and gripper pixel value modification. They concluded from their experiments that

downscaling the images and applying *clip normalisation* were the most effective processing methods.

Clip Normalisation

The process of clip normalisation can be used with both depth expressions. It models the limitation of the camera to perceive object that are outside a specific range of vision, depending on the model of the camera. The process first clips pixel values outside the range, and then normalises them between $[-1,1]$.

Let c_{min} and c_{max} be the minimal and maximal bounds on the pixel values, respectively. The pixels of the image are clipped between those values:

$$p_c = \begin{cases} c_{\max} & \text{if } p > c_{\max} \\ c_{\min} & \text{if } p < c_{\min} \\ p & \text{otherwise} \end{cases} \quad, \forall p \in P \tag{4.3}$$

with P being the set of pixel values. Then the pixels are mapped between $[-1,1]$ according to:

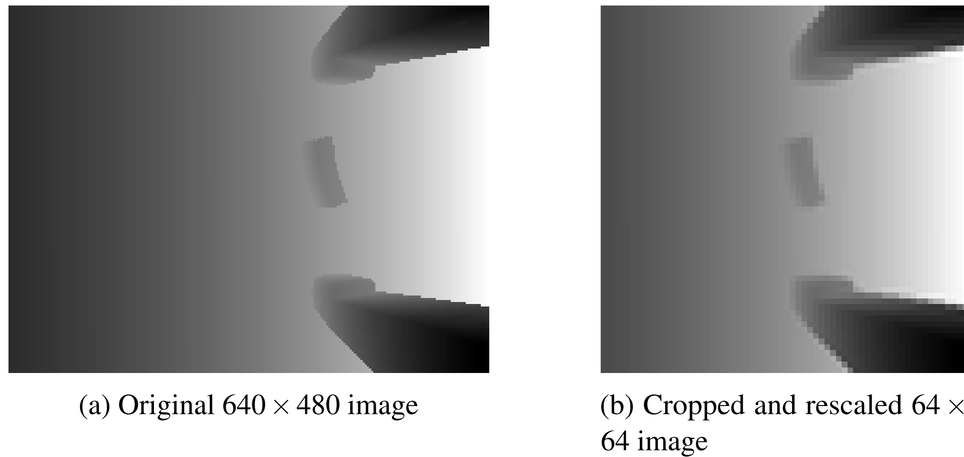
$$p_{cn} = \frac{p_c - c_{min}}{c_{max} - c_{min}} \quad, \forall p_c \in P_c \tag{4.4}$$

with P_c being the set of all clipped pixel values.

Downscaling

Initially, the captured images from the simulator are defined with a 640×480 resolution, showing clear edges of the object's shape. However, as the resolution of images increase, the number of parameters required to operate the neural network increases as well. The objective is thus to keep a maximum amount of information while reducing the size of the data. For images, it is achieved by rescaling them appropriately. The images are then cropped and downscaled to 64×64 to be passed as inputs to the neural network (fig. 4.2). This resolution is deemed appropriate for the task as it does not require a large number of parameters while keeping high-level details of the object.

Furthermore, this relatively low resolution can be used thanks to wrist-mounting the depth camera as it is closer to objects, quickly capturing details. To contrast this idea, as Kalashnikov et al. chose an over-the-shoulder camera for their QT-Opt setup, they have to capture an image of 472×472, growing their network more significant in parameters.



(a) Original 640×480 image

(b) Cropped and rescaled 64×64 image

Fig. 4.2 Original image from depth camera and its preprocessed equivalent

4.3 Domain Randomisation

Domain randomisation is the process of randomising the simulation parameters that might differ and vary in the real world. The process facilitates the transfer of policies from simulation to the real world. With the randomised parameters, the agent has to learn robust policies that it can use in a wide range of situations. Moreover, while the resulting policy's behaviour may not be the same in simulation and the real world, domain randomisation reduces the gap between both. The IntegrIA team chooses to randomise the following parameters:

- **Camera pose:** an offset is applied to the position and orientation of the camera relative to the gripper. The x, y, and z are drawn from the uniform distribution $\mathbb{U}[-5,5]$ mm while the yaw, pitch, and roll are drown from $\mathbb{U}[-5,5]^{\circ}$. It is essential to randomise those parameters as it is impossible to get the same position from the simulation in the real world.
- **Camera field of view (FOV):** an offset is applied to the camera's field of view. This offset tackles the fact that the FOV of the real camera may not be the same as the simulation's one.
- **Image noise:** depth images from simulations are close to perfect as the ground truth depth is available. It is impossible to get the actual ground truth depth of the objects in the real world. Thus, randomisation is necessary to reproduce the actual noise. The simulation uses a combination of Gaussian noise, gamma noise and Perlin noise to get more realistic images. The addition of Gaussian noise accounts for the proportional error of the depth in the actual setup. Adding gamma noise reproduces a type of noise specific to laser-based sensors. Perlin noise reproduces the wavy shape of flat surfaces rendered by the PMD

picoflexx[1] camera used to perceive the environment. As the exact distribution of the noise generated by the sensor is unknown, the parameters of each distribution are drawn from uniforms with arbitrary ranges. IntegrIA has empirically determined the ranges by trying to reproduce the real sensor output with the generated 3D point cloud.

- **Robot initial position:** an offset is applied on the position of the gripper $[x_0, y_0, z_0]$, being first placed just over the object. The uniform distribution $\mathbb{U}[-2,2]$cm draws the x, y, and z. This randomisation reproduces the uncertainty of the vision module to place the gripper above the object.

- **Object properties:** the pose, scale, mass, lateral friction, and spinning friction of the object to grasp are randomised. It represents the randomness of the different object properties in the real world.

4.4 Reward Shaping

Reward shaping is a technique inspired by animal training where extra rewards are provided to make a problem easier to learn. This method engineers a reward function to provide more frequent feedback on appropriate behavior (Wiewiora, 2010). In the task of robotic grasping, a custom penalty diminishes the reward whenever the gripper applies too much force on the target object. This new reward function should theoretically guide the agent towards policies that pick objects without damaging them. The new reward r_t^* given at time t becomes:

$$r_t^* = \begin{cases} r_t - p & \text{if } f_{t,\max} > F_{safety} \\ r_t & \text{otherwise} \end{cases} \tag{4.5}$$

where r_t is the initial reward at timestep t, p is the value of the penalty and

$$f_{t,\max} = \max_{j \in F_t} f_{t,j}$$

is the maximum force applied on the target object compared to the set of all forces F_t applied during the transition from s_{t-1} to s_t. As a reminder, F_{safety} is defined section 3.3.2.

[1] https://3d.pmdtec.com/en/3d-cameras/flexx2/

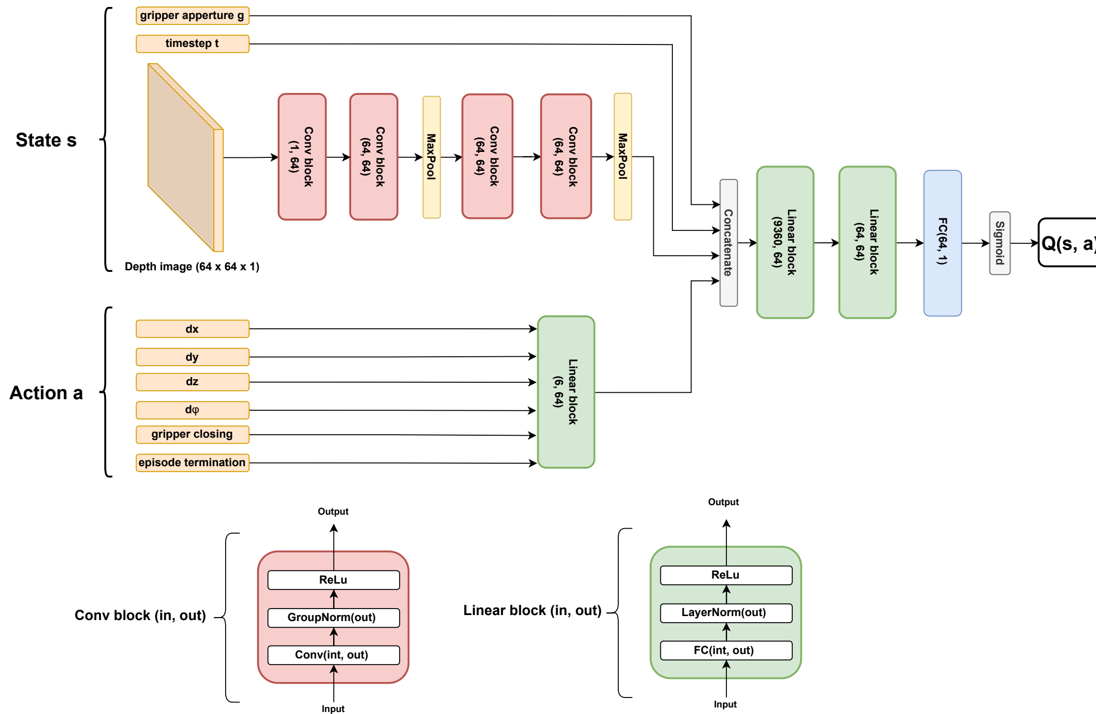


Fig. 4.3 Architecture of the grasping Q-function.

During the experiments, the force penalty p is not constant over time but evolves. Experiments empirically showed that it is best not to penalise forces during the beginning to train a viable policy and then apply a penalty to fine-tune it.

4.5 Network Architecture

This thesis models the Q-function as a deep neural network whose architecture, shown fig. 4.3, is inspired by the one presented in Levine et al. (2016) and Kalashnikov et al. (2018). The network takes the depth image component of the state s and processes it with four convolutional layers. A linear stack processes the six actions, and the result, combined with the two additional state features (timestep t and gripper opening g), is repeated to match the size of the processed depth image and then concatenated together. The resulting features are then processed twice by linear blocks, then once by a fully connected layer that reduces the features to a scalar value. This scalar is then gated through a sigmoid to ensure that Q-values $Q_\theta(s,a)$ are always in the range $[0,1]$.

Each convolutional block and linear block normalises their inputs after processing them and then uses the ReLu activation function (Hahnloser et al., 2000). The convolutional block normalises its batch with group normalisation (Wu and He, 2018) while the linear block uses a layer normalisation technique (Ba et al., 2016). The parameters of the convolutional layers are a kernel of 3x3, a stride of 1 and no padding. All weights of the network are initialised with the initialisation technique from Kalashnikov et al. (2018), a truncated normal random variable ($\mu = 0,\ \sigma = 0.01$).

All experiments use the same network architecture throughout the thesis to model the Q-function.

Chapter 5

Experiments

This chapter covers the experimentation of this master's thesis. It is divided in three main parts. Section 5.1 explains the metrics which evaluates the trained agents, how objects are selected to train agents and the creation of a training dataset with those objects. Section 5.2 describes the results from the experimentation. Finally, section 5.3 discusses those results.

5.1 Detailed Training

5.1.1 Evaluation Metrics

Various metrics are used to measure and compare the performance of trained policies.

Success Rate Measures

The work done at IntegrIA principally focuses on improving the grasping success rate (SR) as well as the grasping success rate without forces($SR_{w\backslash forces}$). The former computes the number of successful episodes over the total number of trials, while the latter computes the number of successful episodes where no forces above the threshold exist over the total number of trials:

$$SR = \frac{\text{\# successful episodes}}{\text{\# episodes}}$$

$$SR_{w\backslash \text{ forces}} = \frac{\text{\# successful episodes without forces}}{\text{\# episodes}}$$

Area Under the Curve

The Area Under the Curve (AUC) is the integral of a function between an interval defined by two points on an axis. The cumulative sum over the interval replaces the integral for discrete-time dynamics.

Let $f : x \to y$ be a function from $x \in [x_0, x_N]$ to $y \in [y_0, y_M]$. Then the area under the curve AUC is:

$$AUC = \sum_{i=0}^{N} f(x_i) \tag{5.2}$$

In robotic grasping, this metric is interesting as it allows to compare the quality of a trained agent (e.g. with the success rate) over the number of training iterations performed. Moreover, since all agents do not share the same number of training iterations due to their training complexity, the AUC is normalised between [0, 1] to compare those policies.

Then, for the same function f, the normalised Area Under the Curve (AUC_{norm}) is:

$$AUC_{norm} = \frac{\sum_{i=0}^{N} f(x_i)}{(y_M - y_0) \times (x_N - x_0)} \tag{5.3}$$

For easier notation, the Normalised Area Under the Curve AUC_{norm} is referred to as AUC.

Coefficient of Variation

The coefficient of variation is defined as the ratio of the standard deviation of a series over its arithmetic mean. It is represented as a unit-less percentage (Cuddy and Valle, 1978). This coefficient will be used to compare the stability of training algorithms. The coefficient of variation CV is mathematically expressed as:

$$CV = \sqrt{\frac{\sum_{j}^{N} (y_j - \bar{y})^2}{N - 1}} \times \frac{100}{\bar{y}} \tag{5.4}$$

where N is the number of elements, y is the series elements and $\bar{y}$ is the arithmetic mean.

5.1.2 Object Selection

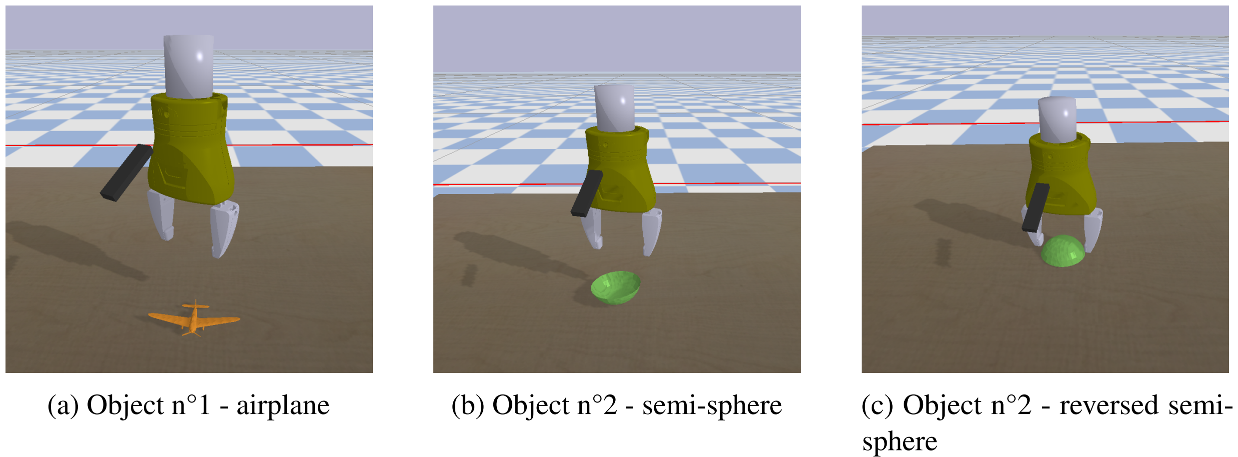
Experiments are reproduced three times against different sets of objects, ensuring that the behaviour from policies is not the result of a specific configuration. The first set of objects is described in section 4.1. As a reminder, this set is divided into a training dataset of 1,800 objects and a validation dataset of 588 objects.

The two other sets of objects are selected from the validation dataset, containing previously unseen objects only. Since it is interesting to train and evaluate agents on single objects, these two datasets contain only one object each.

First, an agent is trained with QT-Opt against the global dataset. Then two objects (fig. 5.1) from the evaluation dataset are selected to be trained on. The selection of object is based on the performance of the policy against these objects. In particular, the success rate and the success rate without forces are used to evaluate objects. The goal is to find objects on which the policy is partially successful, allowing improvement.

The first object is a small regular airplane. The agent performs 80% of successful episodes while trying to grasp it, but the success rate without forces drops to 42% (table 5.1). Even though the small airplane appears with numerous candidate grasps thanks to its symmetry, this object is not so easy to grasp without applied forces. Indeed, one understands that the airplane can easily rotate around the axis defined by the grasping points if the centre of gravity of that airplane does not belong to that axis. This type of rotation can be avoided by applying pressure on the points of contact, increasing the spinning friction forces. This airplane toy is a baseline for assessing the capacity of policies to find correct grasp candidates that do not require the application of force.

The second object consists of a semi-sphere. Once again, the agent performs 78% of success rate, which drops to 60% for the corresponding success rate without forces (table 5.1). Similarly to the airplane, the semi-sphere is composed of multiple grasp candidates. With this concave 3D shape, two types of grasp candidates exist. The first one has a grasping finger inside the concavity, while the second finger is placed outside the concavity. Note that this type of grasp is not possible with the configuration seen on fig. 5.1c. The second type of grasp candidate has both grasping fingers outside the concavity. However, this second type of grasp requires the object's centre of gravity to belong to the axis formed by the points of contact of the fingers. Otherwise, the object would be pushed to the side when closing the gripper.



(a) Object n°1 - airplane (b) Object n°2 - semi-sphere (c) Object n°2 - reversed semi-sphere

Fig. 5.1 Selected objects from the global dataset

Table 5.1 Success rate with and without forces of an agent trained by offline QT-Opt on the global dataset against two previously unseen objects.

Object	**Success rate**	**Success rate**$_{w\backslash forces}$
Airplane	80%	42%
Semi-sphere	78%	60%

5.1.3 Training Dataset

Each training algorithm requires a source of historical transitions in order to fill up its replay buffer. As explained section 3.2, episodes are collected with the NIC5 cluster (table 3.1) and two predefined policies, the heuristic and the random one (section 4.2.1 and section 4.2.2).

The data collection process collects a total of 199,800 episodes from the global dataset's 1,800 objects (111 episodes per object) under the heuristic policy. This dataset of transitions achieves a success rate of 71.4%. Additional 100,800 episodes are collected with the random policy (56 episodes per object), achieving 1.06% of success rate. This first dataset of transitions is then composed of 299,600 episodes and an average success rate of 47.97%.

The second and third datasets are similarly built. Each one comprises 200,000 episodes per object collected with the heuristic and 100,000 episodes per object collected with the random policy. The airplane dataset of transitions achieves a success rate of 60.18% with the heuristic and 1.16% with the random policy, averaging a success rate of 40.5%. Finally, the semi-sphere

Table 5.2 Number of episodes and success rates of the collected datasets with the random policy (section 4.2.2) and the heuristic policy (section 4.2.1).

Object dataset	Policy	Num. episodes	Success rate
Global dataset	Heuristic	199,800	71.40%
	Random	100,800	1.06%
Airplane	Heuristic	200,000	60.18%
	Random	100,000	1.16%
Semi-sphere	Heuristic	200,000	46.95%
	Random	100,000	1.33%

dataset of transitions achieves a success rate of 46.95% with the heuristic and 1.33% with the random policy, averaging a success rate of 31.74% (table 5.2).

5.1.4 Evaluation Dataset

Each trained agent requires a specific evaluation dataset based on the objects on which it was trained. The evaluation dataset for agents trained on the global dataset of 1,800 objects is built with the remaining 588 evaluation objects, as explained in section 4.1. Agents trained on the airplane object or the semi-sphere one are evaluated only on this one since the goal is to evaluate their performance only on the specific object and not on previously unseen ones.

Trained agents are evaluated through multiple episodes, each defined by its starting parameters. Those parameters are the specific evaluation object, its stable pose, its physical properties such as the lateral and spinning frictions, the mass, and its scale compared to the original 3D model.

IntegrIA provides the evaluation dataset for the global set of objects. Each one is represented five times with different parameters, resulting in 2,940 episodes on which agents are evaluated. Note that the physical properties of each object is defined by uniformly sampling from a specific range for each property.

Regarding the evaluation datasets for the selected objects, each one is represented through 200 stable poses, resulting in 200 episodes each. The physical properties are, however, fixed, as agents are evaluated on the exact same object, except the starting pose.

5.2 DDQL versus QT-Opt

This experiment aims at comparing the quality of both QT-Opt implementations (offline and online) against the DDQL training algorithm by evaluating the resulting policies trained with historical transitions collected from predefined policies (table 5.2).

Each training algorithm is evaluated against three sets of objects, resulting in nine trained policies. Each agent is compared through its success rate, success rate without forces, Q-values and discounted rewards at each timestep. The area under the curve (AUC) and coefficient of variation (CV) metrics assess all curves and compare them quantitatively. In addition, the maximum value of the success rate and success rate without forces is measured. Finally, the AUC and CV measures are also computed for the Q-values and discounted reward at each timestep, then aggregated across them to show the evolution of each measure across. The reader is invited to observe the Q-values and discounted rewards across timestep in appendix A as those would consume a significant amount of space in the main section.

Section 5.2.1 describes the general setup for this experiment by detailing the implementation of the algorithms and the hardware that powered them. Finally, section 5.2.2 contains a brief description of the results of the algorithms on each dataset individually, then compares them across the three datasets.

5.2.1 Setup

Implementation

The hyperparameters primarily follow the ones from the IntegrIA team. The reinforcement learning system is implemented with RLlib (Liang et al., 2017), a reinforcement learning framework providing implementations of state-of-the-art RL algorithms for two deep learning frameworks: PyTorch and TensorFlow. Models are optimised with Adam (Kingma and Ba, 2014) and the default parameters provided by RLlib. Each policy is trained with a buffer size of 250,000 and a batch size of 512 transitions.

Offline policies were trained on the datasets described in section 5.1.3. Online QT-Opt continuously samples episodes according to its policy to store them in its replay buffer and then train on them. However, as the agent is not equipped with prior knowledge about the task, it could take many training iterations before the first successful episodes. To accelerate the training, 10% of the replay buffer is filled with transitions collected from the heuristic policy, while 90% of the buffer contains new transitions. The association of successful historical transitions and

newly collected ones is interesting, as those new transitions can be considered as exploratory ones. The ratio between historical and new transitions has been empirically determined. Empirical tests showed that increasing the ratio of historical transitions does not accelerate the training. It has been decided that it is best to maximise the ratio of new transitions in order to explore the environment.

Hardware

Offline data collection was performed on the NIC5 cluster. Each application of a predefined policy on a specific set of objects to collect episodes required both data collection tasks and data preprocessing tasks. For each of these parts, 50 workers were allocated, for a total of 100 workers for each collection process. Since 6 collections were required to collect according to two policies on three object sets, 600 workers were requested and used on the NIC5 cluster to generate the datasets used in this experiment.

Models are trained on the Quadros cluster (table 3.1) with one *Quadro RTX 5000* from Nvidia, 12 CPU cores distributed among six evaluation workers, five rollout workers and one trainer. The evaluation workers are responsible for loading checkpoints of the policy and evaluating it. In contrast, the rollout workers are responsible for loading transitions from the replay buffer and estimating the expected Q-values. Finally, the trainer computes the loss gradients and performs the parameter updates of the deep neural network. The overall system takes advantage of 50Gb of RAM to perform the training.

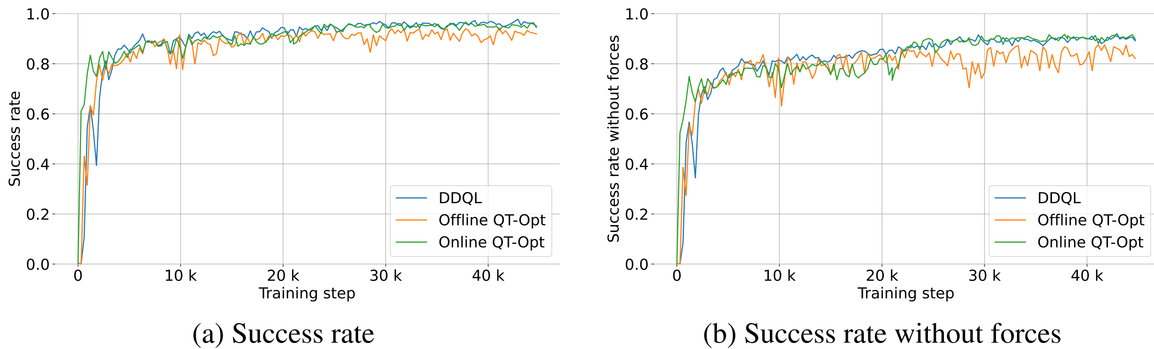
5.2.2 Results

This section describes the results obtained from evaluating three training algorithm: DDQL, offline QT-Opt, and online QT-Opt. The algorithms train three agents each, one for each training dataset described section 5.1.3. It results in nine agents that are compared qualitatively and quantitatively.

Global Dataset

The results show that both success rate measures, with and without forces, converge asymptotically to the unit value, this observation being valid for DDQL and both versions of QT-Opt (fig. 5.2). The success rate of DDQL achieves a maximum of 97%, while offline and online QT-Opt only achieves a maximum of 95% and 97%, respectively. On the other hand, if considering the forces, the average success rate without forces achieves only 92% for DDQL, 89% for offline QT-Opt and 92% online QT-Opt. Both offline algorithms demonstrate the similar stability throughout

learning, with a CV at 15% for DDQL, 13% for offline QT-Opt. At the same time, the online version of QT-Opt had the best stability with a lower CV at 10%. Finally, it appears that DDQL performs slightly better than online QT-Opt according to the AUC. Indeed, DDQL and online QT-Opt have an AUC of 90% for the success rate and 82% for the success rate without forces. On the other hand, offline QT-Opt achieves an AUC of 88% for the success rate and 79% for the success rate without forces (table 5.3).



(a) Success rate (b) Success rate without forces

Fig. 5.2 Success rate with and without forces measured against agents trained with DDQL (blue lines), offline QT-Opt (red lines), and online QT-Opt (green lines) on the total set of objects evolving over the number of training iterations

Table 5.3 Results (in %) of agents trained on the total set of objects

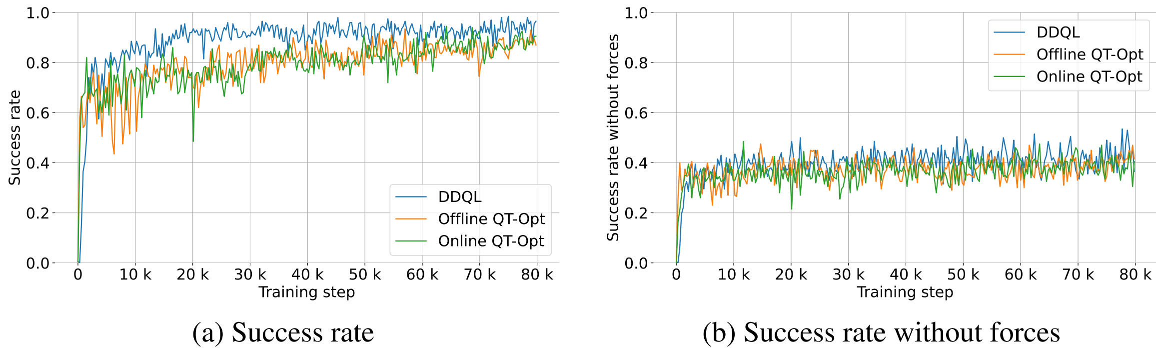
	Success rate			Success rate w\forces		
Agent	**Max**	**AUC**	**CV**	**Max**	**AUC**	**CV**
DDQL	**97.32**	**90.93**	14.94	**92.04**	**82.74**	15.70
Offline QT-Opt	95.33	88.06	12.89	88.71	79.43	13.62
Online QT-Opt	96.67	90.25	**10.23**	91.76	82.31	**12.48**

Airplane Toy

Results of the tests performed on the airplane toy show that the success rate of DDQL and QT-Opt converges asymptotically towards the unit value while the success rate without forces oscillates around 40% (fig. 5.3). The success rate of DDQL performs to a maximum of 99.49% while QT-Opt has a maximum of 96% for its offline version and 95% for the online one. Even though the success rate of QT-Opt seems unstable around 10k training steps, the curves remain stable overall, with a CV around 11% for each implementation.

Regarding the success rate without forces of the tests performed on the airplane toy, DDQL achieves 60% while QT-Opt is around 50% for both implementations. The stability of the curves varies between under 12% for QT-Opt and 15% with DDQL.

Finally, the AUC of DDQL performs a score of 90% for the success rate and 41% for the success rate without forces. Offline QT-Opt is at an AUC of 81% on the success rate and 38% for the success rate without forces, Finally, online QT-Opt performs similarly to its offline counterpart with an AUC of 79% for the success rate and 36% for the success rate without forces (table 5.4).



(a) Success rate

(b) Success rate without forces

Fig. 5.3 Success rate with and without forces measured against agents trained with DDQL (blue lines), offline QT-Opt (red lines), and online QT-Opt (green lines) on the airplane object evolving over the number of training iterations.

Table 5.4 Results (in %) of agents trained on the airplane

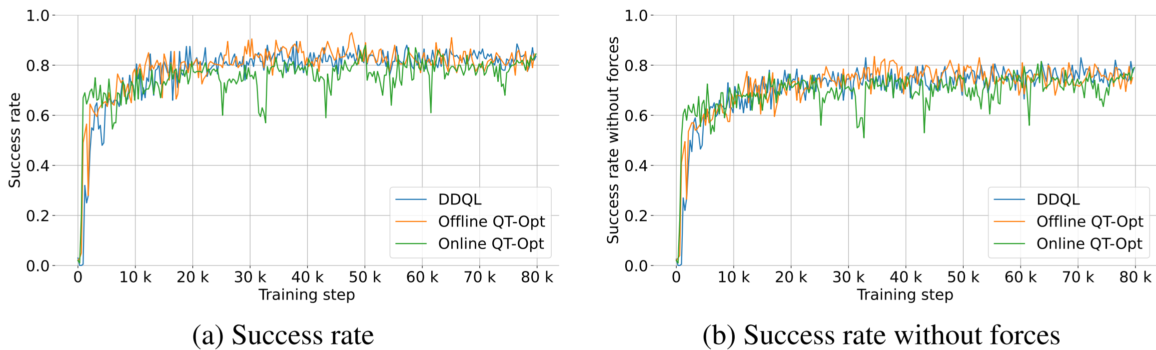
	Success rate			**Success rate w\forces**		
Agent	**Max**	**AUC**	**CV**	**Max**	**AUC**	**CV**
DDQL	**99.49**	**89.74**	**11.04**	**60.50**	**40.72**	15.01
Offline QT-Opt	96.00	81.34	11.96	50.50	37.74	**12.51**
Online QT-Opt	94.49	79.14	11.56	48.50	36.42	13.47

Semi-Sphere

When tested using the semi-sphere, DDQL and offline QT-Opt seem to follow the same trend regarding the success rate with and without forces. Indeed, the curves converge asymptotically towards 80% for the success rate and slight below 80% for the success rate without forces (fig. 5.4). Offline QT-Opt, however, seems to outperform both offline implementations.

DDQL tested on the semi-sphere achieves a maximum of 90% for the success rate and 85% for the success rate without forces. Online QT-Opt on the same test performs slightly under with a maximum value of 88% for the success rate and 82% for the success rate without forces. On the other hand, offline QT-Opt performs better than the two others with a maximum success rate of 93%. Regarding the success rate without forces, offline QT-Opt achieves 88% while its online version achieves 82%. All curves are stable throughout learning, with offline QT-Opt ahead with a coefficient of variation under 10.

The tests using the semi-sphere show that the AUC of DDQL for the success rate and the success rate without forces achieves 79% and 72%, respectively, while online QT-Opt achieves an AUC of 76% for the success rate and 70% for the success rate without forces. Finally, the AUC for offline QT-Opt tested with the semi-sphere outperforms the others with a value of 80% for the success rate and 73% for the success rate without forces (table 5.5).



(a) Success rate

(b) Success rate without forces

Fig. 5.4 Success rate with and without forces measured against agents trained with DDQL (blue lines), offline QT-Opt (red lines), and online QT-Opt (green lines) on the semi-sphere object evolving over the number of training iterations.

Table 5.5 Results (in %) of agents trained on the semi-sphere

	Success rate			**Success rate w\forces**		
Agent	**Max**	**AUC**	**CV**	**Max**	**AUC**	**CV**
DDQL	89.99	79.04	13.75	85.00	72.36	14.09
Offline QT-Opt	**93.00**	**79.91**	**08.96**	**88.00**	**72.53**	**09.59**
Online QT-Opt	87.99	76.18	11.19	81.60	70.18	11.19

Q-Values and Discounted Rewards

The AUC evolution of the Q-values across timesteps follows the same trend on each dataset tested. Values at the first timestep are relatively low, steadily rises a maximal value around 80% until the fourth timestep, and then progressively returns to its original value until the last timestep (fig. 5.5a, fig. 5.5c, and fig. 5.5e).

DDQL's AUC of the Q-values starts at 40%, while both versions of QT-Opt start with a lower value of 20%. It estimates higher Q-values than offline and online QT-Opt, except for the semi-sphere dataset on which the estimation gap is reduced. Furthermore, after the fourth step, one can contrast the previous observation with the rise in the CV for the Q-values estimation of DDQL on the semi-sphere object, which does not exist in the two other datasets (fig. 5.6e).

Once again, the AUC evolution of the discounted rewards across timestep follows the same trend across training algorithms and datasets (fig. 5.5b, fig. 5.5d, and fig. 5.5f). The AUC of the discounted rewards at the first timestep are around 50%, linearly increasing until the fourth to around 80%, then diminishing below 40% at the tenth timestep, with an elbow appearing at the sixth timestep.

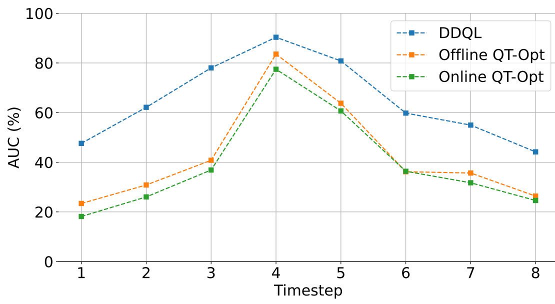
The evolution of the CV for the Q-values and the discounted rewards is very similar across datasets and training algorithm (fig. 5.6).

The CV measured on the Q-values demonstrate the same pattern throughout training algorithms and training datasets. DDQL's CV on the Q-values always starts lower than both offline and online QT-Opt training algorithms. On the other hand, those last two always demonstrate a CV on the Q-values around 30%. All three training algorithms have a CV on the Q-values that decreases from the first timestep to the fourth, even though it is less evident on the airplane training dataset (fig. 5.6c). Once the fourth step is achieved, the CV across all three datasets increases. It is interesting to note that DDQL has a lower CV on the Q-values except for the semi-sphere object (fig. 5.6e).

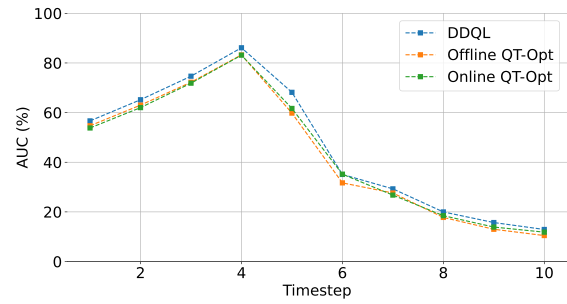
Regarding the CV of the discounted rewards, they all show a plateau from the first to the fourth timestep with a value between 20% and 40%. Once the fourth step is achieved, the curves start to rise with different behaviours (fig. 5.6b, fig. 5.6d, and fig. 5.6f).

The algorithms demonstrate a steady evolution for the CV of discounted rewards on the global dataset with DDQL achieving a score of 160% while both QT-Opt versions are around 100% (fig. 5.6b). Regarding the airplane object, both DDQL and online QT-Opt have their CV on the discounted rewards steadily rising to 100% while offline QT-Opt peaks to 250% (fig. 5.6d). The semi-sphere object presents a high difference between the evolutions of the CV on the discounted

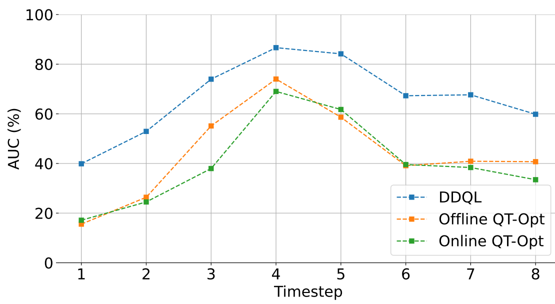
rewards between algorithms (fig. 5.6f). Indeed, DDQL achieves an astonishing score of 3500%, diminishes to 500% only to peak again at 1750% and finally decrease to 500%. Offline QT-Opt also demonstrates a double peak curve, the first peak being at the seventh timestep with a score of 1000% and the second peak located two steps later at around 1100%.



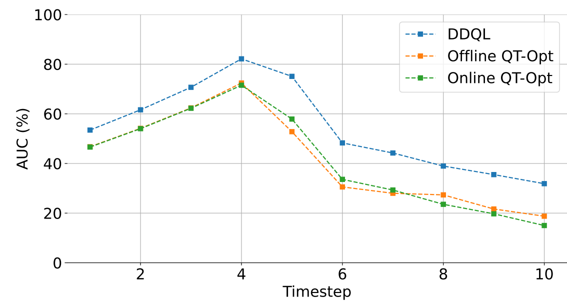
(a) Q-values evolution across timesteps on the global dataset



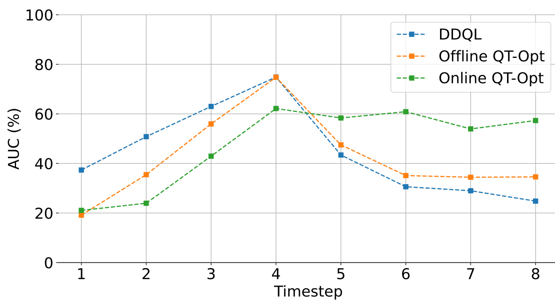
(b) Discounted rewards evolution across timesteps on the global dataset



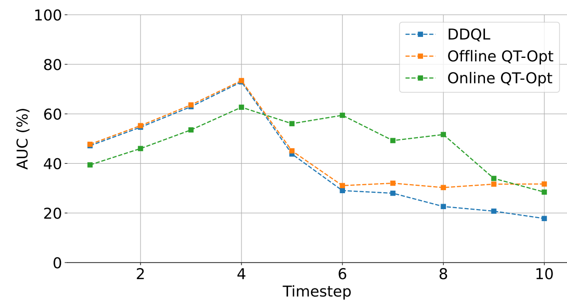
(c) Q-values evolution across timesteps on the airplane object



(d) Discounted rewards evolution across timesteps on the airplane object

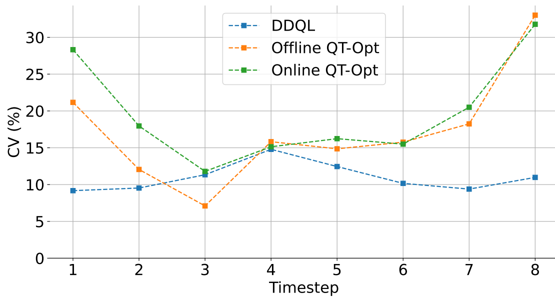


(e) Q-values evolution across timesteps on the semi-sphere object

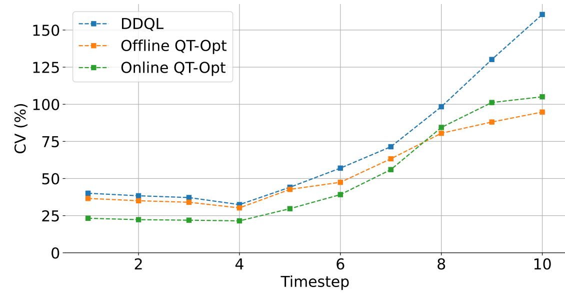


(f) Discounted rewards evolution across timesteps on the semi-sphere object

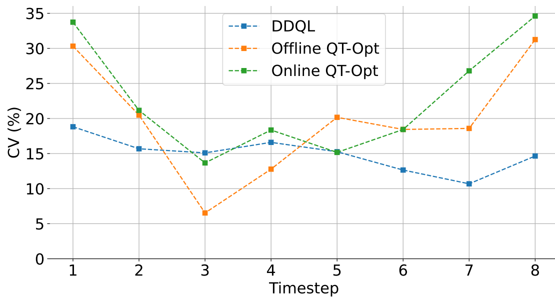
Fig. 5.5 Evolution of the Q-values and the discounted rewards measured with the area under the curve across timesteps for DDQL (blue lines), offline QT-Opt (red lines), and online QT-Opt (green lines) against the three training datasets.



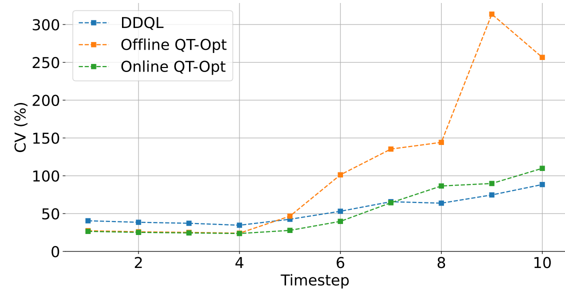
(a) Q-values evolution across timesteps on the global dataset



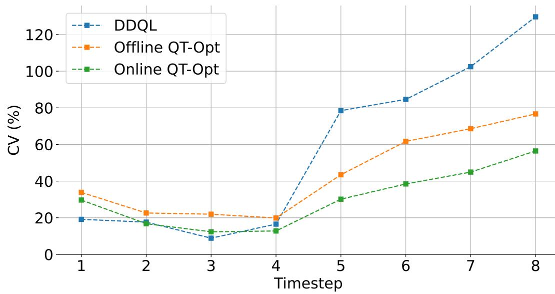
(b) Discounted rewards evolution across timesteps on the global dataset



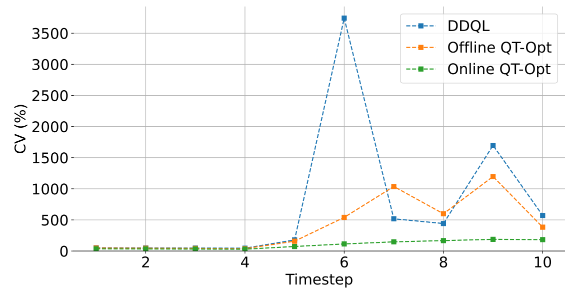
(c) Q-values evolution across timesteps on the airplane object



(d) Discounted rewards evolution across timesteps on the airplane object



(e) Q-values evolution across timesteps on the semi-sphere object



(f) Discounted rewards evolution across timesteps on the semi-sphere object

Fig. 5.6 Evolution of the Q-values and the discounted rewards measured with the coefficient of variation across timesteps for DDQL (blue lines), offline QT-Opt (red lines), and online QT-Opt (green lines) against the three training datasets.

5.3 Discussion

The analysis of the Coefficient of Variation (CV) and Area Under the Curve (AUC) of the discounted rewards and estimated Q-values across timesteps demonstrate that agents are less able to grasp the object after the fourth timestep successfully. Indeed, the AUC of the Q-values and discounted rewards systematically increases until this step. They achieve their maximum on this fourth step and then decline. With the same idea, the CV evolution of the Q-values decreases until the fourth step, demonstrating an increased consistency in estimating the Q-values until this step, and then rises. It shows that agents believe they are about to perform a successful grasp at the fourth timestep. If it does not happen, their confidence in successfully terminating the episode before timestep $T = 10$ declines. The evaluation of the mean number of steps performed reinforces this idea (fig. A.1f, fig. A.4f, fig. A.7f).

Interestingly, online QT-Opt follows the same trend as its two competitors on the AUC of the Q-values and discounted rewards across timesteps. It does not demonstrate a lower decrease rate after the fourth timestep. On the contrary, it even lacks behind the other. One could have searched for a slower slope, resulting from the capacity of online learning to learn from the errors of its policy. This capacity could have, for example, meant that online QT-Opt is more able to learn how to reposition itself in case the first grasp trial fails, as does the original QT-Opt (Kalashnikov et al., 2018).

Comparing the AUC between the Q-values and the discounted rewards across timesteps allows for discussing the overestimation bias in the original Q-Net paper (Mnih et al., 2013). Indeed, one observes through this metric that the difference between the Q-values and discounted rewards is not subject to this bias. This observation is valid for all algorithms. Furthermore, one could postulate that QT-Opt is subject to an underestimation bias, at least until the fourth step. This underestimation biased is believed to be a consequence of the clipped double Q-learning that QT-Opt implements and that DDQL does not. Indeed, the AUC metric shows that the Q-values from QT-Opt consistently underestimate the discounted rewards before the fourth step. On the other hand, the AUC of the Q-values from the three algorithms shows that the Q-network slightly overestimate the actual discounted rewards received from the environment.

Finally, evaluating the success rates with and without forces across all algorithms and datasets does not help to conclude that QT-Opt is a better training algorithm than DDQL for the problem and setup of this thesis. Indeed, even though DDQL does not perform better than the others

according to all metrics, it is globally better. The evaluation of the AUC and CV on the estimated Q-values and discounted returns also demonstrate that DDQL is more consistent than both QT-Opt implementations. The few metrics showing QT-Opt performing better than DDQL appears only on the airplane object or the semi-sphere. Since it does not appear on the global set of transitions, those results are not admissible as a conclusion because they do not generalise enough. Furthermore, the AUC evaluated on both success rate versions demonstrates that DDQL learns with fewer training iterations than its competitors. Since it does not require two target networks to evaluate the target Q-values and train itself, the overall system is faster to train, given the same hardware.

Even though this master's thesis could not provide agents that outperform the original work from IntegrIA, the best agent was able to achieve a success rate of 96.67%, reproducing the 96% from the original QT-Opt paper (Kalashnikov et al., 2018), and demonstrate an improved stability over DDQL (table 5.3). It is interesting since the QT-Opt implementation from this work has some differences compared to the original paper. First, Kalashnikov et al. evaluate their network through the cross-entropy loss, whereas this thesis does it with the Huber loss, selected empirically by the IntegrIA team for its stability. Second, the original QT-Opt system perceives its environment from an over-the-shoulder RGB camera, while this thesis system captures it through a wrist-mounted depth camera. On top of that, Kalashnikov et al. developed an entire distributed reinforcement learning framework to scale up their system across thousands of asynchronous workers. Even if the RLlib library (Liang et al., 2017) and multiple CPU cores power this thesis, it does not have the same ability to scale up the work. Finally, QT-Opt do provide the number of episodes collected and the robot-hour time required. However, it would be interesting to disclose the number of training iterations, the duration of the training, or an equivalent metric (such as the normalised area under the curve over the success rate) to compare the system with other implementations, abstracting the hardware.

Chapter 6

Conclusion

All the information presented in this thesis aims to compare IntegrIA's implementation of the DDQL training algorithm with both online and offline implementations of the QT-Opt training algorithm, which are the thesis main contributions. The IntegrIA team provided a dataset containing 1800 training objects and 588 evaluation ones. Two datasets containing a single object were defined from the main one to test the capacity of the training algorithms on complex objects. Based on multiple qualitative and quantitative analyses, it is concluded that agents resulting from the QT-Opt training algorithm do not improve performance over the DDQL trained ones.

This conclusion is a result of evaluating agents across multiple metrics. Some metrics, such as the success rate and success rate without forces, evaluate the quality of agents on the task of robotic grasping. Other metrics, such as the normalised area under the curve and the coefficient of variation, measured on the discounted rewards and the estimated Q-values allow to compare the evolution of agents internal beliefs across episode timesteps.

The success rate show that agents trained with the DDQL training algorithm on the global dataset achieved a maximum of 97.3% on previously unseen objects, while offline and online QT-Opt trained agents achieve a maximum of 95.33% and 96.67%, respectively. Concerning the success rate without forces, DDQL trained agents performed better than QT-Opt ones, with a maximum of 92.04% for DDQL while offline and online QT-Opt achieved a maximum of 88.71% and 91.76%, respectively.

Regarding the estimated Q-values and discounted rewards, the area under the curve and the coefficient of variation demonstrated that DDQL trained agents have a better understanding of their environments and are more confident about their actions than QT-Opt trained ones.

This thesis answered the main question: '*Does agents trained with the QT-Opt algorithm demonstrate better performance over agents trained with the DDQL training algorithm when evaluated against multiple metrics ?*'.

While answering to the main research question, this master's thesis trained agents with the QT-Opt training algorithm on the global dataset, which achieved a success rate of 96.67% on previously unseen objects, reproducing the 96% from the original paper. It is very promising regarding the potential of simulation-based learning systems. Indeed, these results were obtained using only a tiny proportion of the resources made available to the researchers focused on robotic grasping from Berkley and Google. As an example, the network from the original QT-Opt paper is composed of over 1.2 million parameters, while the implementation from this master's thesis operates with only 716k parameters.

6.1 Further work

The high-dimensionality of the environment causes the training of agents to be challenging for vision-based robotic grasping (Yang et al., 2018). This challenge has been present throughout the entirety of this thesis. It is excessively present when training agents with online algorithms and is probably why online agents are globally less performant compared to their offline equivalent during this thesis. Future directions should then aim at better exploring the environment, which could be achieved through distributional reinforcement learning (Mavrin et al., 2019).

Furthermore, distributional reinforcement learning appears to have the capacity of risk assessing future actions, which, in the context of this thesis, could prevent the application of excessive forces (Ma et al., 2021).

Finally, the recommendation I make to readers is to investigate Q2-Opt (Bodnar et al., 2020), the development of the distributional reinforcement learning equivalent of QT-Opt, realised by the same authors.

References

Unknown author at Intel. Industrial robotic arms: Changing how work gets done, 2020. URL https://www.intel.com/content/www/us/en/robotics/robotic-arm.html#:~:text=Robotic%20arms%20can%20be%20used,a%20risk%20of%20bodily%20injury.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016. URL https://arxiv.org/abs/1607.06450.

Seunghyeok Back, Jongwon Kim, Raeyoung Kang, Seungjun Choi, and Kyoobin Lee. Segmenting unseen industrial components in a heavy clutter using rgb-d fusion and synthetic data. In *2020 IEEE International Conference on Image Processing (ICIP)*, pages 828–832. IEEE, 2020.

Richard Bellman. *Dynamic Programming*. Dover Publications, 1957. ISBN 9780486428093.

Lars Berscheid, Pascal Meißner, and Torsten Kröger. Robot learning of shifting objects for grasping in cluttered environments. *CoRR*, abs/1907.11035, 2019. URL http://arxiv.org/abs/1907.11035.

Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control*, volume I. Athena Scientific, Belmont, MA, USA, 3rd edition, 2005.

Binil Starly, Akshay Bharadwaj, and Atin Angrish. Fabwave cad repository categorized part classes - cad 1 through 15 classes (part 1/3), 2019. URL http://rgdoi.net/10.13140/RG.2.2.31167.87201.

Cristian Bodnar, Adrian Li, Karol Hausman, Peter Pastor, and Mrinal Kalakrishnan. Quantile qt-opt for risk-aware vision-based robotic grasping. *ArXiv*, abs/1910.02787, 2020.

Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. http://pybullet.org, 2016–2021.

J. D. A. Cuddy and P. A.Della Valle. Measuring the instability of time series data. *Oxford Bulletin of Economics and Statistics*, 40(1):79–85, 1978. doi: https://doi.org/10.1111/j.1468-0084.1978.mp40001006.x. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1468-0084.1978.mp40001006.x.

Guoguang Du, Kai Wang, and Shiguo Lian. Vision-based robotic grasping from object localization, pose estimation, grasp detection to motion planning: A review. *CoRR*, abs/1905.06658, 2019. URL http://arxiv.org/abs/1905.06658.

Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. *CoRR*, abs/1802.09477, 2018. URL http://arxiv.org/abs/1802.09477.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR. URL https://proceedings.mlr.press/v9/glorot10a.html.

Richard H. R. Hahnloser, Rahul Sarpeshkar, Misha A. Mahowald, Rodney J. Douglas, and H. Sebastian Seung. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature*, 405(6789):947–951, Jun 2000. ISSN 1476-4687. doi: 10.1038/35016072. URL https://doi.org/10.1038/35016072.

Justin Rose Harold L. Sirkin, Micheal Zinser. The shifting economics of global manufacturing. how a takeoff in advanced robotics will power the next productivity surge, February 2015. URL https://www.slideshare.net/TheBostonConsultingGroup/robotics-in-manufacturing.

R. A. Howard. *Dynamic Programming and Markov Processes*. MIT Press, Cambridge, MA, 1960.

John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Zidek, Anna Potapenko, Alex Bridgland, Clemens Meyer, Simon A. A. Kohl, Andrew J. Ballard, Andrew Cowie, Bernardino Romera-Paredes, Stanislav Nikolov, Rishub Jain, Jonas Adler, Trevor Back, Stig Petersen, David Reiman, Ellen Clancy, Michal Zielinski, Martin Steinegger, Michalina Pacholska, Tamas Berghammer, Sebastian Bodenstein, David Silver, Oriol Vinyals, Andrew W. Senior, Koray Kavukcuoglu, Pushmeet Kohli, and Demis Hassabis. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, Aug 2021. ISSN 1476-4687. doi: 10.1038/s41586-021-03819-2. URL https://doi.org/10.1038/s41586-021-03819-2.

Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, and Sergey Levine. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *CoRR*, abs/1806.10293, 2018. URL http://arxiv.org/abs/1806.10293.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014. URL https://arxiv.org/abs/1412.6980.

Kilian Kleeberger, Richard Bormann, Werner Kraus, and Marco Huber. A survey on learning-based robotic grasping. *Current Robotics Reports*, 1:239–249, 12 2020. doi: 10.1007/s43154-020-00021-6.

Sebastian Koch, Albert Matveev, Zhongshi Jiang, Francis Williams, Alexey Artemov, Evgeny Burnaev, Marc Alexa, Denis Zorin, and Daniele Panozzo. Abc: A big cad model dataset for geometric deep learning. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. URL https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

Yann Le Cun, Leon Bottou, and Yoshua Bengio. Reading checks with multilayer graph transformer networks. *Proceedings - ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing*, 1:151–154, 1997. ISSN 0736-7791. Proceedings of the 1997 IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP. Part 1 (of 5) ; Conference date: 21-04-1997 Through 24-04-1997.

Sergey Levine, Peter Pastor, Alex Krizhevsky, and Deirdre Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *CoRR*, abs/1603.02199, 2016. URL http://arxiv.org/abs/1603.02199.

Eric Liang, Richard Liaw, Philipp Moritz, Robert Nishihara, Roy Fox, Ken Goldberg, Joseph E. Gonzalez, Michael I. Jordan, and Ion Stoica. Rllib: Abstractions for distributed reinforcement learning, 2017. URL https://arxiv.org/abs/1712.09381.

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In Yoshua Bengio and Yann LeCun, editors, *ICLR*, 2016. URL http://dblp.uni-trier.de/db/conf/iclr/iclr2016.html#LillicrapHPHETS15.

Yecheng Jason Ma, Dinesh Jayaraman, and Osbert Bastani. Conservative offline distributional reinforcement learning. In *NeurIPS*, 2021.

Jeffrey Mahler, Florian T Pokorny, Brian Hou, Melrose Roderick, Michael Laskey, Mathieu Aubry, Kai Kohlhoff, Torsten Kröger, James Kuffner, and Ken Goldberg. Dex-net 1.0: A cloud-based network of 3d objects for robust grasp planning using a multi-armed bandit model with correlated rewards. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1957–1964. IEEE, 2016.

Jeffrey Mahler, Matthew Matl, Vishal Satish, Michael Danielczuk, Bill DeRose, Stephen McKinley, and Ken Goldberg. Learning ambidextrous robot grasping policies. *Science Robotics*, 4 (26):eaau4984, 2019.

Borislav Mavrin, Shangtong Zhang, Hengshuai Yao, Linglong Kong, Kaiwen Wu, and Yaoliang Yu. Distributional reinforcement learning for efficient exploration. *ArXiv*, abs/1905.06125, 2019.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013. URL https://arxiv.org/abs/1312.5602.

D. Morrison, P. Corke, and J. Leitner. Egad! an evolved grasping analysis dataset for diversity and reproducibility in robotic manipulation. https://dougsm.github.io/egad/, 2020.

Kumpati S. Narendra and M. A. L. Thathachar. Learning automata - a survey. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-4(4):323–334, 1974. doi: 10.1109/TSMC.1974.5408453.

B. T. Polyak and A. B. Juditsky. Acceleration of stochastic approximation by averaging. *SIAM Journal on Control and Optimization*, 30(4):838–855, 1992. doi: 10.1137/0330046. URL https://doi.org/10.1137/0330046.

L.S. Pontryagin. *Mathematical Theory of Optimal Processes*. Classics of Soviet Mathematics. Taylor & Francis, 1987. ISBN 9782881240775. URL https://books.google.be/books?id=kwzq0F4cBVAC.

Deirdre Quillen, Eric Jang, Ofir Nachum, Chelsea Finn, Julian Ibarz, and Sergey Levine. Deep reinforcement learning for vision-based robotic grasping: A simulated comparative evaluation of off-policy methods. *CoRR*, abs/1802.10264, 2018. URL http://arxiv.org/abs/1802.10264.

Reuven Y. Rubinstein and Dirk P. Kroese. *The Cross Entropy Method: A Unified Approach To Combinatorial Optimization, Monte-Carlo Simulation (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2004. ISBN 038721240X.

EURESYS S.A. Easy3dobject - 3d object extraction and measurement library. URL https://www.euresys.com/en/Products/Machine-Vision-Software/Open-eVision-Libraries/Easy3DObject.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, jan 2016. ISSN 0028-0836. doi: 10.1038/nature16961.

Richard S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44, Aug 1988. ISSN 1573-0565. doi: 10.1023/A:1022633531479. URL https://doi.org/10.1023/A:1022633531479.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018. URL http://incompleteideas.net/book/the-book-2nd.html.

Gerald Tesauro. Td-gammon, a self-teaching backgammon program, achieves master-level play. *Neural Computation*, 6(2):215–219, 1994. doi: 10.1162/neco.1994.6.2.215.

Edward L. Thorndike. *Animal intelligence : experimental studies*. New York : Macmillan Company, 1911., 1911.

Edward L. Thorndike. The law of effect. *The American Journal of Psychology*, 39(1/4):212–222, 1927. ISSN 00029556. URL http://www.jstor.org/stable/1415413.

Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. *CoRR*, abs/1509.06461, 2015. URL http://arxiv.org/abs/1509.06461.

C. J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, Oxford, 1989.

Eric Wiewiora. *Reward Shaping*, pages 863–865. Springer US, Boston, MA, 2010. ISBN 978-0-387-30164-8. doi: 10.1007/978-0-387-30164-8_731. URL https://doi.org/10.1007/978-0-387-30164-8_731.

Yuxin Wu and Kaiming He. Group normalization, 2018. URL https://arxiv.org/abs/1803.08494.

Kai Yang, Zhanpeng Zhang, Hui Cheng, Huadong Wu, and Ziying Guo. Domain centralization and cross-modal reinforcement learning for vision-based robotic manipulation. 2018.

Andy Zeng, Shuran Song, Stefan Welker, Johnny Lee, Alberto Rodriguez, and Thomas A. Funkhouser. Learning synergies between pushing and grasping with self-supervised deep reinforcement learning. *CoRR*, abs/1803.09956, 2018. URL http://arxiv.org/abs/1803.09956.

Zhilu Zhang and Mert R. Sabuncu. Generalized cross entropy loss for training deep neural networks with noisy labels. *CoRR*, abs/1805.07836, 2018. URL http://arxiv.org/abs/1805.07836.

Appendix A

Additional Results from Experiments

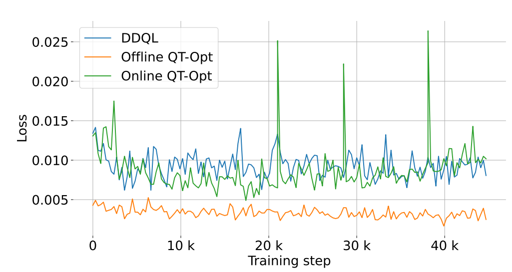
A.1 QT-Opt vs. DDQL

A.1.1 Global Dataset

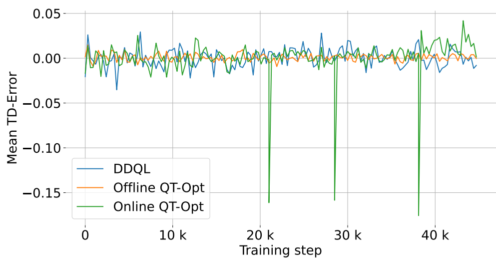
Main Metrics

(a) Success rate

(b) Success rate without forces

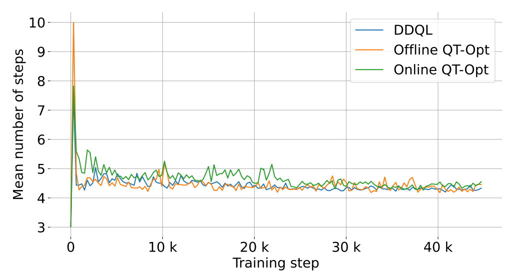


(c) Loss



(d) TD-error

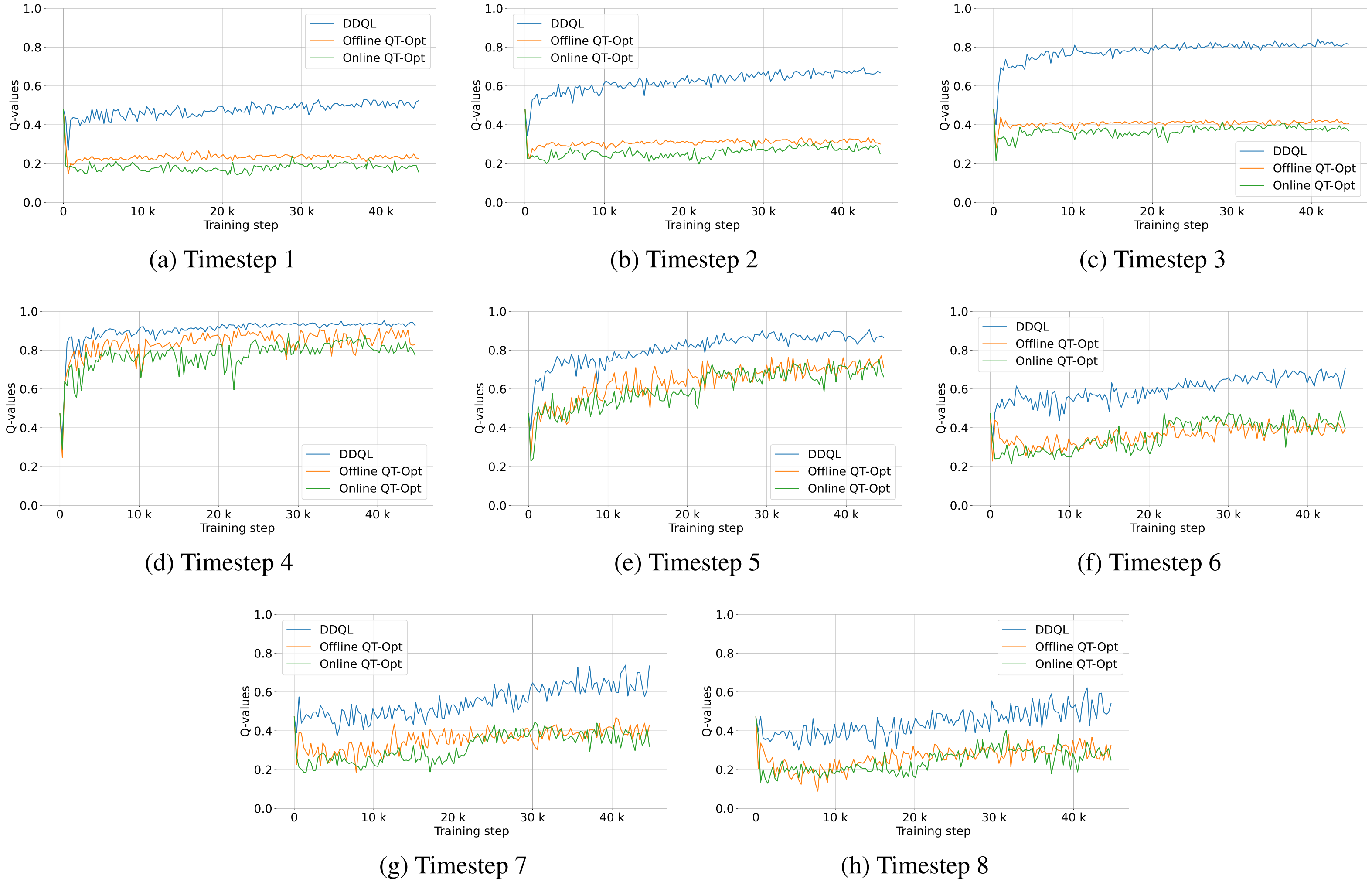
(e) Mean episode reward



(f) Mean number of steps

Fig. A.1 Metrics from DDQL (blue lines), offline QT-Opt (red lines), and online QT-Opt (green lines) trained on the global set of objects against the number of training iterations.

Q-values Evolution



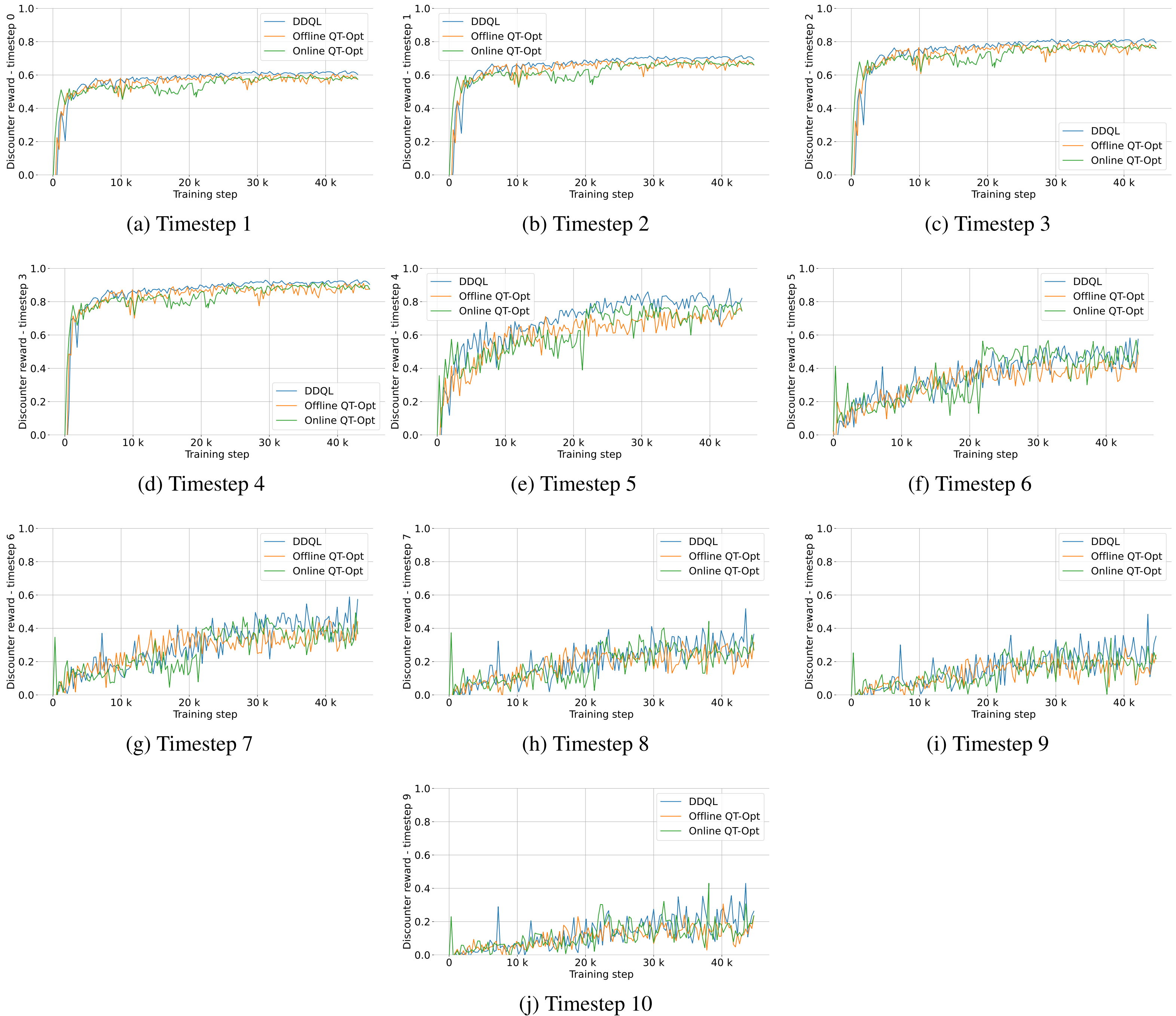
(a) Timestep 1 (b) Timestep 2 (c) Timestep 3

(d) Timestep 4 (e) Timestep 5 (f) Timestep 6

(g) Timestep 7 (h) Timestep 8

Fig. A.2 Mean Q-values from DDQL (blue lines), offline QT-Opt (red lines), and online QT-Opt (green lines) trained on the global set of objects against the number of training iterations for each timestep.

Discounted Rewards Evolution



(a) Timestep 1

(b) Timestep 2

(c) Timestep 3

(d) Timestep 4

(e) Timestep 5

(f) Timestep 6

(g) Timestep 7

(h) Timestep 8

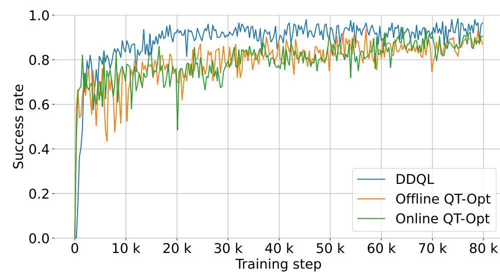
(i) Timestep 9

(j) Timestep 10

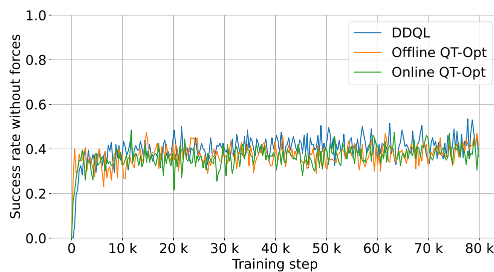
Fig. A.3 Discounted rewards from DDQL (blue lines), offline QT-Opt (red lines), and online QT-Opt (green lines) trained on the global set of objects against the number of training iterations for each timestep.

A.1.2 airplane

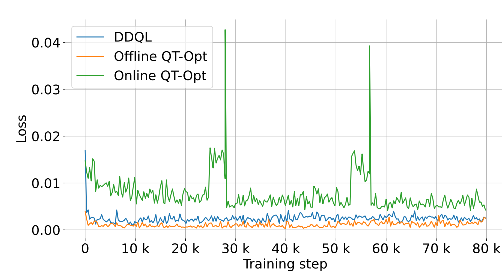
Main Metrics



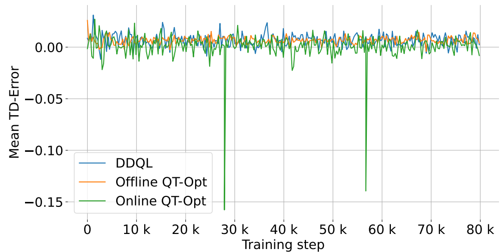
(a) Success rate



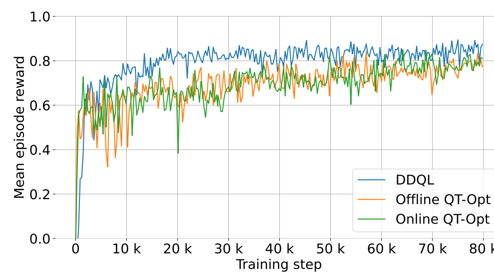
(b) Success rate without forces



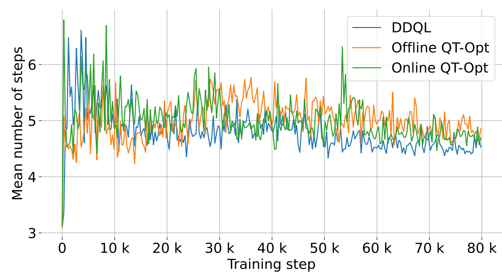
(c) Loss



(d) TD-error



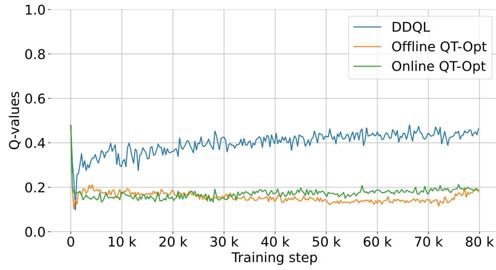
(e) Mean episode reward



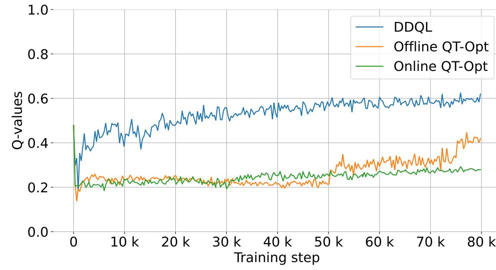
(f) Mean number of steps

Fig. A.4 Metrics from DDQL (blue lines), offline QT-Opt (red lines), and online QT-Opt (green lines) trained on the airplane object against the number of training iterations.

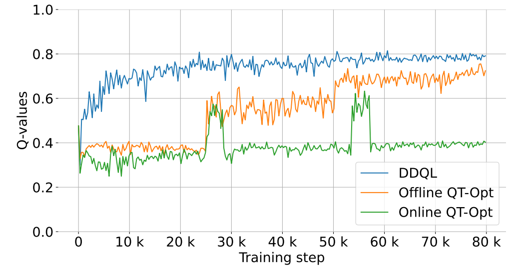
Q-values Evolution



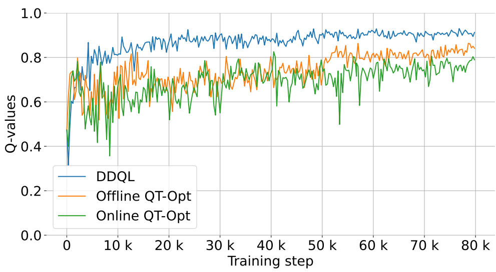
(a) Timestep 1



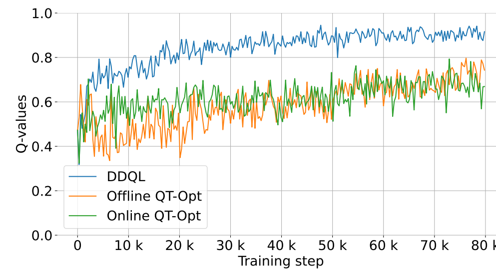
(b) Timestep 2



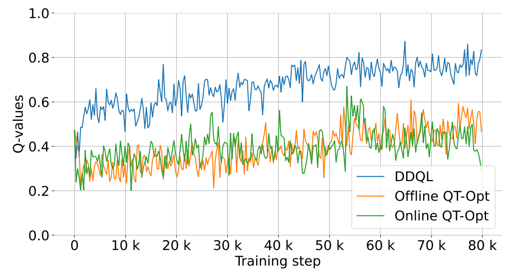
(c) Timestep 3



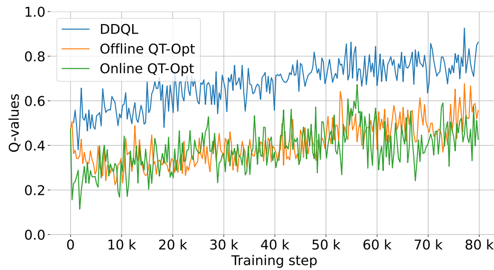
(d) Timestep 4



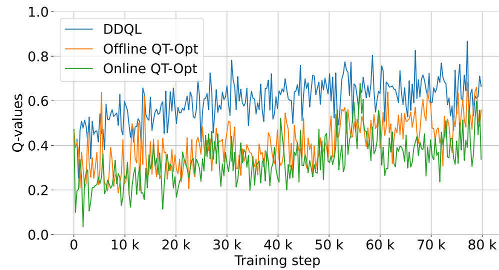
(e) Timestep 5



(f) Timestep 6



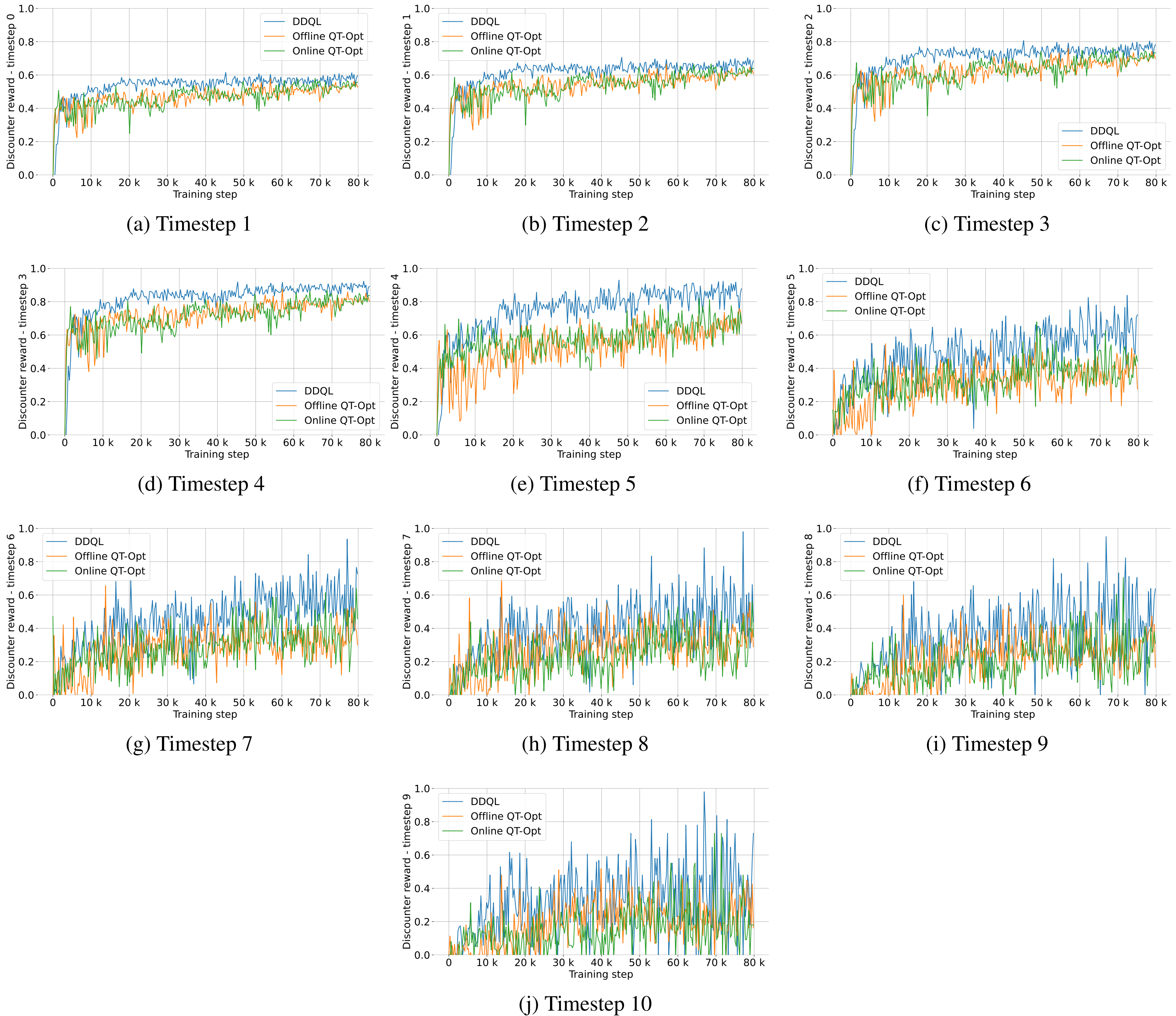
(g) Timestep 7



(h) Timestep 8

Fig. A.5 Mean Q-values from DDQL (blue lines), offline QT-Opt (red lines), and online QT-Opt (green lines) trained on the airplane object against the number of training iterations for each timestep.

Discounted Rewards Evolution



(a) Timestep 1 (b) Timestep 2 (c) Timestep 3

(d) Timestep 4 (e) Timestep 5 (f) Timestep 6

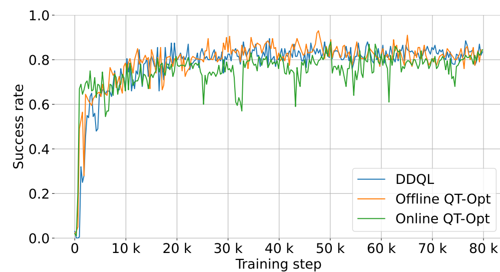
(g) Timestep 7 (h) Timestep 8 (i) Timestep 9

(j) Timestep 10

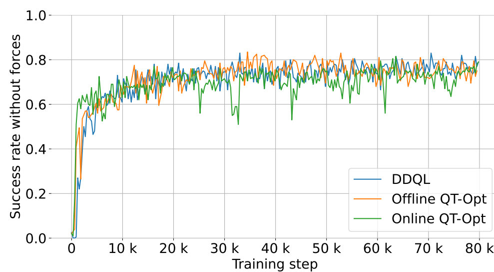
Fig. A.6 Discounted rewards from DDQL (blue lines), offline QT-Opt (red lines), and online QT-Opt (green lines) trained on the airplane object against the number of training iterations for each timestep.

A.1.3 Semi-Sphere

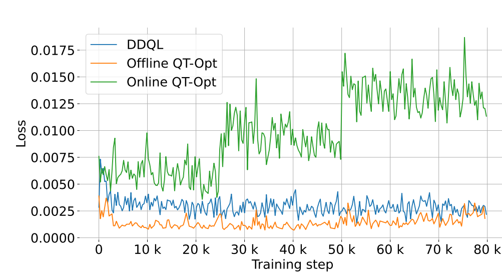
Main Metrics



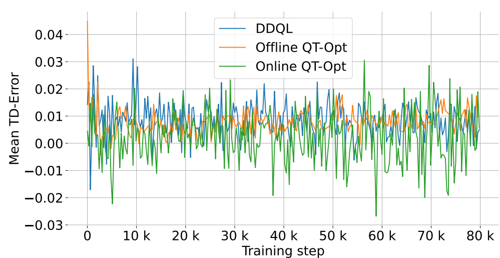
(a) Success rate



(b) Success rate without forces

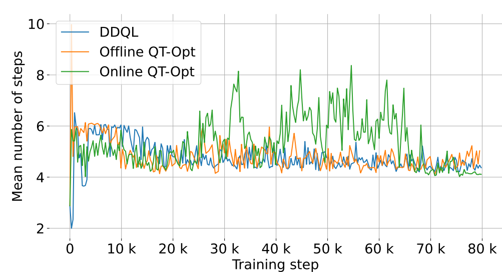


(c) Loss



(d) TD-error

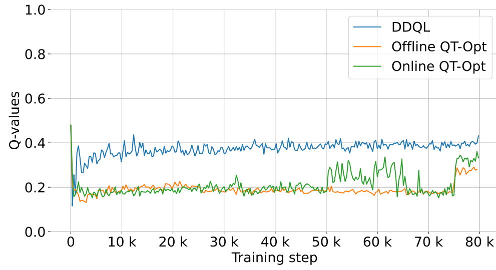
(e) Mean episode reward



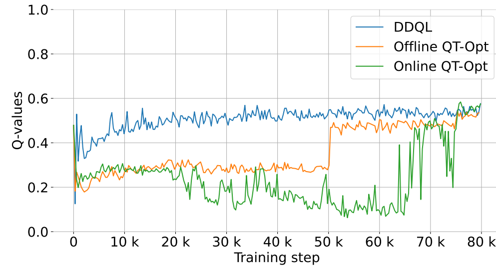
(f) Mean number of steps

Fig. A.7 Metrics from DDQL (blue lines), offline QT-Opt (red lines), and online QT-Opt (green lines) trained on the semi-sphere object against the number of training iterations.

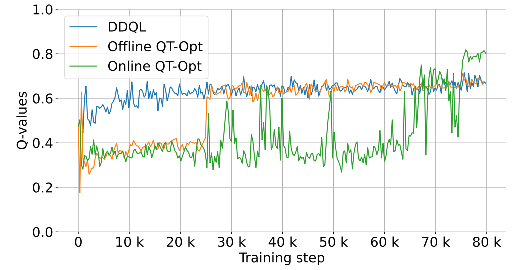
Q-values Evolution



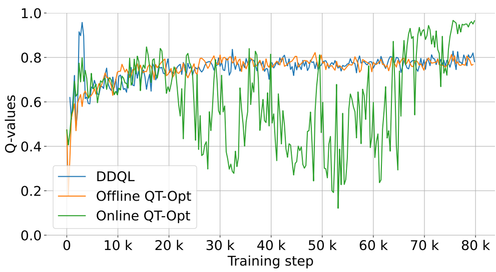
(a) Timestep 1



(b) Timestep 2

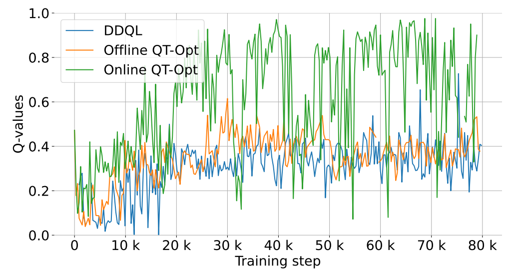


(c) Timestep 3

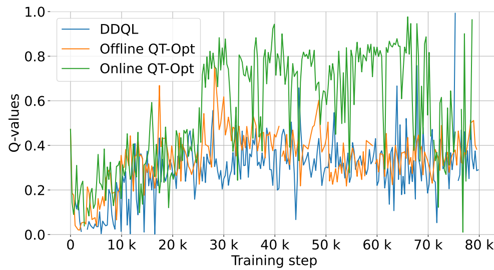


(d) Timestep 4

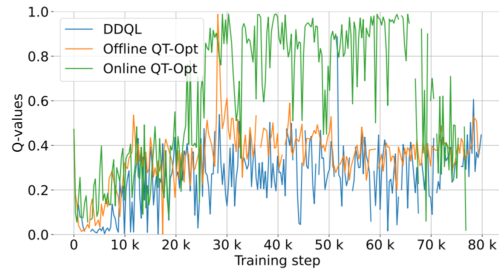
(e) Timestep 5



(f) Timestep 6



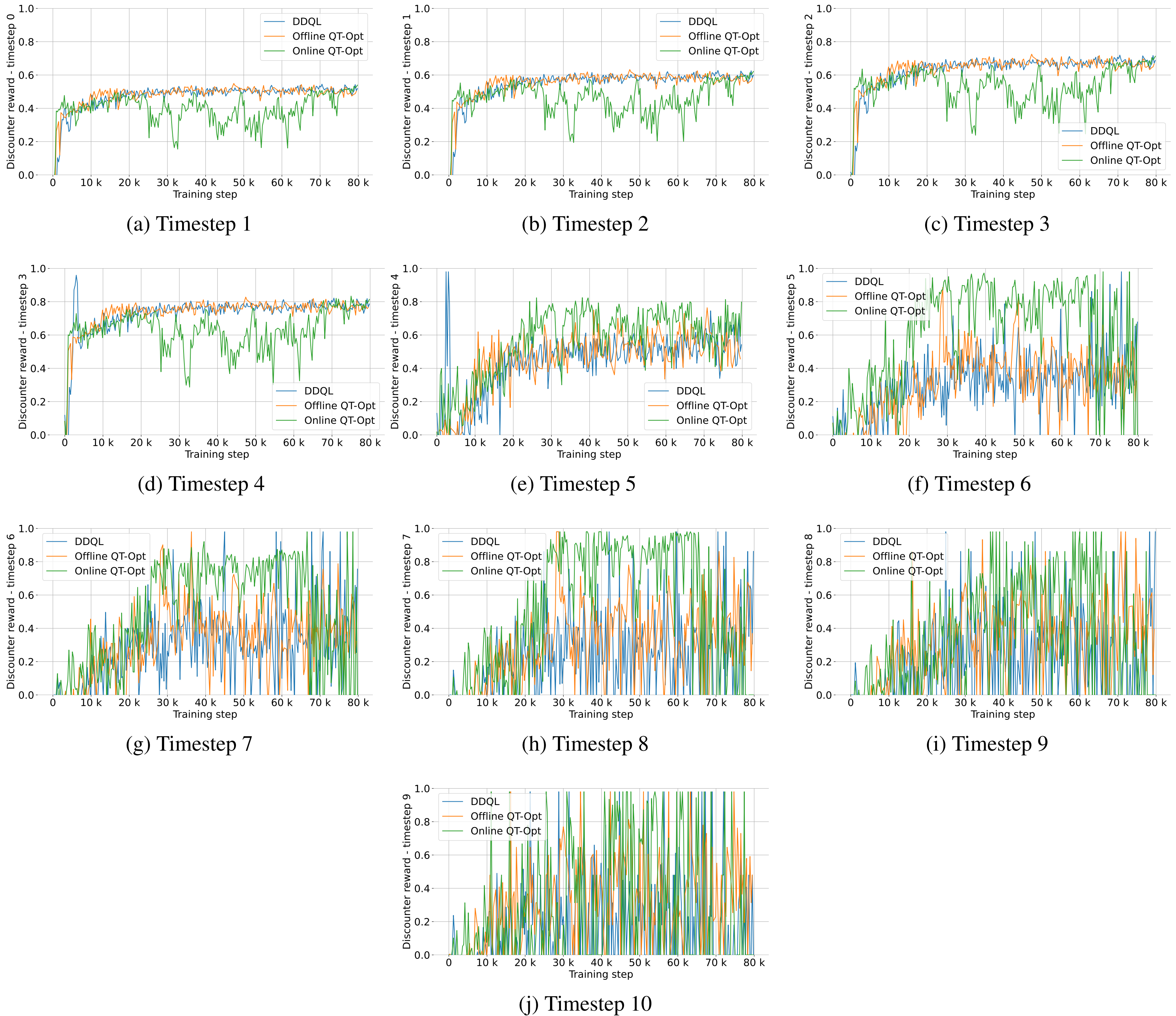
(g) Timestep 7



(h) Timestep 8

Fig. A.8 Mean Q-values from DDQL (blue lines), offline QT-Opt (red lines), and online QT-Opt (green lines) trained on the semi-sphere object against the number of training iterations for each timestep.

Discounted Rewards Evolution



(a) Timestep 1 (b) Timestep 2 (c) Timestep 3

(d) Timestep 4 (e) Timestep 5 (f) Timestep 6

(g) Timestep 7 (h) Timestep 8 (i) Timestep 9

(j) Timestep 10

Fig. A.9 Discounted rewards from DDQL (blue lines), offline QT-Opt (red lines), and online QT-Opt (green lines) trained on the semi-sphere object against the number of training iterations for each timestep.

Appendix B

Detailed Algorithms

B.1 DDQL

Algorithm 1 Offline DDQL

Input: D be the offline experience replay buffer

1: Initialise Q_θ network
2: $k \leftarrow 0$
3: $\theta'_1 \leftarrow \theta$ ▷ Update the target network parameters
4: **for** each training step **do**
5: $(s_i, a_i, r_i, s'_i) \sim D$ ▷ Sample a random minibatch of N samples from D
6: **if** s'_i is terminal **then**
7: $V(s'_i) \leftarrow 0$
8: **else**
9: $V(s'_i) \leftarrow Q_{\theta'}(s'_i, \arg\max_{a' \in A} Q_\theta(s'_i, a'))$ ▷ Compute the value of next state s'_i
10: $\theta \leftarrow \theta - \alpha \sum_i \frac{dQ_\theta}{d\theta}(s_i, a_i)\left[Q_\theta(s_i, a_i) - (r_i + \gamma V(s'_i))\right]$ ▷ Gradient step
11: $k = k + 1$
12: **if** $k = \tau$ **then** ▷ Update the target network parameters
13: $\theta'_2 \leftarrow \theta'_1$
14: $k \leftarrow 0$

The default values for the parameters used for DDQL are:

- Discount factor: $\gamma = 0.9$
- Learning rate: $\alpha = 0.001$

- Target network update frequency: $\tau = 0.95$

B.2 QT-Opt

Algorithm 2 Offline QT-Opt

Input: D be the offline experience replay buffer

1: Initialise Q_θ network
2: Initialise $Q_{\theta'_1}$ network
3: $k \leftarrow 0$
4: $\theta'_2 \leftarrow \theta'_1$
5: **for** each training step **do**
6: $(s_i, a_i, r_i, s'_i) \sim D$ ▷ Sample a random minibatch of N samples from D
7: **if** s'_i is terminal **then**
8: $V_{\theta'_1,\theta'_2}(s'_i) \leftarrow 0$
9: **else** ▷ Polyak-averaging and clipped-double q-learning
10: $V_{\theta'_1,\theta'_2}(s'_i) \leftarrow \min_{i=1,2} Q_{\theta'_i}(s'_i, \arg\max_{a' \in A} Q_{\theta'_1}(s'_i, a'))$
11:
12: $\theta \leftarrow \theta - \alpha \sum_i \frac{dQ_\theta}{d\theta}(s_i, a_i)\,[Q_\theta(s_i, a_i) - (r_i + \gamma V(s'_i))]$ ▷ Gradient step
13: $\theta'_1 \leftarrow \tau_1 \times \theta'_1 + (1 - \tau_1) \times \theta$ ▷ Polyak-averaging of the first target network
14: $k = k + 1$
15: **if** $k = \tau_2$ **then** ▷ Update the second target network parameters
16: $\theta'_2 \leftarrow \theta'_1$
17: $k \leftarrow 0$

The default values for the parameters used for QT-Opt are:

- Discount factor: $\gamma = 0.9$
- Learning rate: $\alpha = 0.001$
- Exponential moving average factor: $\tau_1 = 0.95$
- Lagged target network update: $\tau_2 = 50$[1]

[1] The update frequency is proportional to the batch size according to $(\tau_2 - 1) \times$ batch_size. If the goal is to update the lagged target network every 50 gradient steps, with a batch size of 512, then in practice $\tau_2 = (50 - 1) \times 512 = 25,088$.

B.3 Cross-Entropy Method

Algorithm 3 Cross-Entropy Method (CEM)

Input: $O(s,\cdot)$ ▷ Objective function with state s to maximise

1: $\mu \leftarrow 0$
2: $\sigma \leftarrow 1$
3: **for** $k = 1$ **to** K **do**
4: $S_e \leftarrow \emptyset$
5: $\mathbf{a} \sim N(\mu, \sigma)$ ▷ Sample a minibatch **a** of n actions a_i from the Gaussian $N(\mu, \sigma)$
6: $\mathbf{a}^* \leftarrow \mathbf{a}.\text{SORT}(a_i \in \mathbf{a} : O(s, a_i))$ ▷ Rank the actions according to the objective function $O(s,\cdot)$
7: $S_e \leftarrow \mathbf{a}^*.\text{GET}(n_e)$ ▷ Get the n_e best actions
8: $\mu \leftarrow \text{MEAN}(S_e)$ ▷ Compute the mean of the best actions
9: $\sigma \leftarrow \text{STD}(S_e)$ ▷ Compute the standard deviation of the best actions

The default values for the parameters of the CEM are:

- Number of CEM iterations: $K = 3$
- Number of samples per iteration: $n = 64$
- Number of elites selected per iteration: $n_e = 10$