Towards Optimal Grasping Of Unknown Objects Using Deep Reinforcement Learning

BEHRAD JABARNEJAD

A THESIS SUBMITTED TO THE FACULTY OF GRADUATE STUDIES IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF APPLIED SCIENCES

GRADUATE PROGRAM IN MECHANICAL ENGINEERING
YORK UNIVERSITY
TORONTO, ONTARIO

AUGUST 2024

Abstract

Smart manufacturing, which uses advanced technologies to optimize processes, has become increasingly popular. Improving the agility of robotic systems, including manipulation skills, is crucial in this field. Grasp synthesis, the process of developing grasping plans while manipulating objects, can be approached empirically. Deep reinforcement learning (DRL) is an empirical method that does not require a dataset and learns tasks by interacting with an environment. This study aims to utilize DRL algorithms to perform grasping by creating a novel grasping environment for a simulation model of a three-finger gripper and then validating this model to achieve optimal grasp on unknown objects. The results showed that the trained DRL model in the validated simulation environment successfully grasped unknown objects placed randomly. The agent identified optimal grasps using a grasp quality score in the DRL model's reward function.

Acknowledgements

Thanks, Mom and Dad

Table of Contents

List of Tables

List of Figures

Acronyms

2D	Two-Dimensional.
3D	Three-Dimensional.
API	Application programming interface.
DDPG	Deep Deterministic Policy Gradient.
DRL	Deep Reinforcement Learning.
IK	Inverse kinematics.
IoT	Internet of Things.
PTP	Point-to-Point.
RGB	Red, Green and Blue.
RL	Reinforcement Learning.
RMSE	Root Mean Squared Error.
SAC	Soft Actor-Critic.
SB3	Stable-Baselines3.
TD3	Twin Delayed DDPG.

Chapter 1

Introduction

The recent surge in popularity surrounding the term "smart manufacturing" underscores its designation as the future of production. This has consequently led to a significant rise in publications comprehensively exploring the multifaceted aspects of smart manufacturing[1]. As defined by Chou[2], smart manufacturing encompasses "the use of advanced technologies, systems, and real-time data to optimize the manufacturing process, from design and engineering to physical production, resulting in improved efficiency, productivity, quality, and safety." Manufacturers stand to gain a multitude of benefits by embracing smart manufacturing. These include enhanced efficiency, quality control, and flexibility, alongside cost reduction and improved safety. Advanced technologies such as robotics, artificial intelligence, and the IoT empower manufacturers to automate tasks, monitor production processes in real-time, and adapt swiftly to market fluctuations. Furthermore, smart manufacturing offers the potential to reduce labor costs, improve inventory management, minimize waste generation, and foster a safer work environment[3]. A prime example is NGen®, a leading Canadian smart manufacturing company. NGen®received a $177 million investment from the Government of Canada[4] to capitalize on Canada's strengths in research, technology, and manufacturing for further development.

The National Institute of Standards and Technology (NIST)[2], has identified the enhancement of robot systems for smart manufacturing as a critical project, with a particular emphasis on improving their agility. Object manipulation, a fundamental task for industrial robots, can be segmented into two distinct phases:

- Reaching phase: The goal of this phase is to get close to the target object in a safe procedure.
- Capturing phase: The goal of this phase is to gain full control of the object or so-called grasp it.

The reaching phase is achieved through trajectory planning, a process that defines a collision-free path for the robot to follow. Trajectory planning algorithms consider various application-specific criteria, including collision avoidance, minimal time or energy consumption, maximization of speed, and precision. A significant body of research has been devoted to trajectory planning, yielding promising results. For instance, novel algorithms have been developed to optimize trajectory planning by increasing velocity while minimizing torque consumption[5]. However, despite the advancements in trajectory planning, grasping remains a challenging task in industrial robotics.

While grasping objects appears to be a relatively straightforward task for humans, it presents significant challenges for robots. Even though numerous findings with promising results exists in robotic grasping, this task is still a tough challenge[6]. This chapter will delve into the complexities involved in robotic grasping, focusing on grasp planning, also known as grasp synthesis. Grasp synthesis refers to the process of selecting contact points on an object for the gripper or defining the relative position and orientation between the gripper and the object. Broadly, there are two primary approaches to grasp synthesis: analytical methods and empirical methods.

Analytical methods rely on solving a set of physical equations to identify optimal candidate grasping points. While this approach offers an accurate and reliable plan, it necessitates a comprehensive and precise 3D model of the object. This requirement becomes impractical in contemporary scenarios where robots are expected to exhibit intelligence and handle novel or diverse objects.

In contrast, empirical methods leverage learning approaches to grasp objects. Instead of solving complex mathematical equations, the robot learns the grasping task itself. It has already been proved that empirical methods can learn effective policies for grasp planning[7]. The learning process can be either supervised or unsupervised and may incorporate human demonstrations. Deep Reinforcement Learning (DRL) algorithms represent the most recent advancement in this domain. Notably, DRL eliminates the requirement for a precise and complete object model. Additionally, DRL algorithms can be trained within a simulation environment, circumventing the need for large and expensive datasets.

While a significant portion of research dedicated to developing grasp synthesis has focused on employing DRL with rudimentary grippers, no prior studies have explored the application of this approach with more advanced grippers and subsequent evaluation of the resulting grasps.

1.1 Statement of Problem

Robotic manipulation hinges on a critical aspect known as grasping, which denotes achieving complete control over an object's movement. Grasping refers to the action of securing an object in a stationary position by applying forces and torques at designated contact points. As highlighted in prior research[8], grasping presents numerous challenges, which will be elaborated upon in the subsequent paragraphs.

The inherent physics of grasping presents a significant challenge for robots. Unlike humans who can effortlessly adjust their approach to grasp an object from a more convenient orientation, robots are often limited by the initial pose of the target object. Even seemingly simple tasks for humans, such as grasping an upside-down bowl, become intricate challenges for robots due to the difficulty in applying appropriate grasping forces and maintaining stability. Furthermore, unintended collisions during grasping attempts can significantly hinder success. These collisions can arise from imprecise motion planning or external factors in the environment, and can cause the robot to drop the object or damage itself or its surroundings.

The second challenge lies in the robot's imperfect perception of the environment. A critical component of successful grasping is accurate perception of the object's shape, size, and pose. Primarily, robots rely on sensor data, often captured through cameras, to build this perception. However, sensor data is inherently susceptible to noise, which can distort the robot's understanding of the object's characteristics. Even minor variations in lighting or sensor malfunctions can introduce noise, leading the robot to misjudge the object's size or shape and potentially compromising the grasp. Additionally, occlusion significantly impedes grasping success. The desired object might be obscured by other objects in the environment, the robot's own arms, or even its gripper itself. This occlusion can prevent the robot from accurately identifying the object's grasping points or even lead it to completely miss the object altogether. Moreover, in cluttered environments with multiple objects in close proximity, the robot's perception system might erroneously identify several objects as a single entity. This misperception can lead to the robot attempting to grasp an unintended object or struggling to grasp the desired object due to the presence of its neighbors.

Another significant challenge in robotic grasping stems from the complexity of modern grippers. Unlike simple grippers with limited degrees of freedom, advanced grippers, such as the 5-fingered gripper designed by SCHUNK®with 20 degrees of freedom, offer a high degree of dexterity. However, this very complexity introduces challenges in control. In an ideal scenario, precise control of all the motors within the gripper, governing both position and torque, would be necessary for successful grasping. However, achieving such fine-tuned, real-time control becomes increasingly difficult with an increasing number of degrees of freedom. The computational complexity of coordinating these movements and the potential

for actuator limitations can significantly hinder grasping performance.

Choosing the optimal contact points for a robust grasp presents another significant hurdle. Several factors restrict the selection of ideal contact points: limitations in the gripper's design (e.g., not all gripper configurations can access every point on an object), the robot's workspace constraints (e.g., limitations in the robot's reach or arm positions), and the pose (position and orientation) of the target object itself. Furthermore, for a given object, there may exist multiple sets of contact points that could achieve a successful grasp. However, identifying the most suitable set amongst these possibilities adds another layer of complexity to the grasping task. Selection criteria often involve considerations such as grasp stability (minimizing the risk of the object slipping), object damage prevention (avoiding contact with delicate parts), and efficiency (minimizing the number of contact points required).

1.2 Background and Technical Gaps

To effectively address the aforementioned challenges in robotic grasping, a systematic decomposition of the grasping process is essential. This decomposition typically involves three distinct stages: planning (synthesis), acquisition, and evaluation. While motion planning, a crucial aspect of the planning stage, falls outside the scope of this thesis, it's important to acknowledge its role in coordinating the robot's manipulator to achieve the desired grasping pose. Grasping synthesis, the primary focus of this study, refers to the process of generating effective grasping strategies or plans for robots to manipulate objects. This stage encompasses determining the optimal configuration of the robot's gripper or end-effector, along with the necessary forces and torques, to establish an optimal grasp on the target object.

There are two fundamental approaches to grasping synthesis: analytical and empirical. Analytical methods leverage mathematical models and geometric reasoning to predict the most suitable grasping configuration and forces for a given object. This approach offers a precise mathematical framework for grasping, enabling the synthesis of optimal grasps. However, a significant limitation of analytical methods lies in their limited applicability to novel objects. Additionally, these methods often necessitate reprogramming in response to environmental changes or encounters with new objects. In contrast, empirical methods address these limitations by employing machine learning and data-driven approaches. Here, grasping policies are learned from examples or through trial-and-error processes via a "smart agent." Advancements in artificial intelligence have propelled empirical methods to achieve promising results. However, a major challenge associated with current empirical methods is their dependence on large datasets and extensive training times, which can be impractical for real-world applications. Reinforcement learning offers a potential solution, enabling

robots to learn and perform tasks without the need for massive datasets.

Reinforcement Learning (RL) presents a promising approach to mitigate the challenges associated with large datasets in robot grasping. Within the RL framework, an agent interacts with its environment by taking actions and observing the resulting outcomes. The agent's objective is to maximize a pre-defined reward signal that reflects the success of its actions. RL offers a significant advantage: the ability to train the agent within a simulated environment, facilitating the exploration of various grasping strategies without the risks associated with real-world experimentation. Once trained, the learned grasping policies can then be transferred to the actual robot for real-world deployment.

The effectiveness and accuracy of RL algorithms can be significantly enhanced by incorporating more complex and deeper neural networks. This integration of deep learning techniques with RL is referred to as Deep Reinforcement Learning (DRL). While some researchers might express concerns regarding the practicality of DRL in real-world learning scenarios, there is growing evidence of its successful application in diverse tasks. For instance, DRL has been effectively employed in achieving quadrupedal walking in robots, enabling the manipulation of unfamiliar objects, and facilitating the acquisition of a broad range of complex manipulation skills[9]. These successful applications have fueled the growing interest in exploring DRL algorithms for tackling the challenge of robotic grasping.

While Deep Reinforcement Learning DRL has emerged as a promising approach for robotic grasping, a critical review of existing research reveals limitations that necessitate further investigation. Many studies have primarily focused on simulated environments, lacking the real-world validation necessary for practical implementation. Furthermore, even studies venturing into physical experiments often neglect the crucial aspect of grasp evaluation. As previously established, evaluating a grasp's efficacy is fundamental to overcoming the inherent challenges in robotic grasping. Significantly, to the best of the author's knowledge, no prior research has explored the application of DRL for grasping tasks with a three-fingered gripper. This gap in the literature motivated the current proposal, which advocated for the development of a novel grasping system that leverages a continuous reward function specifically designed for a three-finger gripper.

The proposed reward function will encompass all critical stages of a successful grasp, guiding the agent's learning process towards achieving optimal grasps even for novel objects. This comprehensive approach aims to address the limitations of prior research by incorporating real-world applicability through a focus on robust grasps for unknown objects. Ultimately, this research project aspires to develop a grasping system capable of effectively manipulating unknown objects in real-world environments.

1.3 Purpose of the Study and Research Questions

This study investigated the application of DRL to address the challenge of grasp synthesis for three-finger grippers, specifically targeting the grasping of unknown objects. The primary objective was to establish a framework for autonomous grasping. Autonomous grasping systems equipped with agile grippers, like the three-finger gripper employed in this study, offered enhanced adaptability in handling diverse grasping tasks. Adaptability, in this context, refers to the system's ability to adjust its grasping strategies to a wider range of object shapes and sizes.

The potential benefits of autonomous grasping systems extended to Industry 4.0 applications, particularly when considering the cost-effectiveness of training. Compared to traditional methods that required extensive object-specific programming, DRL offered the possibility of training robots to grasp a wider variety of objects with less upfront investment.

This study utilized the Soft Actor-Critic (SAC) algorithm, a DRL algorithm well-suited for problems with high-dimensional state spaces, such as those encountered in robotic grasping. Unlike conventional DRL algorithms that solely focused on maximizing reward, SAC incorporated an additional term for entropy. This entropy term promoted exploration during training, enabling the agent to discover diverse grasping strategies and fostering robustness against environmental uncertainties.

The core objective of this research was to develop a grasp synthesis method that prioritized optimal grasping of novel objects. The trained agent was expected to achieve optimal configurations for a wide range of previously unseen objects. The following research question guided this investigation:

How can a reinforcement learning-based grasping system be developed to prioritize optimal and robust grasping, and handle novel objects without requiring specific training for each object?

This research question motivated the design and optimization of a grasping system that addressed the limitations of existing methods. Specifically, the proposed approach integrated a grasp evaluation metric into the reward function, guiding the agent to learn strategies that prioritized optimal grasps. Additionally, the system was designed to handle novel objects without extensive pre-training on every unique object encountered.

To evaluate the effectiveness of the proposed system, a comprehensive research plan was implemented. This plan involved deploying the system in both simulated and real-world scenarios, with performance comparisons drawn against existing grasping systems. The anticipated findings from this research held significant promise for advancements in the field of robotics and automation, potentially contributing to the development of more

efficient and reliable industrial processes. The specific objectives that guided this research were discussed in detail in the following section.

Objective 1: *Develop a physics-based simulation environment for modeling and evaluating robotic grasping capabilities.*

This objective will facilitate the development, training, and testing of the proposed grasping system. To achieve this goal, various software tools are available that enable the creation of robotic systems. These tools include Bullet, Gazebo, MuJoCo, CoppeliaSim, and NVIDIA Isaac Sim. Collectively known as physics engines, these software programs simulate the physical behavior of objects within virtual environments. Physics engines offer several advantages for robotics development. First, they allow developers to design, test, and refine robotic algorithms and applications in a safe and controlled setting before deployment on real-world robots. This significantly reduces the risk of damage to the physical robot during the development process. Second, these environments are ideal for pre-training intelligent agents, such as those utilizing deep reinforcement learning. Pre-training in simulation can significantly reduce the time and resources required for training on physical hardware. Finally, physics engines can introduce random noise into the simulation, primarily affecting camera output. This process, known as simulation augmentation, helps to increase the model's robustness and ability to generalize to real-world scenarios with inherent noise and variability.

Objective 2: *Validate the physics-based simulation model encompassing the robot and gripper through rigorous testing and comparison with real-world performance.*

This objective centers on validating the accuracy of the robot arm and gripper models developed within the simulation environment. This validation process constitutes a critical step prior to model training. By establishing a close correspondence between the simulated models' behavior and that of their real-world counterparts, we can identify and mitigate discrepancies between the simulation and the actual system. Reducing this disparity is paramount for enhancing the generalizability of the trained models. Consequently, a more seamless transition to real-world deployment can be achieved.

Objective 3: *Train a deep reinforcement learning agent within the developed simulation environment for optimizing robotic grasping performance.*

This objective facilitates the training of a reinforcement learning agent within a controlled simulation environment. To achieve a high grasping success rate, careful consideration must be given to the selection of appropriate algorithms, such as Soft Actor-Critic (SAC). Additionally, meticulous design of the reward function and deep neural network architecture is crucial. The core focus of this research lies in developing a robust reward function that incentives not only successful grasps, but also optimal configurations. Evalua-

tion of the grasped object's stability should be an integral component of the reward function. Furthermore, the training process is designed to be independent of the specific object being grasped. This allows the agent to generalize its grasping capabilities to previously unseen objects. As a consequence, the resulting grasping system exhibits enhanced generalizability, robustness, and stability.

1.4 Definitions

This section provides the definition of advanced terms used in this thesis. It starts with explaining Industry 4.0 and the role of agile robotic systems in this revolution. Then the concepts of manipulation and grasping are explained. Finally, the reinforcement learning algorithm is explained.

Industry 4.0, often referred to as the Fourth Industrial Revolution, heralds a transformative era in manufacturing characterized by the convergence of digital technologies and physical systems. At its core, Industry 4.0 represents a paradigm shift from traditional manufacturing methods to smart, interconnected production environments. This evolution is driven by cyber-physical systems, which integrate computation, networking, and physical processes, enabling machines and devices to communicate and collaborate seamlessly. The Internet of Things (IoT) plays a pivotal role, connecting sensors, actuators, and other equipment to gather and exchange vast amounts of data in real time. Cloud computing infrastructure provides scaleable storage and computational resources, facilitating data analysis and decision-making across the manufacturing ecosystem. Furthermore, cognitive computing technologies such as artificial intelligence (AI) and machine learning empower machines to learn from data, adapt to changing conditions, and make autonomous decisions. Central to Industry 4.0 is the concept of the smart factory, where interconnected systems self-monitor, self-optimize, and even anticipate issues before they arise, leading to unprecedented levels of efficiency, flexibility, and productivity. Within this landscape, robotics emerges as a cornerstone technology, enabling advanced automation, human-robot collaboration, and agile manufacturing processes.

Robotic systems are indispensable components of Industry 4.0, playing crucial roles in revolutionizing manufacturing operations. Collaborative robots, or cobots, enable safe interaction with humans, enhancing flexibility and agility in shared work spaces. They also tackle hazardous tasks, mitigating risks to human safety and health. Additionally, robotic systems contribute to data-driven decision-making and quality improvement by generating and analyzing vast amounts of operational data. Through seamless data sharing and collaboration with other manufacturing equipment, robots optimize processes, minimize waste, and maximize productivity. In essence, robotic systems serve as linchpins in the advancement towards smarter, interconnected manufacturing environments envisioned by

Industry 4.0.

The main function of the robotic systems in industry is manipulation which is defined as moving objects. Robotic manipulation refers to the ability of robots to grasp, move, and manipulate objects with precision and control. This fundamental capability enables robots to perform a wide range of tasks in various industries, from assembly line operations in manufacturing to handling delicate objects in healthcare settings. Through the use of sensors, actuators, and sophisticated algorithms, robotic systems can perceive their environment, plan optimal trajectories, and execute dexterous motions to interact with objects effectively. Robotic manipulation plays a vital role in automating repetitive tasks, increasing efficiency, and improving overall productivity in diverse applications.

Robotic grasping is a fundamental aspect of robotic manipulation, focusing specifically on the ability of robots to securely and effectively grasp objects using mechanical grippers or end-effectors. Technically, robotic grasping involves the design and implementation of algorithms and mechanisms that enable robots to analyze the shape, size, and weight of objects in their environment, determine suitable grasping strategies, and execute precise motions to achieve successful grasps. This process often requires the integration of sensory feedback, such as vision or tactile sensing, to adaptively adjust grasping strategies based on real-time environmental conditions. Robotic grasping is essential for a wide range of applications, including pick-and-place tasks in industrial automation, manipulation of objects in household robotics, and assistance with activities of daily living in healthcare settings.

The concept of reinforcement learning was first introduced in the 1950s by Richard Bellman and John Ragazzini, which is a type of machine learning where an agent learns to make decisions by interacting with an environment. The agent receives feedback in the form of rewards or punishments for its actions, which it uses to update its behavior and improve its decision-making abilities over time. The goal of reinforcement learning is for the agent to learn a policy, or a set of actions, that maximizes its cumulative rewards over the long term.

While some may consider DRL to be impractical in real-world learning situations, it demonstrated in [9] that DRL techniques have been effectively employed in a wide range of tasks, including quadrupedal walking, manipulating unfamiliar objects, and acquiring diverse and intricate manipulation abilities.

Many algorithms have been introduced for deep reinforcement learning networks; however, Soft Actor-Critic is particularly appealing for robotic applications. The Soft Actor-Critic (SAC)[10] algorithm is a DRL method that is commonly used for training agents in robotics and control tasks. Unlike other DRL algorithms, SAC can learn a stochastic policy that can balance the trade-off between maximizing reward and exploring new actions. This

makes SAC particularly effective for continuous control problems where the optimal policy is often a distribution of actions.

1.5 Thesis Organization

This thesis is comprised of six chapters, systematically exploring the development of a reinforcement learning-based grasping system for agile robotic manipulation in smart manufacturing environments. It commences with Chapter 1, which establishes the context of smart manufacturing and the growing role of agile robotic systems within this domain. Chapter 1 then delves into the concept of grasp synthesis, highlighting current research gaps and motivating the present investigation.

Chapter 2 provides a comprehensive overview of existing grasp synthesis methods through a critical examination of the application of various learning algorithms within the field of robotics. Chapter 3 delves into the theoretical foundations of reinforcement learning and its application within robotics, focusing on the specific reinforcement learning concepts and algorithms utilized throughout the thesis.

The methodology employed in this research is meticulously detailed in Chapter 4. This includes explanations of the inverse kinematics algorithm, gripper modeling and control within the simulation environment, environment construction, validation methods, and the rationale behind selecting the Soft Actor-Critic (SAC) algorithm for training the reinforcement learning agent. Chapter 5 presents the findings of the validation process alongside the training results achieved for the grasping system.

Finally, Chapter 6 critically analyzes the obtained results, comparing them to relevant studies within the field. This chapter also discusses the contributions and limitations of the present work, while offering potential avenues for future research and exploration.

1.6 Contributions

This thesis presents the development of a novel reinforcement learning environment specifically designed for smart grasping applications. This environment was constructed entirely from scratch, offering a unique platform for training and evaluating grasping algorithms. Furthermore, this work represents a pioneering application of a three-finger gripper within a reinforcement learning framework for smart grasping tasks.

Chapter 2

Literature Review

Robotic grasping remains a complex undertaking due to inherent challenges such as the physics of object manipulation, limitations in perception systems, the growing sophistication of gripper designs, and the critical task of selecting optimal grasp points. This research focuses on the sub-problem of grasp synthesis, which essentially involves identifying the most suitable locations on an object to secure it during manipulation.

Broadly, two primary approaches exist for grasp synthesis: analytical and empirical. Analytical methods rely on solving pre-defined equations to determine potential grasping locations. While this approach offers high accuracy and often yields multiple feasible grasp options, it necessitates a precise 3D model of the object being grasped. Conversely, empirical methods forgo explicit equation-solving; instead, they leverage data or human demonstrations to learn effective grasping strategies. Imitation learning, deep learning, and reinforcement learning all fall under the umbrella of empirical methods.

Among these empirical approaches, reinforcement learning (RL) offers a particularly attractive solution due to its potential for time and cost efficiency. Unlike supervised learning methods that require extensive labeled datasets, RL algorithms can learn grasping strategies through trial and error within a simulated environment. This research project leverages reinforcement learning for grasp synthesis tasks.

This chapter delves into the existing body of research related to intelligent grasping. The following sections will explore various aspects of this field:

- **Section 1:** Analytical Methods for Grasp Synthesis: This section will examine the theoretical foundations and applications of analytical approaches to grasp planning.
- **Section 2:** Empirical Methods in Robotics: This section will provide an overview of various empirical learning techniques used in robotic grasping tasks.

- **Section 3:** Deep Dives into Empirical Grasping Techniques: This section will delve deeper into three specific empirical methods: deep learning, imitation learning, and reinforcement learning. Each subsection will review relevant research studies concerning the application of that particular method to robotic grasping.

2.1 Analytical Grasp Synthesis Methods

Analytical methods in grasping refer to approaches that use geometrical and mathematical models considering kinematics and dynamics formulation to analyze and predict the best grasping configuration and forces for a given object. These methods often involve analyzing the shape, size, and other physical properties of the object, as well as the capabilities of the robotic gripper or end-effector. Analytical methods can be used to determine the optimal grasp location, the required grasp force and torque, and the contact points to achieve a optimal grasp. These methods are used in specific cases, and the difference between the model and the real environment is an issue.

Liu[11] proposed an algorithmic solution is proposed to obtain a 3D force-closure grasp for objects of any type, whether they have a curved surface or not, and with or without friction. The method involves a combination of local and recursive strategies for problem decomposition. Initially, the algorithm inserts n-contacts on the object and searches recursively for a convex hull that contains the origin. Once a local minimum is reached, the algorithm uses the recursive decomposition method to break down the problem into sub-problems. Although numerical examples are used to evaluate the results, it cannot be completely guaranteed that a feasible grasp will always be found due to the computational complexity, which depends on the number of contacts used.

In [12], the author's focus was on achieving strong force-closure grasps for objects of varying shapes using hands with multiple fingers. To accomplish this, the authors utilized a method in which they randomly generated the position of fingers and then determined the position of the final finger by creating a strictly negative linear combination with one of the first generated fingers' wrench basis. This approach allowed for a faster algorithm but came at the cost of a lower success rate.

After developing analytical methods for grasping spheres, cylinders, cones, and boxes, Miller[13] explored the possibility of approximating other objects to these primitive shapes. To do this, some methods decomposed the object model into a set of primitives and perform grasping on each part separately. This simplified the grasping process, and heuristics methods are proposed to complement analytical approaches in ensuring grasp quality.

Morales et al.[14] proposed an integrated grasping framework incorporating a visual system to enable humanoid grippers to grasp objects commonly found in kitchen environ-

ments. Their method utilizes a database containing object models and pre-defined grasping plans for each object. This database is populated using offline grasp planning software. The software simulates interactions between object and hand models to identify stable grasps for each object. These grasps are characterized by features such as grasp type, starting point, approach direction and line, and hand orientation. Upon completion of the database, the real-time vision system detects and localizes objects, enabling their retrieval from the database for grasp planning. Notably, the authors did not evaluate the performance of the entire grasping framework, but only the object detection and localization capabilities of the vision system. A significant limitation of this framework is its dependence on a pre-established, offline database. This renders it incapable of adapting to dynamic environments. Additionally, the paper lacks performance and efficiency metrics for the overall system.

De Souza et al.[15] proposed a functional grasping framework for picking aerospace industry objects, facilitating use by factory floor operators. This framework comprises two key components: an offline grasp planning algorithm and an online grasp selection module. The offline stage utilizes the "GraspIt!" simulator to generate candidate grasps through an embedded Very Fast Simulated Re-Annealing (VFSA) optimization algorithm[16]. During online operation, the robotic program selects the optimal grasp candidate based on predefined factors and priorities. These factors include joint space utilization, depth distance, Euclidean distance, and various rotational distances (roll, pitch, yaw). The authors evaluated the framework using representative aerospace industry objects and presented quantitative results. However, a significant limitation remains: the requirement for a human supervisor to monitor the chosen grasp candidate.

Jain et al.[17] proposed a novel algorithm for detecting robotic grasps suitable for assistive manipulation systems. The system leverages a point cloud depth camera to capture a 2.5D scene representation and subsequently segments it into individual objects. For each object, surface properties are analyzed to identify the best-fitting geometric primitive (sphere, cylinder, or box). Building upon the primitive classification, the authors define strategies that mimic human grasping preferences to generate multiple candidate grasps of different types (top, side, or pinch) for each object. The system's efficacy was evaluated in real-world experiments using an assistive robotic arm on a set of 30 household objects. The results demonstrated the system's capability to generate multiple valid grasps for most objects. However, the authors acknowledged a limitation: the current algorithm's inability to handle objects with irregular shapes.

Goldfeder et al.[18] introduced a method for grasp planning that substantially reduces the number of simulations needed to identify suitable grasps for complex objects. This is achieved by decomposing the object into a hierarchy of simpler shapes known as su-

perquadrics. Superquadrics are a parametric shape class capable of representing a wide range of geometric features. The authors propose an automated algorithm for decomposing a 3D object model into a tree structure of superquadrics. This decomposition tree progressively captures the object's overall form at higher levels and finer details at lower levels. The grasp planning method iteratively traverses the decomposition tree, evaluating candidate grasps on the simpler superquadric shapes at each level. Grasps deemed infeasible at higher levels in the tree are pruned from further evaluation, significantly reducing the computational burden of complex simulations. The authors evaluate their method on a set of test objects, demonstrating its success in finding stable grasps for various shapes. In conclusion, the paper presents a novel grasp planning method that leverages decomposition trees to efficiently identify suitable grasps for complex objects. This method reduces the computational cost of grasp planning by eliminating infeasible grasps early in the process. However, the accuracy of the method is acknowledged to be contingent upon the quality of the superquadric decomposition, highlighting the need for further research on improving decomposition algorithms.

Aleotti and Caselli[19]introduced a novel robot grasping method that leverages topological segmentation of 3D objects. This approach departs from traditional methods reliant on predefined grasp taxonomies or rule-based algorithms. The authors propose a system that first segments a 3D object model into parts using a Reeb graph. A Reeb graph is a mathematical structure that captures the object's essential shape characteristics by representing its skeletal form. By analyzing this graph, the system identifies potential grasping points on the object's parts, likely corresponding to natural grasp affordances exploited by humans. Following segmentation, the system classifies the object (e.g., cup, table) using a 3D shape retrieval algorithm. This classification enables the system to associate semantic information with the identified grasping points. For instance, the system might recognize a cup handle and prioritize grasps that incorporate it. The paper evaluates the proposed method on a set of 3D object models, demonstrating its success in identifying multiple valid grasps for various objects. The authors conclude that their method offers a promising approach for robot grasping, leveraging semantic information about object parts to achieve more natural and efficient grasping capabilities. However, limitations include the computational cost of Reeb graph computation, particularly for complex objects. Additionally, the method's accuracy relies on the quality of the 3D object models and the shape retrieval algorithm used for object classification.

Rakesh et al.[20] proposed a method for optimizing robotic grasping of complex objects using a modified genetic algorithm (GA). This GA searches for high-quality grasps by evaluating a broad range of potential contact points on the object's surface. The method commences by generating an initial feasible grasp using an established algorithm. The GA then iteratively refines the grasp quality by assessing various combinations of contact points.

Grasp quality is measured by the radius of the largest inscribed sphere within the convex hull of the grasp wrenches. The authors evaluated their method on a set of test objects, demonstrating its success in finding high-quality grasps for various shapes. However, a significant limitation lies in the algorithm's exhaustive search nature. This can become computationally expensive for complex objects with numerous potential contact points.

Ciocarlie[21] explored the application of low-dimensional posture subspaces for robotic hands. This work investigated using a subspace of hand configurations, called eigengrasps, for grasping objects with dexterous robotic hands. Eigengrasps are a set of basis vectors that capture the majority of the variation observed in human hand postures during grasping tasks. The author demonstrated that by utilizing this reduced-dimensionality approach, successful grasps can be achieved for various objects with minimal computational burden. This method is particularly advantageous for dexterous hands with many degrees of freedom, as it simplifies the complex challenge of optimizing finger configurations for grasping. However, the effectiveness of this approach is acknowledged to be contingent upon the selection of eigengrasps. Further research is necessary to determine the optimal subspace for different tasks and hand designs.

As outlined in the previous section, the analytical approach suffers from various issues, such as intricate modeling, computational complexity, practical limitations, and narrow assumptions. Due to these limitations, there has been a surge of interest in the use of learning methods, and its advancement is on the rise. This trend is evidenced by the increasing number of research papers on this topic. In the following sections, studies in learning based methods for robotics tasks with a focus on grasp planning are reviewed.

2.2 Empirical Methods in Robotics

Learning-based methods are becoming an inseparable part of robotics. Due to their ability to handle uncertainties in real-world environments and complex models, there is a growing interest in these methods. This section analyzes studies in learning-based methods for robotics, divided into two main areas. First, we explore works regarding model-based learning methods. Second, we review the application of learning methods in mobile robots, followed by a comprehensive review of learning-based methods for grasping.

Levin[22] developed a method for training visuomotor policies that can directly learn control policies from raw image observations for robots. The authors proposed a guided policy search algorithm that combines deep convolutional neural networks (CNNs) with trajectory optimization. This allowed the robot to learn complex visuomotor tasks without requiring separate perception and control modules. The method was evaluated on a variety of real-world manipulation tasks, including screwing a cap onto a bottle and grasping a toy hammer, demonstrating successful learning and improved consistency compared to training

the vision and control components separately. However, the approach required the full state of the system to be known during training, which may not always be feasible in real-world applications.

Chebotar[23] presented a policy search method for learning complex feedback control policies for robotic manipulation tasks with discontinuous contact dynamics. The authors proposed a method called path integral guided policy search (PI2GPS) that incorporates a model-free local policy optimizer based on path integral stochastic optimal control (PI2). This enabled the method to handle tasks with highly discontinuous contact dynamics, such as door opening or picking and placing objects. Additionally, PI2GPS utilized on-policy sampling, which allows the robot to continually learn and improve policies on new task instances as they are experienced in the world. The authors demonstrated the effectiveness of their method by applying it to real-world door opening and pick-and-place tasks. Their results showed that PI2GPS outperforms prior policy search methods, particularly for tasks with discontinuous contact dynamics. However, the method required access to the full state of the system during training, which may not be feasible in all real-world scenarios.

Ha et. al.[24] purposed of the paper is to presented an automated environment for deep reinforcement learning of locomotion tasks for a reconfigurable legged robot. The authors proposed a system that utilized a vision-based tracking system and a resetting mechanism to allow the robot to learn directly on hardware. They employ two deep reinforcement learning algorithms, Trust Region Policy Optimization (TRPO) and Deep Deterministic Policy Gradient (DDPG), to train neural network policies for simple rowing and crawling motions. The experimental results demonstrated that both algorithms are able to learn effective control policies directly on the hardware for three different leg types. The authors also showed that transferring the learned policies to more complex configurations can expedite the learning process. A limitation of this approach was that it required manually specifying a reward function, which can be challenging to design for complex tasks.

Jemin et. al.[25] developed a method for training agile and dynamic motor skills for legged robots using reinforcement learning (RL). The authors proposed a combined approach that leveraged the benefits of both simulation and real-world training. They addressed the limitations of prior methods by introducing a data-driven approach that utilized a deep neural network to model the complex dynamics of the robot actuators. This allowed the learned controller to be directly transferred from simulation to the real robot without the need for extensive fine-tuning. The results showed that the proposed method enables the robot to learn various complex behaviors, including following velocity commands, running at faster speeds, and recovering from falls. These achievements surpassed the capabilities of previous control methods for legged robots. However, the method relied on accurately modeling the robot's actuators, which may not be feasible for all legged robot designs.

The authors in[26] presented a robust controller for blind quadrupedal locomotion on challenging terrain. The authors proposed a model-free reinforcement learning (RL) approach that utilized a proprioceptive policy trained in simulation to control the robot's locomotion in real-world environments. The policy was trained on a simulated ANYmal quadruped robot with a focus on adaptability and robustness. The robot successfully navigated various challenging terrains, including steep slopes, mud, snow, and debris, demonstrating the effectiveness of the learned policy for real-world quadrupedal locomotion. The authors concluded that their method enables robots to learn complex behaviors in simulation and then generalize them to real-world settings without extensive fine-tuning. A limitation of the approach was that it required a powerful simulation environment and may not be applicable to legged robots with significantly different hardware designs.

The authors in[27] presented a policy gradient reinforcement learning approach for generating a fast walk on legged robots. The authors proposed a method for automatically learning a gait for a quadrupedal robot by optimizing a set of parameters that define the gait's walking pattern. They used a Sony Aibo ERS-210A robot as their experimental platform. Their method achieved a gait that was faster than both hand-tuned and previously learned gaits. The authors concluded that their policy gradient reinforcement learning approach was a viable method for automatically learning fast gaits for legged robots. However, they noted that there are limitations to their approach, such as the amount of time it toke to train a robot to walk using this method.

The authors in[28] demonstrated that deep reinforcement learning can be used to efficiently train a quadruped robot to walk directly on various real-world terrains. The authors proposed a method that utilizes deep Q-learning with careful design decisions to achieve sample-efficient learning. Their results showed that their method can train a quadrupedal robot to walk in just 20 minutes on various terrains, including flat ground, soft mulch, grass, and a hiking trail. This was significantly faster than previous methods, which often require hours or days of training in simulation. The authors concluded that their method is a promising approach for real-world robot locomotion tasks. However, they also acknowledged that their method had limitations, such as the computational cost of training and the need for careful design decisions.

2.3 Empirical Grasp Synthesis Methods

Empirical methods in robotic grasping involve using data-driven approaches to learn grasping policies or grasping plans directly from examples or through trial-and-error. These methods are often used when the analytical models are difficult to define or when the physical properties of the object are complex or uncertain. Empirical methods can involve using various machine learning algorithms such as deep learning, reinforcement learning,

and imitation learning.

2.3.1 Machine Learning

The purpose of the paper by Jiang et al.[29] was to develop a new method for grasping novel objects using a robot gripper . Their method involved learning a grasping rectangle representation from images and depth maps. This rectangle representation encodes the 3D location, orientation, and opening width of the gripper. The authors proposed a two-step learning algorithm to efficiently find the best grasping rectangle for a given object. In the first step, they used fast-to-compute features to reduce the search space to a small number of candidate rectangles. In the second step, they used more accurate but computationally expensive features to select the best rectangle from the remaining candidates. Their experimental results showed that their method enabled a robot to successfully grasp a variety of novel objects. However, their method relied on a learning approach that required labeled training data, which can be time-consuming to collect.

The purpose of the paper by Mahler et al.[30] was to develop a method for training a deep learning model to plan robust grasping positions for robotic arms. Their method involved training a Convolutional Neural Network (CNN) on a large dataset of synthetic point clouds, grasps, and corresponding analytic grasp metrics. This dataset, called Dex-Net 2.0, was generated using a physics simulator to simulate thousands of 3D models in various random poses on a table. The CNN model, called the Grasp Quality Convolutional Neural Network (GQ-CNN), was then used to predict the probability of success for a given grasp pose based on a depth image from a camera. The authors found that the GQ-CNN model could achieve high accuracy in predicting successful grasps, even for objects that the model had never seen before. This suggests that the model was able to learn generalizable features of robust grasping that are applicable to a wide variety of objects. However, the limitations of this approach include the need for a large amount of training data and the computational cost of training the CNN model.

The purpose of the paper by Andreas ten Pas and Robert Platt[31] was to develop an algorithm that detects grasp poses in 3D point clouds using geometric reasoning. Their method first samples a set of candidate hand placements on the object surface, and then uses machine learning to classify the candidates as valid grasps or not. A key aspect of their approach is that it leverages geometric constraints to reduce the number of candidate placements that need to be considered. This makes the grasp detection process more efficient. The authors found that their method achieved promising results, even in cluttered environments. However, their method requires a significant amount of training data to be effective.

The purpose of the paper[32] was to develop a new approach to robotic grasping that

utilizes machine learning techniques. The authors proposed a method that uses a Support Vector Machine (SVM) to learn a mapping between object shape, grasp parameters, and grasp quality. This allows the robot to find grasps that are likely to be successful for a given object, even if the object is unfamiliar. The authors tested their method on a simulated robotic hand and found that it could achieve good results. However, they also acknowledged that their method is limited by the need for training data, which can be time-consuming to collect.

2.3.2 Deep Learning Methods

Lenz[33] proposed a deep learning algorithm for detecting robotic grasps, specifically focusing on rectangle-based grasp detection in cluttered lab scenes. The authors proposed a two-stage cascaded system with deep learning networks to avoid time-consuming hand-designed features. The first network had fewer features, is faster to run, and can effectively prune out unlikely candidate grasps. The second, with more features, was slower but only had to run on the top few detections from the first stage. To handle multimodal RGB-D data, the authors introduced a structured regularization method on the weights based on multimodal group regularization. Their results showed that this method improved performance on an RGB-D robotic grasping dataset, and can be used to successfully execute grasps on two different robotic platforms. However, the limitations were that the system is computationally expensive and may not be suitable for real-time applications.

The purpose of the study[34] was to develop an accurate, real-time system for robotic grasp detection using convolutional neural networks. The authors proposed a single-stage regression approach to directly predict graspable bounding boxes, eliminating the need for standard sliding window or region proposal techniques. Their network achieved an accuracy of 88% and could run at 13 frames per second, significantly outperforming previous state-of-the-art methods. This work addressed the limitations of prior robotic grasp detection systems by eliminating the need for complex 3D models and hand-crafted features. However, the system relied on a large convolutional neural network, which may not be suitable for resource-constrained robots.

Guo[35] proposed a novel hybrid deep architecture for real-world robotic grasp detection. This architecture combined visual and tactile sensing to enhance a robot's perception and grasping ability. The authors proposed a data collection phase where both visual and tactile data were collected. A deep visual network and a deep tactile network were then used in the grasp detection phase. The tactile sensing data was used to improve the learning ability of the visual network. The authors found that their method achieved better grasp detection accuracy than previous state-of-the-art methods. However, a limitation of this approach is the computational cost associated with deep learning networks.

Chu[36] developed a deep learning architecture for real-time multi-object, multi-grasp detection. The authors proposed a convolutional neural network (CNN) architecture that can predict multiple grasp candidates for a single object or multiple objects in a single image. Their method avoids the need for separate object segmentation and grasp classification steps. The proposed system achieved 96.0% and 96.1% accuracy on image-wise and object-wise splits of the Cornell dataset, outperforming previous state-of-the-art methods. It also achieved a grasping success rate of 89% on a test set of household objects. A limitation of the approach is that it requires a large amount of training data.

The purpose of the author in[37] was to address the ambiguity inherent in robotic grasping by proposing a novel method for detecting multiple grasp hypotheses from a single RGB image. The authors propose a deep learning model that predicts Two-Dimensional (2D) grasp belief maps, which encode the spatial uncertainty around a grasp location, rather than traditional bounding boxes. They argue that this method allows the model to better capture the distribution of possible grasps for an object. The model is trained using a multi-grasp prediction framework, which allows it to learn from a wider range of grasp configurations. The authors found that their method achieved a grasp detection accuracy of over 90% for unseen objects, outperforming previous state-of-the-art methods. However, a limitation of this approach is the computational cost associated with training deep learning models.

The purpose of the paper[38] was to improve the accuracy of robotic grasp detection in cluttered environments. The authors proposed a novel grasp detection method that incorporates surface normals and multiple views into a single descriptor to improve grasp classification accuracy. They also introduced a new benchmark task to systematically evaluate grasp success rates in dense clutter. Their results showed that their method achieved an average 93% end-to-end grasp success rate for novel objects presented in dense clutter. However, a limitation of this approach is that it requires a large amount of training data.

2.3.3 Imitation Learning

The purpose of the paper[39] was to develop a novel CNN-based imitating learning framework for selecting robots' grasping points. This framework learns from human grasping operations captured on camera. The authors achieved this by extracting key points from an RGB image sequence, which were then converted into an array and used to estimate the grasping posture sequence. They found that their imitating learning algorithm could help a dual-arm robot master the correct grasping posture of an object within only 20 minutes. This is a 26.1% improvement in grasping planning efficiency compared to traditional geometric modeling-based methods. However, the authors acknowledge that their method is limited by the need for human grasping data, which can be time-consuming to collect.

The purpose of the paper[40] was to present an autonomous grasping approach for complex-shaped objects using a high-DOF robotic hand. The authors proposed a system that learned how to utilize a high-DOF human-like robotic hand for manipulation using a variety of machine learning approaches, including ResNet, KNN, point-set registration, and dynamical movement primitives (DMPs). Unlike prior works that used imitation learning for robotic manipulation, this system learned grasping poses, hand trajectories, and finger motions for grasping complex-shaped objects from a limited number of demonstrations. The system achieved promising results grasping five objects with different shapes and orientations. However, the limitations of the system include the requirement for a large amount of training data and the computational cost of the machine learning algorithms.

2.3.4 Reinforcement Learning

The section highlights that the main challenge of implementing deep learning policies is the requirement for a large, labeled database. However, even with a large database, it is difficult to assess if the labeled data sufficiently captures all necessary practical assumptions. Hence, some researchers tried to address this issue by using deep reinforcement learning methods.

The studies pertaining to grasping using deep reinforcement learning are comprehensively summarized in Table 2.1. For each paper, the targeted task, the employed reinforcement learning algorithm, the observation technique, the reward functions, and the grasp success rate are compiled and presented.

The first and most notable work in this was conducted by Kalashinkov et. al.[41] in which they studied how to learn vision-based grasping policies using deep reinforcement learning. The authors proposed a method called QT-Opt, which is a scalable off-policy reinforcement learning algorithm for continuous action spaces. Their method uses a deep Q-network to directly predict the value of taking an action in a given state, without needing an explicit actor network. They found that this method achieved high success rates on unseen objects, even when trained with a simple reward function of just success or failure. This suggests that their method can learn complex grasping strategies through self-supervised learning. However, a limitation of this approach is the large amount of data required to train the deep Q-network.

This work served as a foundational exploration, inspiring subsequent researchers to employ reinforcement learning algorithms for various robot grasping tasks, including pick-and-place operations, grasping moving objects, and handling unknown adjacent objects. The subsequent paragraphs provide a review of these endeavors.

The author in[43] proposed a framework for robots to pick up clustered objects on a table using a combination of deep reinforcement learning and a rule-based method. The

authors argued that this method would be more efficient than using reinforcement learning alone, which can be time-consuming for robots to learn grasping skills. Their method involved dividing the robot's actions into two categories: pushing, which was learned using deep reinforcement learning, and grasping, which was determined by a rule-based image morphological processing algorithm. The pushing action aimed to separate stacked objects and create suitable grasping points for the robot's gripper. The grasping algorithm assessed the resulting image from the pushing action to determine if a grasp was possible and would return a reward signal for the pushing action's learning process. The authors found that their method achieved a high grasp rate with low computational complexity, enabling robots to quickly clear cluttered tables. However, they acknowledged that their approach might not be suitable for grasping highly deformable objects because the grasping algorithm relied on image processing.

The author in[42] developed a deep reinforcement learning framework for robotic grasping using visio-motor feedback. The authors proposed a novel Grasp-Q-Network to output grasp probabilities used to learn grasps that maximize pick success. They used a double deep Q-learning framework along with a visual servoing mechanism that uses a multi-view camera setup to observe the scene which contains the objects of interest. The experiment results showed that the proposed method outperformed the baseline Q-learning framework and increased grasping accuracy by adapting a multi-view model in comparison to a single-view model.

The author in[44] investigated the use of Deep Reinforcement Learning (RL) to train a robotic agent to perform pick-and-place tasks without prior knowledge of the objects. The authors proposed a system that uses a Convolutional Neural Network (CNN) to estimate Q-values, which are then used by an ϵ-greedy policy to determine the robot's actions. The system was tested in a simulated environment where the robot learned to grasp objects based on RGBD camera images. The results showed that the MobileNet pre-trained CNN achieved the best performance, with an average accuracy of 84% after training in the simulated environment. The limitations of the study include the use of a simulated environment and the fact that the robot only learned to grasp objects within a square-shaped area due to limitations of the CNN input size.

One of the most challenging task in this context is capturing moving objects. Regarding this task, Chen[45] developed a deep reinforcement learning (DRL) grasping framework for moving objects in unstructured environments. The system integrated a DRL algorithm, Soft Actor-Critic, with object detection techniques to implement an approaching-tracking-grasping scheme. The researchers employed a Kinect depth sensor to capture real-time three-dimensional coordinates of the moving object. They proposed an improved Soft Actor-Critic algorithm to handle the high-degree-of-freedom manipulator's state and action space.

Simulation results demonstrated that the system could autonomously grasp a moving object with different moving trajectories. However, the limitations of the study were that the object detection module relied on simulation and did not learn from depth images in real-time.

The author in [46] introduced a self-learning robotic grasping strategy based on model-free deep reinforcement learning, named Deep Reinforcement Grasp Policy (DRGP). This method aimed to achieve grasping in robotic manipulation for unknown objects (objects not seen during training) based on limited prior knowledge. The system learned from scratch, from only visual observations to decisions-making, and trained in an end-to-end, trial-and-error manner. The researchers achieved this by using a deep neural network consisting of a visual network to extract features from visual observations and a policy network to map the features into the action space, both trained simultaneously. Their results demonstrated successful grasping on novel objects in real-world scenarios with minimal training time and simple objects in simulation. This suggests that the method can generalize to unseen objects.

The author in [47] proposed a new approach for grasping objects using a robotic arm by combining deep reinforcement learning with inverse kinematics. The researchers employed You Only Look Once v5 (YOLOv5) for object detection and backward projection to determine the 3D position of the target object. Then, they used deep deterministic policy gradient (DDPG) to train the robotic arm to reach the desired grasping position by calculating the angles of its joints. Their method achieved a 95.5% accuracy rate after only 400 episodes, outperforming other state-of-the-art approaches.

The author in [48] addressed two challenges associated with robotic grasping in cluttered and occluded environments: coordinating push and grasp actions, and avoiding occlusion of the camera. The authors proposed a multi-view and change observation-based approach (MV-COBA) that utilizes two cameras to capture multiple views of the workspace and employs change observation to determine which action (grasp or push) to take. The system achieved an average grasp success rate of 83.6%, 86.3%, and 97.8% in cluttered, well-ordered object, and occlusion scenarios, respectively.

Al-Shanoon et al.[49] developed a self-learning strategy for robots to grasp novel objects in challenging environments, where adjacent objects are cluttered together. The researchers proposed a method called Shift-to- grasp (StG) that combines deep reinforcement learning with pre-grasping actions. In StG, the robot first learns to move (shift) objects to create space for grasping and then learns to perform the grasping action itself. The system is trained in a simulation environment and then tested on real-world objects. The researchers found that StG was successful in grasping novel objects in challenging scenarios.

Ji et al.[50] improved grasping control of a vision robot using deep reinforcement learn-

ing. The authors proposed a Deep Attentive Deterministic Policy Gradient (DADPG) algorithm that incorporates an attention region proposal network (ARPN) for pre-exploration and a stratified reward function. The ARPN reduces the amount of irrelevant information processed by the network by focusing on the target object before sending the image data to the actor-critic network. The stratified reward function assigns rewards based on the distance between the gripper and the center of the pre-exploration area, improving the training efficiency. The results showed that DADPG achieved good performance in grasping tasks with noisy environments.

The author in [51] proposed a method for learning robot grasping tasks in simulation and then transferring the learned behavior to a real robot. The authors used model-free reinforcement learning algorithms to learn control actions for the robot directly from simulated data. They compared the performance of two algorithms: Proximal Policy Optimization (PPO) and Soft Actor-Critic (SAC). The learning process was accelerated by using a fine-tuning procedure, where a policy learned for a base task was used to initialize the learning process for a new, similar task. The authors designed a reward function that encouraged successful grasping and lifting of the target object, while also penalizing the robot for collisions and jerky movements. Their results showed that the proposed approach could successfully learn grasping tasks in simulation and transfer them to a real robot, achieving 100% success rates in grasping tasks.

You et al.[52] developed a more efficient method for robot grasping by combining deep reinforcement learning with prioritized experience replay. The authors proposed a method called Cross-entropy Soft Actor-Critic with Prioritized Experience Replay (CSAC-PER). CSAC-PER uses a Cross-entropy method to select the optimal action in the continuous action space and a prioritized experience replay strategy to improve the training efficiency. The results showed that CSAC-PER achieved a higher grasping success rate (90%) than other reinforcement learning algorithms. However, the researchers noted that this method requires more training time initially.

Li et al.[53] proposed a reinforcement learning method for four-pin grippers to achieve form closure grasping. The authors argued that this method would be more effective than traditional synthetic form closure grasping methods, which can suffer from poor efficiency and unmodeled uncertainties. The authors designed an improved four-pin gripper that extended the parameter space of previous designs. They also proposed a reward function based on environmental constraints to guide the reinforcement learning process. In their experiments, the authors found that their method was successful in planning stable grasping configurations for 2.5D objects in a simulated environment. However, they acknowledged that their method would need to be further validated in real-world grasping tasks.

The author in[54] developed a robotic object grasping approach that combines a com-

puter vision-based object detection/recognition/localization algorithm and a deep reinforcement learning algorithm with self-learning capability. The proposed approach utilizes the YOLO algorithm to detect and classify objects of interest within the field of view of a camera. Then, based on the detection results, the Soft Actor-Critic (SAC) deep reinforcement learning algorithm is employed to provide a desired grasping pose for the robot manipulator to perform object grasping. In order to speed up the training process and reduce the cost of training data collection, the authors implemented the Sim-to-Real technique, which utilizes a V-REP simulation environment to train the deep reinforcement learning neural network. The experimental results demonstrated that the 6-DOF industrial manipulator successfully performed object grasping with the proposed approach, even for previously unseen objects.

The author in[55] developed a deep reinforcement learning framework for robots to perform grasping and pushing manipulations of objects in cluttered environments. The authors proposed a method that combines prehensile grasping and non-prehensile pushing manipulations, allowing the robot to handle situations where grasping alone is difficult. Their approach utilizes a deep Q-network (DQN) to learn affordances for each action, enabling the robot to select the most appropriate action (grasping, left-sliding, or right-sliding) for a given pixel location in the workspace. The authors achieved promising results in a simulated environment, demonstrating that their method can successfully handle various cluttered scenarios with regular and irregular-shaped objects. However, they acknowledged that the framework would need to be validated in real-world settings to assess its generalizability.

Table 2.1: Systematic Review of Smart Grasping Using Reinforcement Learning

Year	Task	Method	Observation	Reward	Gripper
2018[41]	Unknown Objects	Q-Learning	Image	Binary	2-Finger
2020[42]	Grasping	Double Q-Learning	Image	Binary	2-Finger
2020[43]	Grasping in Clutter	Twined Delayed DDPG	Depth Image	Binary	2-Finger
2021[44]	Grasping	DQN	RGBD	Distance	2-Finger
2021[45]	Moving Objects	SAC	Position, Velocity	Distance, Velocity, Binary	2-Finger
2021[46]	Unknown Objects	Q-Learning	Height map	Binary	2-Finger
2021[47]	Autonomous Grasping	DDPG	Position	Distance	2-Finger
2021[48]	Clutter	Q-Learning	Position	Distance	2-Finger
2022[49]	Unknown adjacent objects	Q-Learning	Height map	Binary	2-Finger
2022[50]	Grasping	Improved DDPG	Feature Map	Distance Binary	2-Finger
2022[51]	Grasping	PPO, SAC	Position	Distance, Binary	2-Finger
2022[52]	Grasping, Pushing	Modified AC	Position	Distance, Binary	2-Finger
2023[53]	Stable Grasping	PPO	Position, Orientation	GQS	2-Finger
2023[54]	Unknown Objects	SAC	Position	Binary	Vacuum
2023[55]	Grasping in Clutter	DQN	Height map	Discrete	2-Finger
2024[56]	Unknown Objects	Multi-Agent TD3 With High-Quality Memory	RGBD	Distance, Binary	2-Finger
2024[57]	Clutter	SAC	RGBD	Empty Bin	2-Finger
2024[58]	Grasping, Pushing	DQN	RGBD	Binary	2-Finger
2024[59]	Underwater Grasping	Proximal Policy Optimization	Position Velocity	Distance, Orientation	Hand

Chapter 3

Preliminaries and Theoretical Background

This chapter lays the groundwork for the methods used in this research. Reinforcement learning, a relatively new approach in machine learning, will be given particular attention. A solid understanding of its core principles is necessary to both implement the algorithm effectively and analyze the results accurately.

The concept of reinforcement learning emerged in the 1950s through the work of Richard Bellman and John Ragazzini. It's a unique type of machine learning where an agent interacts with an environment to learn decision-making. The agent receives feedback in the form of rewards for good actions and penalties for bad ones. This feedback allows the agent to adjust its behavior and improve its decision-making abilities over time. The ultimate goal of reinforcement learning is for the agent to develop a policy, essentially a set of actions, that maximizes its total rewards in the long run. The following sections will explore the key terms and background information for three of the most recent DRL algorithms. Section 3.1 will introduce the common terminology used in this field. Section 3.2 delves into Q-Learning, considered the most popular algorithm and the foundation for many others. Q-Learning aims to establish a connection between the current state of the system and the value associated with that state. Section 3.4 focuses on policy-based algorithms, the other significant foundation for novel algorithms. These algorithms strive to create a link between the system's state and the optimal action for that specific state. Section 3.5 will examine how these approaches are combined to create a new class of algorithms called actor-critic. The subsequent sections will introduce the successors of actor-critic algorithms, including Deep Deterministic Policy Gradient (DDPG) in section 3.6, Twin Delayed DDPG (TD3) in section 3.7, and Soft Actor-Critic (SAC) in section 3.8.

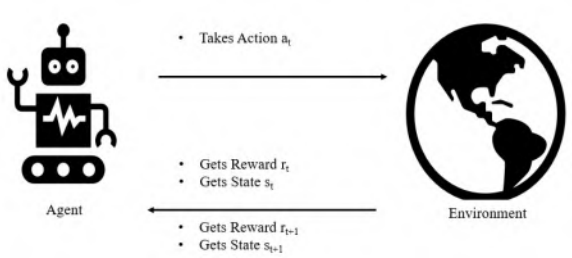


Figure 3.1: The RL Process: State, Action, and Reward Loop

3.1 Introduction to Reinforcement Learning

This section covers the RL problem and introduces common terms associated with this machine learning algorithm.

The primary motivation of RL lies in an artificial agent's ability to learn a task autonomously through interactions with its environment. Unlike other machine learning algorithms that rely on datasets containing known inputs and corresponding outputs, RL operates differently. Instead of relying on datasets, RL utilizes an environment where the agent learns the best action for each state. This approach is inspired by humans' natural learning experiences, where we gradually acquire skills by interacting with our surroundings. RL serves as a computational approach to learning from experience. The formal definition of RL is provided by Kaelbling[60]:

"Reinforcement learning is a general framework for making sequential decisions. An agent observes the state of the world, takes an action, and receives a numerical reward signal indicating the goodness of the action. The agent's goal is to learn a policy, a mapping from states to actions, that maximizes the total amount of reward it receives in the long run."

In the RL process, the agent engages in interactions with the environment. During these interactions, the agent receives information about the current state S_t and the current reward R_t. Subsequently, the agent selects an action A_t based on this information. As a result of this action, the environment undergoes a change, transitioning to a new state S_{t+1} and providing a new reward R_{t+1} to the agent. This iterative interaction process continues, allowing the agent to gradually learn a behavior that maximizes the potential reward.

Figure 3.1 illustrates the schematic of this RL framework, depicting the sequential flow of interactions between the agent and the environment. Through repeated iterations of receiving information, selecting actions, and observing outcomes, the agent refines its decision-making process to optimize the cumulative rewards obtained over time.

This process is also referred to as a Markov Decision Process (MDP). A key aspect of MDPs is the Markov Property. This property states that the future state of the environment

depends only on the current state and the action taken by the agent, not on the entire history of past states and actions. In simpler terms, the agent only needs information about the current situation (state) to make an informed decision about the next action (without needing to remember everything that happened before). This characteristic simplifies the learning process for the agent, allowing it to focus on the current context rather than the entire history.

In RL the total reward an agent accumulates over time is also referred to as the expected return. A fundamental question arises: why is maximizing reward the agent's objective? This concept is rooted in the reward hypothesis, which suggests that all goals can be expressed as maximizing a single value – reward in the case of RL. The cumulative reward or the expected reward is defined as:

$$R(\tau) = r_{t+1} + r_{t+2} + r_{t+3} + \cdots \tag{3.1.1}$$

which is equivalent to:

$$R(\tau) = \sum_{k=0}^{\infty} r_{t+k+1} \tag{3.1.2}$$

The issue with this summing rewards together is that the rewards in the initial steps are more likely to happen than the rewards in the final steps. The discount rate γ between 0 and 1 is defined to discount the rewards in each steps. The larger discount rate means that the long-term rewards are more important, and on the other hand, the smaller discount rate means that short-term rewards become more important. Also, the rewards will be discounted by γ to the exponent of time step. The reason for this is that the future reward is less likely happen, so the agent cannot really trust that. Adding all these to equation 3.1.2, the discounted expected cumulative reward can be written as:

$$R(\tau) = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots \tag{3.1.3}$$

which is equivalent to:

$$R(\tau) = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{3.1.4}$$

While defining goals through reward maximization seems straightforward, it introduces a fundamental challenge: the exploration-exploitation dilemma. This dilemma arises because an agent must navigate between two seemingly contradictory strategies:

- **Exploitation:** Leveraging its current knowledge to choose actions that maximize the expected reward based on past experiences.
- **Exploration:** Trying new, potentially suboptimal actions to gather information

about the environment and discover potentially better options in the long run.

Striking a balance between these two approaches is crucial for successful learning. Imagine an agent playing a game. If it only exploits (chooses familiar actions), it might miss a more rewarding path it hasn't explored yet. Conversely, if it only explores (tries random actions), it might waste time learning about ineffective approaches. A common approach to address the exploration-exploitation dilemma is the ϵ-greedy strategy, which will be explained in more detail in the Q-Learning section. This strategy essentially creates a dynamic balance between exploitation and exploration throughout the training process. It starts with an initial value of ϵ set to 1. In simpler terms, this means there's a 100% chance of exploration at the beginning. As training progresses, the value of ϵ is gradually decreased. This incentives the agent to exploit its knowledge more often (choose actions with a higher expected reward) as it accumulates information about the environment. Therefore, in the initial stages, the agent focuses on exploration to gather information about various possibilities. Later stages prioritize exploiting its knowledge to maximize rewards. The ϵ-greedy strategy offers a simple yet effective way to balance exploration and exploitation, allowing the agent to learn efficiently and adapt its behavior over time.

One of the other remaining terms that needs to be defined from the RL process is state/observation. A state s is a full description of the environment which includes all the information, and there is nothing hidden from the state. On the other side, an observation o is a partial description of the environment that can omit some information. When the agent is able to observe all the information from the environment it's called fully observed. On the other side, when the agent has limited access to the information it's called partial observed. Most of the times, the environment is fully observed, so that the state and observation are equal.

Understanding the environment is crucial for reinforcement learning (RL) agents. Two key concepts define an agent's perception of its surroundings: state and observation. A state s represents a complete snapshot of the environment at a particular point in time. It encompasses all relevant information, leaving nothing hidden from the agent's perspective. In simpler terms, the state is the "big picture" – everything the agent needs to know to make informed decisions. In contrast, an observation o is a partial view of the environment. It might not capture all the details, potentially omitting some information. Imagine an agent navigating a maze. The state might include the position of the agent, all walls, and the location of the exit. An observation, however, could be limited to the immediate surroundings of the agent, like the presence of walls in its north, south, east, and west directions. The level of observability can vary:

- **Fully Observed:** When the agent has access to all information about the environment, the state and observation are equivalent. This simplifies learning as the agent

has a complete picture of the situation.

- **Partially Observed:** In most real-world scenarios, environments are partially observed. The agent has limited access to information, requiring it to learn and make decisions based on incomplete data. This adds complexity to the learning process.

Understanding the distinction between state and observation is essential for RL. The state represents the complete picture, while the observation might be a more limited view. The level of observability can impact the learning process, with fully observed environments offering a simpler learning experience.

Another key component of the RL process is the agent's decision or action. The set of all possible actions an agent can take within a specific environment is called the action space. This space can be either discrete or continuous. Understanding the nature of the action space is crucial for selecting the appropriate RL algorithm to achieve the desired goal. Some algorithms, like Q-Learning, are well-suited for discrete action spaces where the agent can choose from a finite set of distinct actions. For instance, imagine an agent navigating a four-way intersection. Its action space might be "move north," "move south," "move east," or "move west." On the other hand, some algorithms, like Soft Actor-Critic (SAC), are designed for continuous action spaces. Here, the agent can select any action from a continuous range of possibilities. A robot arm manipulating objects might have a continuous action space for joint angles, allowing for precise and fluid movements.

Reinforcement learning (RL) tasks can be categorized as either episodic or continuing. This distinction relates to the structure of the environment and the agent's experience within it. In episodic tasks, there is a well-defined beginning and end. The environment has a terminal state, which signifies the completion of the episode. Imagine an agent playing a game with a clear objective, like reaching the finish line in a maze. Once the agent reaches the terminal state, the episode ends, and a new one might begin with the reset of the environment. In contrast, continuing tasks lack a predefined starting or ending point. The environment persists indefinitely, and the agent interacts with it continuously. The goal is not to reach a specific terminal state but to learn a policy that maximizes long-term rewards. For example, a robot operating in a dynamic environment like a home might continuously learn optimal ways to navigate and interact with its surroundings.

The terms reinforcement learning (RL) and deep reinforcement learning (DRL) are sometimes confused. While RL employs various methods for value estimation, DRL specifically leverages deep neural networks for this purpose. Traditional RL algorithms might use simpler techniques like tables or functions to approximate value. In simpler terms, DRL substitutes tables or functions with deep neural networks, which are powerful tools for learning complex patterns from data. This allows DRL agents to handle problems with

high-dimensional state spaces, where traditional methods struggle due to the sheer volume of information.

Having established the core concepts of RL, a crucial question emerges: how does an agent learn to take actions that maximize its long-term reward (return)? Two primary approaches address this challenge: policy-based and value-based methods. Before delving into these approaches, it's essential to define the concept of policy π. The policy essentially acts as the agent's "brain." It's a function that maps the current state of the environment (perceived information) to a corresponding action for the agent to take. The fundamental goal of RL is to identify an optimal policy π^*, one that maximizes the expected reward the agent receives over time. This optimal policy is not readily available and needs to be learned through an iterative training process. Policy-based and value-based methods offer distinct strategies to achieve this goal.

Policy-based approaches directly train the agent to select the best action for a given state. This method essentially teaches the agent a mapping from each state to the optimal action in that state. The policy π can be either deterministic or stochastic:

- **Deterministic Policy:** In a deterministic policy, the agent always selects the same action for a given state. This approach offers clear decision-making but might not account for uncertainty in the environment.
- **Stochastic Policy:** A stochastic policy utilizes a probability distribution to select actions for each state. This allows the agent to explore different options and potentially discover better strategies over time.

The mathematical representation for these two policy types is presented in equation 3.1.5 for reference.

$$a = \pi(s) \tag{3.1.5}$$

$$\pi(a|s) = P[A|s] \tag{3.1.6}$$

Value-based approaches take an indirect approach to training the agent. Instead of directly learning a policy that maps states to actions, these methods focus on learning a value function. This function estimates the expected long-term reward (return) the agent can expect from being in a particular state. The value of a state is essentially the total discounted reward the agent anticipates receiving if it starts in that state and follows the optimal policy thereafter. In simpler terms, the value function helps the agent identify states that are more promising in terms of future rewards. Mathematically, the value function can be represented as:

$$V^{\pi}(s) = \mathop{\mathbb{E}}_{\tau \sim \pi}[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | S_t = s] \tag{3.1.7}$$

The value function defined earlier is known as the on-policy value function. It estimates the expected discounted return the agent can achieve if it starts in a particular state s and consistently follows the current policy π for making decisions. In essence, it provides a road map based on the agent's current strategy. There exists another important concept: the optimal value function. This function represents the expected discounted return the agent can achieve if it starts in a particular state s and always follows the optimal policy π^*, the one that maximizes long-term rewards. The optimal value function serves as a benchmark, indicating the best possible outcome achievable from any given state.

$$V^*(s) = \max_{\pi} \mathop{\mathbb{E}}_{\tau \sim \pi} [\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | S_t = s] \tag{3.1.8}$$

In this context, the agent starts in state s and acts according to the optimal policy within the environment. This optimal policy represents the ideal course of action that maximizes the agent's long-term rewards.

This section has covered the fundamental terms and concepts that underpin RL. These concepts provide the building blocks for understanding RL algorithms, which will be explored in the following section.

3.2 Q-Learning

The previous section established the core principles of reinforcement learning (RL) algorithms. We saw that RL falls under the umbrella of machine learning, where the agent learns a policy through interacting with an environment to maximize its expected reward. One approach to tackling the RL problem involves value-based methods. Here, the agent prioritizes actions that lead to states with higher predicted value. Once the value function is determined (the process for finding this function will be explained shortly), the agent can identify states with greater value and take actions that move it in that direction.

Drawing upon these concepts and expanding on this foundation, this section will delve into the details of the first RL algorithm we will explore: Q-Learning, first introduced by Watkins in a PhD thesis [61], was further developed in 1992 [62].

As discussed earlier, value-based approaches involve training the agent to learn a value function. This function estimates the expected discounted reward the agent can obtain if it starts in a particular state and follows a specific policy. The policy itself can be any function that maps states to actions. A common policy choice for value-based methods is the $\epsilon-$Greedy policy, which prioritizes actions with the highest predicted value. It balances exploration (trying new actions) and exploitation (focusing on actions known to be good) using an epsilon ϵ value. This is in contrast to policy-based methods, where the agent

directly learns the optimal policy π^* through training. In value-based methods, we define the policy, and the agent learns the optimal value function.

Within value-based methods, there are two key function types: state-value functions and action-value functions. State-Value Function (refer to Eq. 3.1.7): This function predicts the expected return for each state in the environment, assuming the agent starts in that state and follows the policy indefinitely. Action-Value Function (Q-Function): This function estimates the expected return for a specific state-action pair. In other words, it predicts the expected return if the agent starts in a particular state, takes a specific action, and then follows the policy thereafter. The action-value function, denoted by $Q(s, a)$, can be mathematically defined as:

$$Q^{\pi}(s, a) = \mathop{\mathbb{E}}_{\tau \sim \pi}[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_0 = s, a_0 = a] \tag{3.2.1}$$

Q-Learning takes its name from the concept of "quality," reflecting the value of taking a specific action within a particular state. Here's a crucial distinction: a state-value function represents the expected discounted reward achievable from a state, considering the policy being followed. In contrast, a Q-value ($Q(s, a)$) represents the expected discounted reward achievable from a specific state (s) if the agent takes a specific action (a) and then follows the policy thereafter.

A critical question arises: how can we determine these state values or Q-values effectively? A simple approach might involve calculating the sum of all possible rewards the agent can obtain starting from a particular state. However, this method can be computationally expensive and time-consuming, especially for complex environments.

The Bellman equation provides a more efficient solution for estimating state values. This equation will be explored in detail in the following section.

The Bellman equation stands as a cornerstone of Q-Learning's efficiency in calculating state values and Q-values. In its absence, determining the return for each state within an environment would necessitate a separate calculation for every single state. This exhaustive approach, essentially evaluating each state in isolation, becomes computationally expensive and impractical for large or complex environments. The Bellman equation offers a powerful alternative through recursion. It defines the value of any state s as the sum of two crucial components:

- Immediate Reward $R(s, a)$: This represents the immediate reward the agent receives upon taking action a in state s.
- Discounted Future Value: This component captures the long-term reward the agent

anticipates from the state that follows $\acute{s}$. The value of this next state is discounted by a factor γ between 0 and 1. This discount reflects the principle of diminishing returns, where future rewards have less influence on the current decision compared to immediate rewards.

The mathematical representation is:

$$V_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma \times V_\pi(S_{t+1})|S_t = s] \tag{3.2.2}$$

In essence, the Bellman equation breaks down the value of a state into two manageable parts: the immediate gratification the agent experiences and the anticipated value of the following state, taking into account the devaluation of future rewards. This elegant approach allows for a more efficient and scalable way to calculate state values and Q-values, which is essential for Q-Learning to function effectively in complex environments.

Before delving into the Q-Learning algorithm, it's crucial to address another key concept related to value function training. As mentioned earlier, both value-based (using a value function) and policy-based approaches involve training a function during the agent's interaction with the environment. This section explores two primary strategies for training this function: Monte Carlo and temporal-difference learning.

The Monte Carlo learning strategy leverages the entire experience gained throughout an episode to train the value function. In simpler terms, the agent waits until the end of an episode, which could be reaching a goal or encountering a terminal state, before updating the value function. This approach relies on the complete episode's return (denoted as G_t) to estimate the true value of each state visited during the episode. The update rule for the value function in Monte Carlo learning can be represented by the following equation:

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)] \tag{3.2.3}$$

Here, α represents the learning rate, which controls the weight given to newly acquired information compared to previously learned values. G_t represents the return, which is the total discounted reward accumulated throughout the episode. Through this iterative process, the value function is gradually refined based on the experiences gathered across multiple episodes.

Temporal-difference learning, in contrast, adopts a more incremental approach. It aims to learn and update the value function at each time step (t) within an episode. However, unlike Monte Carlo learning, the full episode return (G_t) is not readily available since the episode might not be finished yet. Therefore, temporal-difference learning employs a technique called bootstrapping to estimate the return. Bootstrapping involves using the immediate reward received at the current time step ($R_(t + 1)$) along with the discounted

value of the next state ($V(S_(t+1))$) to estimate the return. This estimated return then serves as the target for updating the value function. This approach allows the agent to learn continuously throughout an episode without waiting for its completion. The update rule for the value function in Temporal-difference learning can be represented by the following equation:

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] \tag{3.2.4}$$

The key distinction between these methods lies in the timing of updates:

- **Monte Carlo:** Updates happen only at the end of an episode.
- **Temporal-Difference:** Updates happen after each step within an episode.

Monte Carlo learning offers the benefit of learning from complete episodes, potentially leading to more accurate value estimates. However, it can be slow for complex environments with long episodes. Temporal-difference learning provides the advantage of continuous learning and faster adaptation, making it suitable for dynamic environments where immediate updates are valuable.

The choice between these methods depends on the specific characteristics of the environment and the desired learning speed.

Another important concept in RL that algorithms are categorized based on that is on-policy and off-policy algorithms. In reinforcement learning, algorithms can be categorized based on how they utilize the data collected during training. On-policy algorithms directly update the policy the agent is currently using to make decisions. They learn from the agent's experience as it interacts with the environment, essentially refining its strategy based on its own actions. In contrast, off-policy algorithms can learn from a broader range of experiences, even those generated by a different policy. This allows them to potentially learn an optimal policy even if the agent's current behavior is not ideal for exploration. The choice between on-policy and off-policy algorithms depends on the specific learning goals and the characteristics of the environment.

With the necessary background information on Q-Learning now provided, we can delve into an explanation of the algorithm itself. Q-Learning represents an off-policy value-based method employed to train its action-value function through Temporal Difference (TD) techniques. Within the realm of Q-Learning, the very algorithm utilized for training the Q-function is Q-Learning itself. The Q-Function, in this context, assumes the form of a table wherein each cell corresponds to the value of a state-action pair. Such tabular representations are aptly suited when both the action space and observation space are discretely defined. However, in scenarios where these spaces become continuous or possess a multitude of dimensions, a more intricate function, or possibly a neural network, may be warranted.

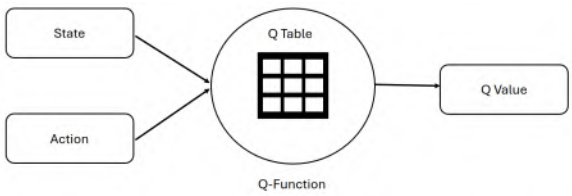


Figure 3.2: Q-Function Structure

The structural layout of the Q-Function is visually depicted in Figure 3.2. As depicted in this figure, the Q-Function serves as a mapping tool, employing either a function or a table to determine the Q-Value associated with a given state-action pair. This function or table encapsulates the learned values corresponding to various state-action combinations.

The pseudo code for Q-Learning algorithm is as follows:

Algorithm 1 Q-Learning

Input: policy π, positive integer *num_episodes*, small positive fraction α, GLIE ϵ_i
Output: value function Q ($\approx q$ if num_episodes is large enough)
Initialize: Q arbitrarily (e.g., $Q(s,a) = 0$ for all $s \in S$ and $a \in A(s)$, and $Q(terminal - state, \cdot) = 0$)

1: **for** $i \leftarrow 1$ to *num_episodes* **do**
2: $\quad \epsilon \leftarrow \epsilon_i$
3: $\quad$ Observe S_0
4: $\quad t \leftarrow 0$
5: $\quad$ **while** S_t is not terminal **do**
6: $\quad\quad$ Choose action A_t using policy derived from Q (e.g., $\epsilon - greedy$)
7: $\quad\quad$ Take action A_t and observe R_{t+1}, S_{t+1}
8: $\quad\quad$ $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma max_a Q(S_{t+1}, a) - Q(S_t, A_t))$
9: $\quad\quad$ $t \leftarrow t + 1$
10: $\quad$ **end while**
11: **end for**
return Q

In the initial stage, the Q-table is initialized either with random values or zeros. Subsequently, an action is selected based on the adopted policy. In the case of the ϵ-Greedy policy, during the initial stages of training, there is a higher probability of choosing exploratory actions, allowing the agents to explore the environment. As the training progresses, the value of ϵ decreases, leading to a reduction in the probability of selecting exploratory actions. Following this, the agent executes action A_t and receives the new reward R_{t+1}, transitioning to the new state S_{t+1}. Subsequently, the Q-Function is updated utilizing the equation specified in line 8 of Algorithm 1 with this experiential data.

3.3 Deep Q-Network

In the previous section, the Q-table served as a means to estimate the Q-value for a given state-action pair. While tables are effective in simple environments with a limited number of states, they prove inadequate for complex environments characterized by a high state count. Consider, for example, an observation comprising a black-and-white image with 256 pixels, resulting in $256 \times 256 = 2^{16}$ potential states. To mitigate this challenge, more robust mapping tools, such as deep neural networks, are preferable over simple tables. Deep neural networks excel as function estimators and are adept at handling complex environments with a multitude of states.

As depicted in Figure 3.2, the Q-function is traditionally represented by a Q-table, where each cell corresponds to the quality of a state-action pair. In contrast, the Deep Q-Network (DQN) employs a deep neural network in lieu of a table, facilitating scalability in Q-Learning. DQN was first introduced in 2015[63]. Consequently, the structure of the DQN can be visualized as follows: In Deep Q-Networks (DQN), a parameterized function

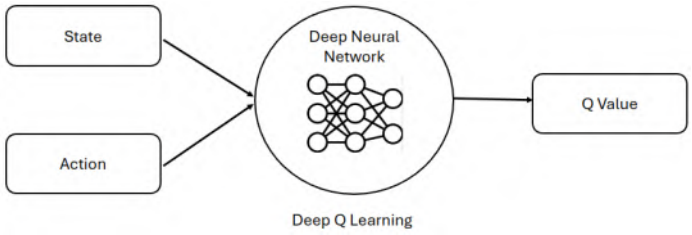


Figure 3.3: DQN Structure

$Q_\theta(s, a)$ is utilized, aiming to optimize the parameters of this function to maximize the expected discounted reward. Unlike Q-Learning, the training process for DQN diverges. While Q-Learning updates the Q-table incrementally with each new experience, DQN employs a loss function that compares the predicted Q-value with the Q-target. Subsequently, gradient descent is utilized to adjust the weights of the deep neural networks, enhancing the approximation of the Q-values.

The Q-target is computed as the sum of the immediate reward and the discounted estimate of the optimal Q-value for the subsequent state. The loss, on the other hand, is defined as the difference between the previously estimated Q-value and the Q-target. Mathematically, this relationship can be expressed as:

$$y_i = r_j + \gamma max_{\acute{a}} \hat{Q}(\phi_{j+1}, \acute{a}; \theta^-) \tag{3.3.1}$$

$$y_i - Q(\phi_j, a_j; \theta) \tag{3.3.2}$$

In equations 3.3.1 and 3.3.2, y_i represents the Q-target, r_i signifies the immediate reward, $\hat{Q}$ denotes the target action-value function, ϕ_{j+1} and $\acute{a}$ denote the state and action at step

$j+1$, respectively, and θ^- stands for the weight parameter for the target network. Equation 3.3.1 outlines the target, while equation 3.3.2 delineates the loss or error between the actual value and the estimated value.

The learning process comprises two phases:

- **Sampling:** Actions are executed by the agent, and the observed experience data is stored in a replay memory.
- **Training:** A subset of this replay memory, referred to as a batch, is randomly selected, and learning takes place using gradient descent.

This learning process can lead to instability due to the combination of a neural network and data from estimations. As mentioned in the previous section, bootstrapping is employed to estimate the return using the Bellman equation. To mitigate this instability, two main solutions are implemented. The first solution involves using experience replay, which enhances sample efficiency. This is achieved by employing a replay buffer to store experience samples that can be utilized later during training. The second solution aims to stabilize the training by employing fixed Q-targets. As depicted in Equation 3.3.2, the loss or TD error is the disparity between the current Q-value and the Q-target. However, when the Q-target is computed using the Bellman equation, as shown in Equation 3.3.1, it represents an estimation rather than the actual value. This estimation is calculated using the same parameters as the current Q-value, meaning that the loss is the discrepancy between two Q-values determined using the same parameters. Consequently, during each training step, both the Q-values (target and current) shift, and as the current value approaches the target, the target value also changes. To tackle this issue, a separate neural network with fixed parameters is employed to estimate the TD target. To update this network, the values from the Deep Q-Network are copied every C steps to update the target network. By implementing these two solutions to enhance the training process, the pseudo code for Deep Q-Learning is as follows:

3.4 Policy-Based Methods

As discussed in the previous sections, the primary objective of reinforcement learning (RL) is to determine an optimal policy π^* that maximizes the return. Policy-based methods represent one approach to achieving this objective, wherein the agent learns the policy directly through interaction with the environment. Similar to the Q-function, the policy is also parameterized, and deep neural networks serve as function estimators. This parameter, typically denoted as θ, aims to be optimized to attain the best values for θ using gradient ascent to maximize the return. This section delves deeper into policy-based methods, with a focus on policy-gradient methods, which constitute a subclass of policy-based methods

Algorithm 2 Deep Q-Learning

Initialize reply memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with wights $\theta^- = \theta$

for episode$= 1, M$ **do**
  Initialize sequence $s_1 = x_1$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = argmax_a Q(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    Set

$$y_i = \begin{cases} r_j, & \text{if episode terminates at step } j+1. \\ r_j + \gamma max_{\acute{a}} \hat{Q}(\phi_{j+1}, \acute{a}; \theta^-), & \text{otherwise} \end{cases}$$

    Perform a gradient descent step on

$$(y_i - Q(\phi_j, a_j; \theta))^2$$

    with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  end for
end for

and serve as the foundation for subsequent algorithms. In policy-gradient methods, the parameter θ is optimized directly, unlike in other policy-based methods where it is optimized indirectly through gradient ascent on the objective function. This section first explains the policy-gradient method and then compares it to other methods.

The overarching aim of the policy-gradient method is to optimize parameters θ in a manner that enhances the overall return. These parameters need to be adjusted such that the return is maximized. Initially, the agent engages with the environment, gathering data over an episode according to the initial policy. Subsequently, the return is computed, and if it is positive, the likelihood of that trajectory occurring is augmented; conversely, if it is negative, the likelihood is diminished. While this method appears effective, a pertinent question arises: how does the agent assess the return or determine its quality? To address this query, an objective function $J(\theta)$ is formulated. This function serves to evaluate the agent's performance for a given trajectory, yielding the expected cumulative reward. As stochastic policies are more prevalent, we shall employ one as an example for the remainder of this section. The objective function for a stochastic policy can be expressed mathemati-

cally as follows:

$$J(\theta) = \sum_{\tau} P(\tau;\theta)R(\tau) \tag{3.4.1}$$

Equation 3.4.1 outlines the expected return $J(\theta)$ by summing over all trajectories. It involves the probability of selecting trajectory τ multiplied by the return of said trajectory. Here, $R(\tau)$ represents the cumulative return from the trajectory, while $P(\tau;\theta)$ signifies the probability of selecting trajectory τ with weights θ. This probability is calculated as follows:

$$P(\tau;\theta) = [\prod_{t=0} P(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)] \tag{3.4.2}$$

As elucidated in this section, policy-based methods offer an alternative approach to solving the RL problem, differing from value-based methods. The advantages of policy-gradient methods over value-based methods are as follows: Policy-based methods have the capability to learn stochastic policies, allowing for the acquisition of complex tasks, whereas value-based methods are limited to learning deterministic policies. Additionally, as stochastic policies output a probability distribution over actions, agents can explore the environment without repeating the same trajectory, thereby eliminating the need for implementing the exploration/exploitation trade-off. Moreover, policy-based methods tend to be more effective in high-dimensional action spaces.

Conversely, policy-based methods entail certain disadvantages compared to value-based methods: These methods often become trapped in local optima instead of reaching the global optimum, and their training process occurs step by step, leading to longer training times. Typically, policy-based methods are not utilized independently in RL; rather, they are combined with Q-functions to form a new category of algorithms known as actor-critic methods, which will be elucidated in the subsequent section.

3.5 Actor-Critic Methods

As discussed in the previous section, in policy-gradient methods, the probability of an action is adjusted based on the quality of that action. This approach can lead to a phenomenon known as reinforce variance, where the same starting state can result in different returns. To mitigate this issue, one immediate solution is to use a larger number of trajectories, effectively increasing the batch size. However, increasing the batch size significantly reduces sample efficiency. This trade-off between variance reduction and sample efficiency presents a challenge in training policy-gradient methods effectively.

The solution to address reinforce variance is to combine policy-based and value-based methods, resulting in what is known as actor-critic methods. The concept of actor-critic was first introduced in 1983 by Barto et al[64]. The core idea of the actor-critic method is to

simultaneously learn and receive feedback. Instead of relying on a single policy or a single value function, it uses a combination of both. There is a policy component, called the actor, which dictates the agent's actions. Additionally, there is a value function component, called the critic, which provides feedback to the actor by evaluating the value of the taken actions. This dual structure helps in stabilizing the learning process and improving performance. The framework for this method of learning is illustrated in Figure 3.4. As shown in the figure,

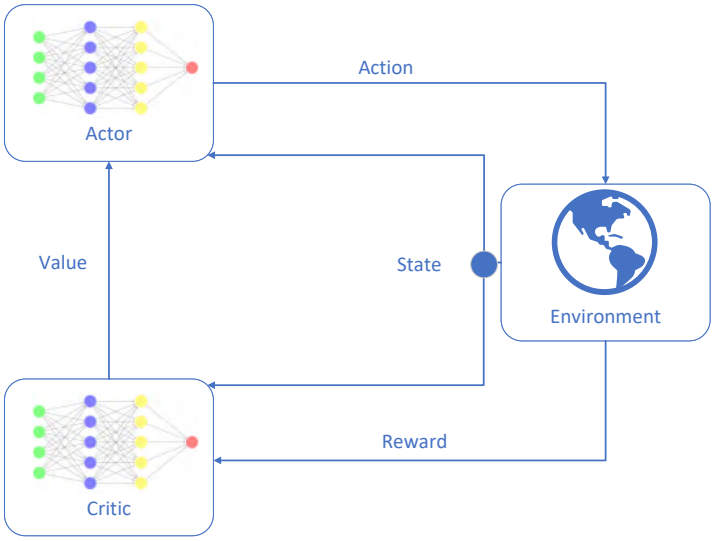


Figure 3.4: Actor-Critic Framework

there are two functions (neural networks): the first is a policy function $\pi_\theta(s)$ parameterized by θ, and the second is a Q-function $Q_w(s, a)$ parameterized by w. At each time step, the agent receives the state S_t from the environment, which is then fed to both the actor and critic networks. The actor network determines an action A_t based on the received state S_t. Simultaneously, the critic network receives the action and the state, and computes the Q-value $Q(S_t, A_t)$ for that action-state pair. The action chosen by the policy is executed, leading the environment to transition to a new state S_{t+1} and yield a reward R_{t+1}. Using the new reward and Q-value, the actor and critic networks are subsequently updated to improve their performance.

This method of learning has been further developed over time, leading to the emergence of several effective and innovative algorithms, including DDPG, TD3, and SAC. In the following sections, we will examine these algorithms in detail.

3.6 Deep Deterministic Policy Gradient

As discussed in the previous section, the combination of policy-gradient methods and value-based methods creates a new family of algorithms called actor-critic based algorithms.

The first actor-critic algorithm to be explained is Deep Deterministic Policy Gradient (DDPG). The deterministic policy gradient algorithm was first introduced in 2014 by Silver et al. [65], and the DDPG algorithm was developed in 2015 by Timothy et al. [66]. This algorithm is an off-policy deep reinforcement learning algorithm that uses the Bellman equation to learn the Q-function while simultaneously using this Q-function to learn the policy.

The main motivation behind DDPG was to develop the DQN algorithm in a way that it could be used for environments with continuous action spaces. As discussed before, if the optimal action-value function $Q^*(s, a)$ is known, then for any given state s, the optimal action $a^*(s)$ can be found by solving the following equation:

$$a^*(s) = arg\,\max_a Q^*(s, a) \tag{3.6.1}$$

The main difference between DDPG and DQN is that in DQN, finding the action that maximizes the Q-value is a straightforward task because there are a limited number of states. By calculating the Q-value for each state, the maximum can be determined. However, when the action space is continuous, finding the maximum in the same way as DQN is not feasible. Additionally, using optimization algorithms to find the maximum is time-consuming and computationally expensive. Instead of calculating the Q-value for all states or running an optimization process, DDPG approximates the optimal action that leads to the optimal Q-function $max_a\, Q^*(s, a)$ with $max_a\, Q^*(s, a) \approx Q(s, \pi(s))$. Therefore, a policy is trained that takes a state s as input and outputs an optimal action a^* that leads to the optimal Q-function $Q^*(s, a)$. In the following parts of this section, the mathematical equations and pseudocode of this algorithm will be covered in detail.

The Q-learning aspect of the DDPG algorithm employs the mean-squared Bellman error (MSBE) to train the Q-function. The MSBE measures how closely the Q-function approximates the Bellman equation. The Q-function approximator, denoted as $Q_\phi(s, a)$, is represented by a neural network with parameters ϕ and utilizes a set of experiences $\mathbb{D}$ collected during the episode. Using this information, the MSBE equation can be written as:

$$L(\phi, \mathbb{D}) = \underset{(s_t, a_t, r, s_{t+1}, d) \sim \mathbb{D}}{E} [(Q_\phi(s, a) - (r + \gamma(1 - d)\, \underset{a_{t+1}}{max}) Q_\phi(s_{t+1}, a_{t+1}))^2] \tag{3.6.2}$$

In the above equation, $L(\phi, \mathbb{D})$ represents the MSBE, which is a function of ϕ and $\mathbb{D}$, and d is a binary value indicating whether the terminal state has been reached. This equation is not used in isolation to learn the Q-function; two main techniques are employed: using replay buffers and target networks. Utilizing an experience replay buffer allows the algorithm to maintain stable learning behavior. Enlarging the replay buffer helps avoid overfitting because relying solely on the most recent data can lead to overfitting of the Q-functions.

Conversely, expanding the replay experience too much can slow down the learning process due to the increased volume of data to be processed. In the training process, this trade-off can be managed by adjusting a parameter known as the batch size.

The second technique is the use of target networks, as discussed in the context of DQN. In DQN, the loss equation is defined as the difference between the target value and the current value. If the same parameters are used for both the current value and the target value, the learning process can become unstable. To address this issue in DQN, the target network is updated with a time delay. However, the difference between the DQN and DDPG algorithms lies in how the target network is updated. In DQN, the main parameters of the value function are simply copied to the target network at intervals. In contrast, DDPG updates the target network using the Polyak averaging equation:

$$\phi_{targ} \leftarrow \rho\phi_{targ} + (1-\rho)\phi \tag{3.6.3}$$

In this equation, ρ is a hyper-parameter known as the Polyak coefficient, which lies between 0 and 1. The Polyak coefficient controls the update rate of the target network, allowing for a smoother and more stable convergence by slowly incorporating the new parameters into the target network. This gradual update helps to mitigate the instability that can arise from using the same parameters for both the current value and the target value, thus enhancing the overall performance and reliability of the learning algorithm.

The policy learning aspect of DDPG is straightforward compared to the Q-value side. The objective is to learn a deterministic policy $\pi_\theta(s)$ that identifies the action maximizing $Q_\phi(s,a)$. This goal is achieved through gradient ascent with respect to the policy parameters, treating the Q-function parameters as constants. The gradient ascent ensures that the policy improves by adjusting its parameters to increase the expected return. The mathematical formulation for this update process is given by:

$$\underset{\theta}{max} \underset{s\sim\mathbb{D}}{\mathbb{E}}[Q_\phi(s,\pi_\theta(s))] \tag{3.6.4}$$

Combining all these equations together, the pseudocode for DDPG is as follows:

3.7 Twin Delayed DDPG

Although DDPG is one of the successful algorithms, particularly in robotic tasks, a common issue is that the learned Q-function tends to overestimate Q-values. In other words, the error in the Q-function is exploited, resulting in low error values but oscillating Q-values. Twin Delayed DDPG (TD3), developed by Fujimoto et al. in 2018[67], addresses this problem by implementing three methods simultaneously .

Algorithm 3 DDPG algorithm[66]

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\pi(s|\theta^\pi)$ with weights θ^Q and θ^π
Initialize target network Q' and π' with weights $\theta^{Q'} \leftarrow \theta^Q$,$\theta^{\pi'} \leftarrow \theta^\pi$
Initialize reply Buffer $\mathbb{D}$

for episode $= 1, M$ **do**
  Initialize a random process $\mathbb{N}$ for action exploration
  Receive initial observation state s_1
  for $t = 1, T$ **do**
    Select action $a_t = \pi(s|\theta^\pi) + \mathbb{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in $\mathbb{D}$
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from $\mathbb{D}$
    Set

$$y_i = r_i + \gamma Q'(s_{i+1}, \pi'(s_{i+1}|\theta^{\pi'})|\theta^{Q'})$$

    Update critic by minimizing the loss:

$$L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$$

    Update the actor policy using the policy gradient:

$$\nabla_{\theta^\pi} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\pi(s_i)} \nabla_{\theta^\pi} \pi(s|\theta^\pi)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \rho\theta^Q + (1-\rho)\theta^{Q'}$$
$$\theta^{\pi'} \leftarrow \rho\theta^\pi + (1-\rho)\theta^{\pi'}$$

  end for
end for

The first method is using clipped double-Q learning, which involves learning two Q-functions instead of one. The smaller value of the two Q-values is used to form the target in the MSBE equation. The second method involves updating the policy network with a delay compared to the Q-function network. The third method is adding noise to the target action to make it more challenging for the actor network to exploit the critic network. The target policy is smoothed by adding clipped noise to each dimension of the action, and then the target action is clipped to lie within the valid action range. This target action can be mathematically represented as:

$$a'(s') = \text{clip}(\pi_{\theta_{targ}}(s') + clip(\epsilon, -c, c), a_{low}, a_{high}), \epsilon \sim \mathbb{N}(0, \sigma) \tag{3.7.1}$$

where ϵ is the added noise, σ is the standard deviation of the noise, and c is the clipping value. This ensures that the noise is within a certain range, preventing large deviations and

maintaining stability in the learning process.

The Q-functions use the same target, which is calculated with the smaller target value using this equation:

$$y(r, s', d) = r + \gamma(1 - d) \min_{i=1,2} Q_{\phi_{i,targ}}(s', a'(s')) \tag{3.7.2}$$

After this, both Q-functions are learned by regressing to this target:

$$L(\phi_1, \mathbb{D}) = \underset{(s,a,r,s',d)\sim\mathbb{D}}{E}[(Q_{\phi_1}(s, a) - y(r, s', d))^2] \tag{3.7.3}$$

$$L(\phi_2, \mathbb{D}) = \underset{(s,a,r,s',d)\sim\mathbb{D}}{E}[(Q_{\phi_2}(s, a) - y(r, s', d))^2] \tag{3.7.4}$$

This ensures that the Q-functions are updated more conservatively, reducing the risk of overestimation. By taking the minimum of the two Q-values, the algorithm effectively mitigates the problem of over-optimistic value estimates, leading to more stable and reliable learning.

The actor learning is the same as in DDPG, as explained in equation 3.6.4. Combining all these equations and methods together, the TD3 algorithm can be written as follows:

Algorithm 4 TD3 algorithm[67]

Initialize critic networks Q_{θ_1}, Q_{θ_2}, and actor network π_ϕ with random parameters θ_1, θ_2, ϕ
Initialize target networks $\theta'_1 \leftarrow \theta_1, \theta'_2 \leftarrow \theta_2, \phi' \leftarrow \phi$
Initialize reply buffer $\mathbb{D}$

for $t = 1, T$ **do**
  Select action with exploration noise $a \sim \pi_\phi(s) + \epsilon, \epsilon \sim \mathbb{N}(0, \sigma)$ and observe reward r and new state s'
  Store transition tuple (s, a, r, s') in $\mathbb{D}$
  Sample mini-batch of N transitions (s, a, r, s') from $\mathbb{D}$
  $\tilde{a} \leftarrow \pi_{\phi'}(s') + \epsilon, \qquad \epsilon \sim clip(\mathbb{N}(0, \tilde{\sigma}), -c, c)$
  $y \leftarrow r + \gamma min_{i=1,2} Q_{\theta'_i}(s', \tilde{a})$
  Update critics $\theta_i \leftarrow argmin_{\theta_i} N^{-1} \sum (y - Q_{\theta^i}(s, a))^2$
  if t mod d **then**
    Update ϕ by the deterministic policy gradient:

$$\nabla_\phi J(\phi) = N^{-1} \sum \nabla_a Q_{\theta_1}(s, a)|_{a=\pi_\phi(s)} \nabla_\phi \pi_\phi(s)$$

    Update target networks:

$$\theta'_i \leftarrow \rho\theta_i + (1 - \rho)\theta'_i$$
$$\phi'_i \leftarrow \rho\phi_i + (1 - \rho)\phi'_i$$

  end if
end for

3.8 Soft Actor-Critic

In the previous two sections, we thoroughly examined two off-policy DRL algorithms based on actor-critic networks: DDPG and TD3. While these algorithms can efficiently learn from past samples using a replay buffer, they are notably sensitive to hyperparameters. This sensitivity necessitates extensive tuning to achieve convergence and obtain satisfactory results. To mitigate this issue, the Soft Actor-Critic (SAC) algorithm was introduced by Haarnoja et al. in 2018 [10].

SAC has two primary features that distinguish it from other actor-critic based algorithms: the use of a stochastic policy instead of a deterministic one, and the objective of maximizing entropy. The use of a stochastic policy in Deep Reinforcement Learning (DRL) encourages exploration by enabling the agent to select actions probabilistically rather than deterministically. This probabilistic action selection allows the agent to explore a broader range of actions and states, helping to avoid local optima and increasing the likelihood of discovering more effective strategies or solutions within the environment.

Moreover, SAC trains the policy to maximize not only the expected return but also the entropy of the policy. Entropy, a measure of randomness in the policy, plays a crucial role in preventing the policy from becoming stuck in local optima. By incorporating entropy maximization, SAC encourages the policy to remain sufficiently exploratory, which enhances its robustness and ability to adapt to various situations.

Overall, the introduction of SAC represents a significant advancement in the field of DRL. Its unique approach to balancing exploration and exploitation through stochastic policies and entropy maximization addresses some of the key challenges faced by previous actor-critic algorithms, leading to more stable and efficient learning.

Entropy is a term that is usually used in other fields like thermodynamics. In machine learning algorithm it's used to indicate how random a variable is. If s is random variable with density function P, the entropy H of x is computed from it's distribution: Entropy is a term commonly used in fields such as thermodynamics, but in the context of machine learning algorithms, it refers to the randomness of a variable. In this context, if s is a random variable with a probability density function P, the entropy H of s is computed from its distribution as follows:

$$H(P) = \underset{x \sim P}{E}[-\log P(x)] \tag{3.8.1}$$

By incorporating entropy into the reinforcement learning problem, the agent receives an additional reward that is proportional to the entropy of the policy. This adjustment encourages the agent to maintain exploration throughout the learning process. Consequently,

the primary equation of reinforcement learning can be modified as follows:

$$\pi^* = arg\,\max_{\pi} \underset{\rho\sim\pi}{E}[\sum_{t=0}^{\infty} \gamma^t (R(s_t, a_t, s_{t+1}) + \alpha H(\pi(.|s_t)))] \tag{3.8.2}$$

In this context, α represents the entropy coefficient, which balances the trade-off between increasing entropy and maximizing the return. With this understanding of entropy in place, we can now delve into the primary aspects and equations of the Soft Actor-Critic (SAC) algorithm.

In the SAC algorithm, similar to TD3, a policy π_θ and two Q-functions Q_{ϕ_1}, Q_{ϕ_2} are learned simultaneously. The Q-learning aspect of SAC is largely analogous to TD3 with a few key differences. In SAC, both Q-functions are trained using mean-squared Bellman error (MSBE) minimization by regressing to a single shared target. The target Q-networks are updated using Polyak averaging of the Q-networks during training. The main difference between SAC and TD3 in the Q-learning component is that the target includes the entropy term, and the next state-action used in the target comes from the current policy instead of a target policy. Consequently, the Bellman equation for the entropy-regularized Q^π can be written as:

$$Q^\pi(s,a) = \underset{\substack{s'\sim P \\ a'\sim\pi}}{E}[R(s,a,s') + \gamma(Q^\pi(s',a') - \alpha \log \pi(a'|s'))] \tag{3.8.3}$$

The right-hand side of this equation is an expectation over the next state, which can be approximated using samples. By using sampled data from the replay buffer, the expectation is computed based on actual experiences, which helps in approximating the target Q-value more accurately. This sampling method allows the algorithm to learn from a broader range of experiences, thereby enhancing the stability and performance of the learning process.

$$Q^\pi(s,a) \approx r + \gamma(Q^\pi(s',a') - \alpha \log \pi(a'|s')) \tag{3.8.4}$$

In the SAC algorithm, the MSBE loss for each Q-function is set using sample approximation for the target. Consequently, the loss function for the Q-networks in SAC is formulated as follows:

$$L(\phi_i, \mathbb{D}) = \underset{(s,a,r,s',d)\sim\mathbb{D}}{E}[(Q_{\phi_i}(s,a) - y(r,s',d))^2] \tag{3.8.5}$$

where the target is:

$$y(r,s',d) = r + \gamma(1-d)(\min_{j=1,2} Q_{\phi_{targ,j}}(s',a') - \alpha \log \pi_\theta(a'|s')) \tag{3.8.6}$$

The policy learning aspect of SAC is quite similar to that of TD3, with two key differences: the addition of entropy to the loss function and the use of a squashed Gaussian policy.

This type of policy ensures that the samples are obtained using the following equation:

$$a'_\theta(s,\xi) = \tanh\left(\mu_\theta(s) + \sigma_\theta(s) \odot \xi\right) \tag{3.8.7}$$

By using this kind of policy, the optimization works according to this equation:

$$\max_\theta \underset{\substack{s\sim\mathbb{D}\\ \xi\sim\mathbb{N}}}{E}[\min_{j=1,2} Q_{\phi_j}(s, a'_\theta(s,\xi)) - \alpha \log \pi_\theta(a'_\theta(s,\xi)|s)] \tag{3.8.8}$$

Integrating all the equations and methodologies together, the SAC algorithm can be delineated as follows:

Algorithm 5 SAC algorithm[10]

Initialize parameter vectors θ, ϕ_1, ϕ_2
Set target parameters to main parameters

$$\phi_{targ,1} \leftarrow \phi_1$$
$$\phi_{targ,2} \leftarrow \phi_2$$

while Training **do**
  Observe state s and select action $a \sim \pi_\theta(.|s)$
  Execute a in the environment
  Store (s, a, r, s', d) in reply buffer $\mathbb{D}$
  for j in range update **do**
    Sample a batch of transitions from reply buffer
    Compute targets for Q-networks:

$$y(r,s',d) = r + \gamma(1-d)(\min_{j=1,2} Q_{\phi_{targ,j}}(s',a') - \alpha \log \pi_\theta(a'|s'))$$

    Update Q-network using gradient descent

$$\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s6\prime,d)\in B} (Q_{\phi_i}(s,a) - y(r,s',d))^2$$

    Update policy using:

$$\nabla_\theta \frac{1}{|B|} \sum_{s\in B} (\min_{i=1,2} Q_{\phi_i}(s, a'_\theta(s)) - \alpha \log \pi_\theta(a'_\theta|s))$$

    Update target networks with:

$$\phi_{targ,i} \leftarrow \rho\phi_{targ,i} + (1-\rho)\phi_i$$

  end for
end while

Chapter 4

Methodology and Smart Grasping Framework

As outlined in the first chapter, the primary objective of this thesis is to develop a reinforcement learning model capable of grasping various objects using a three-finger gripper. The central research question guiding this research is:

How can a reinforcement learning-based grasping system be developed to prioritize optimal grasping, and handle novel objects without requiring specific training for each object?

In order to answer this question, three objectives were defined:

- **Objective 1:** Develop a physics-based simulation environment for modeling and evaluating robotic grasping capabilities.
- **Objective 2:** Validate the physics-based simulation model encompassing the robot and gripper through rigorous testing and comparison with real-world performance.
- **Objective 3:** Train a deep reinforcement learning agent within the developed simulation environment for optimizing robotic grasping performance.

In this chapter, a framework is established to achieve these objectives. Each component of the framework is then explained in detail, along with the methods employed. Finally, the contributions of this chapter are discussed.

4.1 Smart Grasping Framework

Before delving into the components and methods, a framework for the smart grasping task is proposed. The primary goal is to grasp unknown objects using a three-finger gripper.

Alongside the gripper, a robot is required to mount the gripper and facilitate its movement. Additionally, a vision sensor is necessary to gather information from the environment. These three components are integrated with a computer that runs the DRL model and connects to all the components. Combining these elements, the smart grasping framework is illustrated in Figure 4.1. The gripper is mounted on a serial robot, and a vision sensor is positioned to

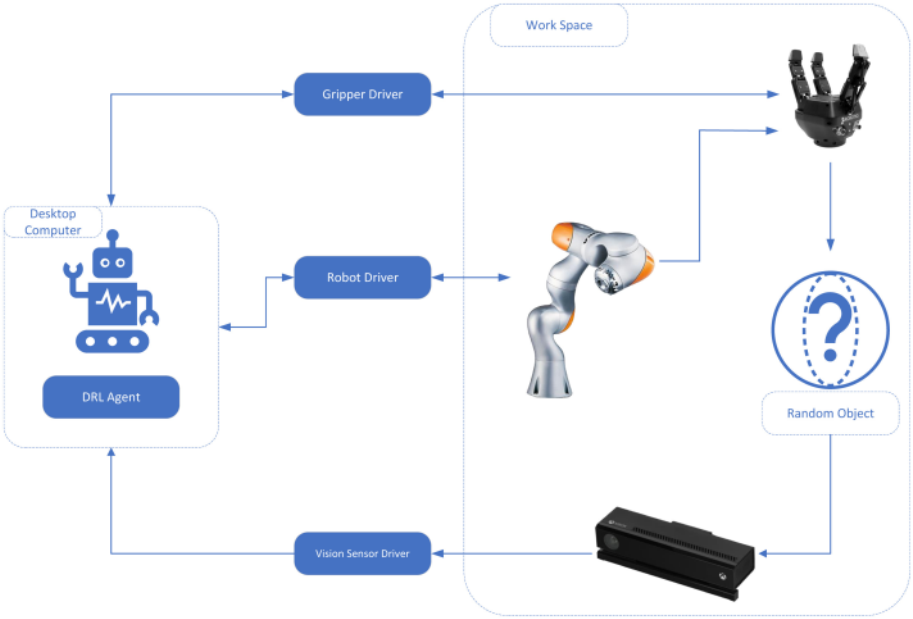


Figure 4.1: Proposed Smart Grasping Framework

define the workspace along with the target objects. The robot, gripper, and vision sensor are connected to their respective drivers, which are all linked to a central computer. This computer receives sensor data from the vision sensor and the robot, processes it through the DRL agent, and then sends commands back to the robot and gripper to perform actions. The vision sensor provides the agent with information about the object, including its position, height, and shape. The data from the robot helps calculate the gripper's position, while data from the gripper indicates contact status. The interface between the DRL agent and the environment uses this information to form the state, which the agent then uses to predict the action. The central computer sends the command to the robot and gripper, guiding the gripper to the desired location, closing it, and grasping the object. This system functions as a closed-loop control system, where controlling the end-effector's pose enables object grasping. The equipment used for each component is listed in Table 4.1.

As discussed in the first chapter, one significant advantage of reinforcement learning (RL) is that the agent can be trained in a simulation environment rather than in the real world. Training in a simulation environment offers several benefits:

- **Speed:** Training in simulation allows for the use of parallel environments, where

Table 4.1: Equipment List

Component	Model	Manufacturer
Manipulator Robot	LBR iiwa 14 R820	*KUKA*
Gripper	3-Finger Adaptive Robot Gripper	*ROBOTIQ*
Vision Sensor	XBOX Kinect Sensor	*Microsoft*
Desktop Computer	Precision Tower 7920	*Dell*

multiple simulated environments run simultaneously to increase the training speed. This technique will be explained in more detail in section 4.7.

- **Equipment Depreciation:** Training an RL model often requires over a million time steps. Running expensive equipment for such a high number of cycles can significantly reduce their effective lifespan. Simulation training avoids this issue by preserving the physical equipment.

- **Safety:** During the initial stages of training, the agent performs random actions to explore the environment. This can result in undesired states for the robot and gripper, posing safety risks to both humans and equipment. Simulation training mitigates these risks by allowing the agent to learn and make mistakes in a controlled, virtual environment.

To address these issues, the training is conducted in a simulation environment. Once acceptable results are obtained, the model can either undergo further training in the real world or be validated. For the proposed grasping system to be simulated effectively, the robot and gripper must be accurately modeled in the simulation environment. This allows for the creation of a simulation framework, illustrated in Figure 4.2.

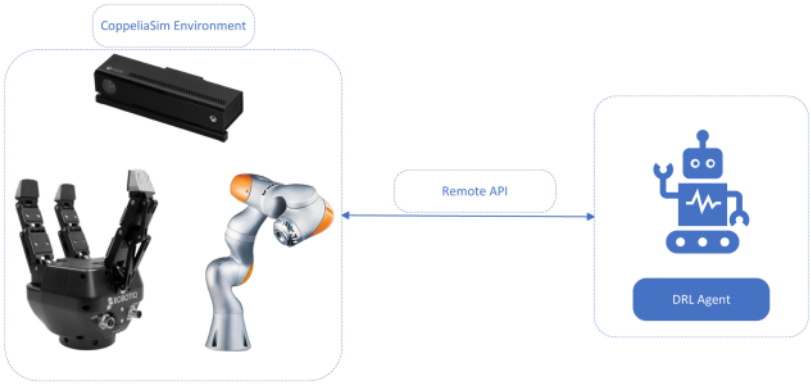


Figure 4.2: Proposed Smart Grasping Framework in Simulation

In this chapter, we will explore all the components depicted in Figures 4.1 and 4.2, along with the methods and techniques employed to achieve the desired functionality.

4.2 Simulation Software

In this section, we delve into the selected simulation software and physics engine, starting with a comprehensive introduction to physics engines.

The first component of the simulation environment is the software used to simulate the environment. To ensure fast and stable learning for the RL agent, it is crucial to use reliable and efficient software. This software will be repeatedly called upon during training, making its performance and stability essential. In the context of smart grasping, physics engines are indispensable due to their ability to simulate collisions accurately. The contact between the gripper and the object must be realistically modeled to achieve practical and reliable results.

Physics engines form the backbone of modern video games, powering simulations across various platforms, from mobile devices to game consoles. These engines use mathematical algorithms to solve physical equations that model real-world dynamics. Two main features of physics engines are real-time physics simulation and collision detection. Real-time physics simulation enables the simulation of physical objects and their interactions as they occur, providing an immediate and dynamic representation of physical phenomena. Collision detection, on the other hand, allows the physics engine to identify and model interactions between objects, ensuring that collisions are accurately represented and resolved.

Several prominent physics engines are widely used in robotics and related fields. These include Bullet, Newton[68], ODE[69], MuJuCo[70], and Isaac Sim[71]. Each of these engines has unique strengths and capabilities, making them suitable for different applications. For instance, Bullet is known for its robust collision detection and rigid body dynamics, while MuJuCo is celebrated for its efficiency in simulating complex, high-dimensional systems.

It is important to note that physics engines are typically accessed through software interfaces that allow users to create and manipulate simulation environments. These interfaces often provide a suite of tools and libraries to facilitate the creation of scenes and the integration of physics simulations. For example, PyBullet[72] is a Python library that offers a user-friendly interface for creating scenes and connecting to the Bullet physics engine. This integration enables users to simulate and visualize the dynamics of their models in real time.

One of the most popular software platforms for creating and managing simulation environments is CoppeliaSim[73]. CoppeliaSim offers a graphical user interface for building

Table 4.2: Robot Specification

Specification	Parameter
Model	LBR iiwa 14 R820
Manufacture	*KUKA*
Maximum Reach	820 mm
Rated Payload	14 kg
Number of axis	7
Weight	30 kg
Controller	KUKA Sunrise Cabinet
Type of axis	Rotational

detailed scenes and includes support for multiple physics engines. This flexibility allows users to choose the most appropriate engine for their specific needs. CoppeliaSim's versatility and ease of use make it a valuable tool for researchers and developers working in robotics and automation.

In this thesis, the simulation environment was constructed using CoppeliaSim. The Bullet physics engine was selected to handle real-time physics simulation due to its proven reliability and performance in simulating rigid body dynamics and collision detection. The scene created in CoppeliaSim includes all necessary components of the smart grasping framework, such as the robot, the three-finger gripper, and the vision sensor. By integrating these components into a cohesive simulation environment, we can accurately model the interactions and dynamics required for training the reinforcement learning agent.

4.3 Kinematic Model Of The Robot

As previously mentioned, the robotic arm utilized in this thesis is a 7 DOF serial robot manufactured by *KUKA*. Serial robots, compared to parallel robots, offer the advantage of a larger workspace, making them ideal for grasping tasks. The specifications of the robot are detailed in Table 4.2. In this section, the methods used to solve the forward kinematics (FK) and inverse kinematics (IK) problems for this robot are explained. Initially, the kinematic model and Denavit-Hartenberg (DH) parameters are extracted. Using this model, the forward kinematic equations are derived. To solve the inverse kinematic problem, the solution proposed by Shimizu et al.[74] is employed and subsequently implemented in Python.

The KUKA LBR iiwa 14 R820 is an anthropomorphic robot with a serial 7-degree-of-freedom non-offset structure. This robot is designed based on the optimal kinematic configuration for a 7 DOF serial manipulator proposed by Hollerbach[75]. Its redundancy is intended to avoid singularities and enhance the robot's agility, but it also complicates the

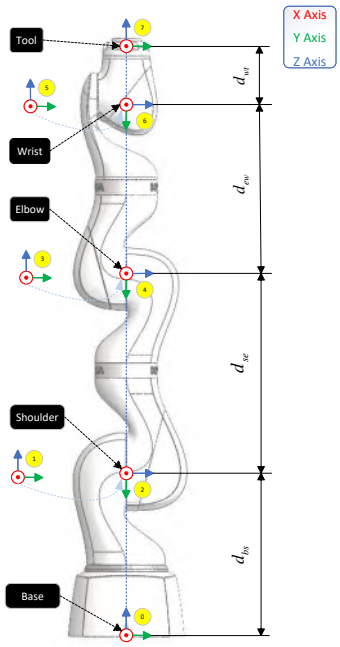


Figure 4.3: Robot Structure

IK problem. Additionally, the IK problem does not always yield a unique solution, often resulting in multiple potential answers for a given pose. To address this issue, additional criteria must be defined to uniquely identify an optimal solution.

We begin by describing the structure of the robot. The DH systematic method is employed in this case. For each axis, a unique coordinate frame is defined, and using the relationship between these frames, the DH table can be constructed. The structure of the robot is illustrated in Figure 4.3. In this figure, all the coordinate frames are displayed along with the link lengths. The intersections of two joints are critical in kinematic modeling; thus, these intersections are designated with names inspired by the human arm: shoulder, elbow, and wrist. Utilizing this structure, the DH parameters can be determined, as illustrated in Table 4.3, with the numerical values for the links as:

$$[d_{base}, d_{se}, d_{ew}, d_{wt}] = [360, 420, 400, 276][mm] \tag{4.3.1}$$

The first parameter is the configuration set (CS), which specifies the set of inverse kinematics solutions. The CS parameter indicates whether the signs of joints 2, 4, and 6 are positive or negative. The selection of this parameter depends on the specific needs of the task and typically does not affect the joint limits or singular points; it is more related to obstacle avoidance. The physical interpretation of CS_2 is whether the shoulder is turned or not, and CS_4 indicates whether the elbow is above or below. The configuration set is

Table 4.3: Denavit-Hartenberg Parameters

i	a_i	$\alpha_i[rad]$	d_i	θ_i
1	0	$-\pi/2$	d_{base}	θ_1
2	0	$\pi/2$	0	θ_2
3	0	$\pi/2$	d_{se}	θ_3
4	0	$-\pi/2$	0	θ_4
5	0	$-\pi/2$	d_{ew}	θ_5
6	0	$\pi/2$	0	θ_6
7	0	0	d_{wt}	θ_7

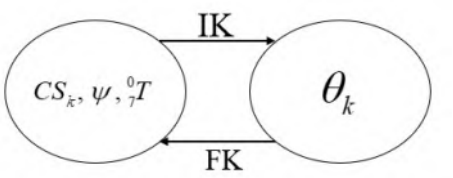


Figure 4.4: Configuration Set, Arm Angle, Pose, Joints relation

given by:

$$CS_k = \begin{cases} 1 & if \quad \theta_k \geq 0 \\ -1 & if \quad \theta_k < 0 \end{cases}, \quad \forall k \in \{2 \;\; 4 \;\; 6\} \tag{4.3.2}$$

The second parameter is the arm angle (ψ), introduced by Hollerbach [75], which determines the rotation of the elbow about the shoulder-to-wrist vector. When the arm angle is zero, it indicates that the third joint is locked in place.

The arm angle ψ can be determined by introducing two planes. The first is the arm plane, defined by the shoulder, elbow, and wrist. The second is the reference plane, which corresponds to the arm plane when the third joint is locked. It is important to note that the configuration set and the arm angle are determined during the forward kinematics process, along with the tool pose. Therefore, when solving the inverse kinematics problem, specific values are assumed for these two parameters alongside the desired pose as a starting point. The relationship between these parameters is illustrated in Figure 4.4.

The forward kinematics problem in this case is relatively straightforward. When the joint angles θ_K are known, other parameters can be directly and uniquely determined. The configuration set can be identified using equation 4.3.2. The end effector's pose, ${}^0_7T$, is determined by multiplying the transformation matrices. For each row in the DH table, the

transformation matrix is given by:

$$
{}^{i-1}_{i}T = \begin{bmatrix} \cos\theta_i & -\sin\theta_i\cos\alpha_i & \sin\theta_i\sin\alpha_i & a_i\cos\theta_i \\ \sin\theta_i & \cos\theta_i\cos\alpha_i & -\cos\theta_i\sin\alpha_i & a_i\sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.3.3}
$$

By constructing these transformation matrices for all frames and multiplying them using the following equation, the tool's pose relative to the base frame can be determined. This approach allows for a comprehensive and accurate calculation of the end effector's position and orientation in the workspace.

$$
{}^{0}_{7}T = {}^{0}_{1}T \times {}^{1}_{2}T \times {}^{2}_{3}T \times {}^{3}_{4}T \times {}^{4}_{5}T \times {}^{5}_{6}T \times {}^{6}_{7}T \tag{4.3.4}
$$

To determine the arm angle, the method proposed in [76] is employed. As outlined in their original paper, this method involves defining specific vectors to facilitate the calculation. The vectors used in this approach are defined as follows:

$$
\begin{aligned}
{}^{0}\mathbf{p}_2 &= \begin{bmatrix} 0 & 0 & d_{bs} \end{bmatrix}^T \\
{}^{2}\mathbf{p}_4 &= \begin{bmatrix} 0 & d_{se} & 0 \end{bmatrix}^T \\
{}^{4}\mathbf{p}_6 &= \begin{bmatrix} 0 & 0 & d_{ew} \end{bmatrix}^T \\
{}^{6}\mathbf{p}_7 &= \begin{bmatrix} 0 & 0 & d_{wt} \end{bmatrix}^T
\end{aligned}
$$

The vector connecting the shoulder to the wrist can then be determined using the following equation:

$$
{}^{2}\mathbf{p}_6 = {}^{0}\mathbf{p}_7 - {}^{0}\mathbf{p}_2 - {}^{0}R_7 \times {}^{6}\mathbf{p}_7 \tag{4.3.5}
$$

The virtual elbow angle, θ_4^v, representing the value of joint 4 when joint 3 is locked, can then be calculated using the following equation:

$$
\theta_4^v = CS_4 \arccos\left(\frac{\|{}^{2}\mathbf{p}_6\|^2 - (d_{se})^2 - (d_{ew})^2}{2d_{se}d_{ew}}\right) \tag{4.3.6}
$$

Subsequently, the virtual joint number one is determined by:

$$
\theta_1^v = \begin{cases} atan2({}^{2}\mathbf{p}_{6,y}, {}^{2}\mathbf{p}_{6,x}) & if \quad \|{}^{2}p_6 \times {}^{0}R_{1,z}\| > 0 \\ 0, & if \quad \|{}^{2}\mathbf{p}_6 \times {}^{0}R_{1,z}\| = 0 \end{cases} \tag{4.3.7}
$$

Then, the virtual shoulder joint, denoted as θ_2^v, can be obtained by:

$$\theta_2^v = atan2(\sqrt{({}^2\mathbf{p}_{6,x})^2 + ({}^2\mathbf{p}_{6,y})^2}, {}^2\mathbf{p}_{6,z}) + CS_4 \ \ \phi \tag{4.3.8}$$

The angle ϕ is computed using the following equation:

$$\phi = \arccos\left(\frac{(d_{se})^2 + \|{}^2\mathbf{p}_6\|^2 - (d_{ew})^2}{2\, d_{se}\|{}^2\mathbf{p}_6\|}\right) \tag{4.3.9}$$

Given that virtual joint number three θ_3^v is zero, and the other joints θ_1^v, θ_2^v, and θ_4^v have been determined, the pose of the virtual elbow relative to the base ${}_4^0T^v$ can be obtained using equations 4.3.3 and 4.3.4. At this point, the positions of the shoulder, elbow, and wrist in both the virtual and actual poses are known. The normal vector of the planes spanned by these three points can be determined using the following equations:

$$\mathbf{v}_{sew}^v = (\frac{{}^0\mathbf{p}_\mathbf{4}^\mathbf{v} - {}^0\mathbf{p}_\mathbf{2}^\mathbf{v}}{\|{}^0\mathbf{p}_\mathbf{4}^\mathbf{v} - {}^0\mathbf{p}_\mathbf{2}^\mathbf{v}\|}) \times (\frac{{}^0\mathbf{p}_\mathbf{6}^\mathbf{v} - {}^0\mathbf{p}_\mathbf{2}^\mathbf{v}}{\|{}^0\mathbf{p}_\mathbf{6}^\mathbf{v} - {}^0\mathbf{p}_\mathbf{2}^\mathbf{v}\|}) \tag{4.3.10}$$

$$\mathbf{v}_{sew} = (\frac{{}^0\mathbf{p_4} - {}^0\mathbf{p_2}}{\|{}^0\mathbf{p_4} - {}^0\mathbf{p_2}\|}) \times (\frac{{}^0\mathbf{p_6} - {}^0\mathbf{p_2}}{\|{}^0\mathbf{p_6} - {}^0\mathbf{p_2}\|}) \tag{4.3.11}$$

The arm angle can now be calculated using the following equation:

$$\psi = sg_\psi \arccos{(\widehat{\mathbf{v}_{sew}^v} \cdot \widehat{\mathbf{v}_{sew}})} \tag{4.3.12}$$

The signs of the arm angle, denoted as sg_ψ, are determined by:

$$sg_\psi = sgn[(\widehat{\mathbf{v}_{sew}^v} \times \widehat{\mathbf{v}_{sew}}) \cdot {}^2\mathbf{p}_\mathbf{6}^\mathbf{v}] \tag{4.3.13}$$

Now that the arm angle has been determined, providing a comprehensive description of the robot in task space and solving the FK problem are feasible. However, the subsequent step involves addressing the IK problem, which presents certain challenges. It's notable that based on the arm angle and configuration set, multiple solutions may arise for the same pose.

In this section, we tackle the IK problem using the methodology proposed in [74]. Herein, the desired pose ${}_7^0T$ is provided, aiming to ascertain the joint angles that correspond to this pose. As previously mentioned, this robot exhibits redundancy, implying that there isn't a unique solution for a given pose. Our premise in this phase assumes that the arm angle and configuration set are predefined, and the objective is to identify a singular solution for the joints. Subsequently, we will address the redundancy issue using an alternative approach, separately.

It's noteworthy that the elbow joint remains independent of the arm angle and is thus

the initial joint to be determined. This joint's value can be calculated from:

$$\theta_4 = CS_4 \ \arccos \frac{\|{}^0\mathbf{p}_{sw}\|^2 - d_{se}^2 - d_{ew}^2}{2d_{se}d_{ew}} \tag{4.3.14}$$

where ${}^0\mathbf{p}_{sw}$ is

$${}^0\mathbf{p}_{sw} = {}^0\mathbf{p}_7 - {}^0\mathbf{p}_2 - {}^0_7\mathbf{R}^6\mathbf{p}_7 \tag{4.3.15}$$

To determine the shoulder joints, θ_1, θ_2, and θ_3, we initially consider the arm angle to be zero, implying that θ_3 is fixed, or locked. Subsequently, we ascertain the values of the virtual shoulder joints, θ_1^v and θ_2^v, using equations 4.3.7 and 4.3.8. Following this, the actual joint values are derived by rotating the virtual plane around the shoulder and wrist axis, incorporating the arm angle ψ.

Once the virtual shoulder and elbow joints are determined, we can ascertain the virtual rotation matrix of frame number three relative to the base, denoted as ${}^0_3R^v$. Subsequently, with the given arm angle ψ, the shoulder joints can be determined using the following equation:

$${}^0_3\mathbf{R} = \mathbf{A}_s \sin\psi + \mathbf{B}_s \cos\psi + \mathbf{C}_s \tag{4.3.16}$$

The constant matrices $\mathbf{A}_s$, $\mathbf{B}_s$, and $\mathbf{C}_s$ are predefined as follows:

$$\mathbf{A}_s = [{}^0\mathbf{u}_{sw}\times]\,{}^0_3R^v \tag{4.3.17}$$

$$\mathbf{B}_s = [{}^0\mathbf{u}_{sw}\times]^2\,{}^0_3R^v \tag{4.3.18}$$

$$\mathbf{C}_s = [{}^0\mathbf{u}_{sw}\,{}^0\mathbf{u}_{sw}^T]\,{}^0_3R^v \tag{4.3.19}$$

where ${}^0\mathbf{u}_{sw}$ is the unit vector of ${}^0\mathbf{p}_{sw}$, and $[{}^0\mathbf{u}_{sw}\times]$ shows the skew-symmetric matrix of that vector. By expanding the actual rotation matrix in equation 4.3.16, the relation between arm angle and shoulder joints can be determined using

$$\theta_1 = atan2(CS_2[a_{s22}\sin\psi + b_{s22}\cos\psi + c_{s22}], CS_2[a_{s12}\sin\psi + b_{s12}\cos\psi + c_{s12}]) \tag{4.3.20}$$

$$\theta_2 = CS_2 \arccos\left(a_{s32}\sin\psi + b_{s32}\cos\psi + c_{s32}\right) \tag{4.3.21}$$

$$\theta_3 = atan2(CS_2[a_{s33}\sin\psi + b_{s33}\cos\psi + c_{s22}], -CS_2[a_{s31}\sin\psi + b_{s31}\cos\psi + c_{s31}]) \tag{4.3.22}$$

Given the arm angle and configuration set, a specific set of solutions for the shoulder joints can be derived from the equations provided above. The subsequent task involves determining the wrist joints, namely θ_5, θ_6, and θ_7. To accomplish this, we first reconfigure the rotation component of equation 4.3.4 as follows:

$${}^0_7\mathbf{R} = {}^0_3\mathbf{R}\ {}^3_4\mathbf{R}\ {}^4_7\mathbf{R} \tag{4.3.23}$$

As previously discussed, the actual arm plane is derived from the virtual plane rotated along the vector from the shoulder to the wrist by the angle ψ. Consequently, the rotation matrix ${}^{0}_{3}\mathbf{R}$ can be expressed as:

$$ {}^{0}_{3}\mathbf{R} = {}^{0}_{\psi}\mathbf{R}\ {}^{0}_{3}\mathbf{R}^{v} \tag{4.3.24} $$

where ${}^{0}_{\psi}\mathbf{R}$ is

$$ {}^{0}_{\psi}\mathbf{R} = \mathbf{I}_3 + \sin\psi[{}^{0}\mathbf{u}_{sw}\times] + (1-\cos\psi)[{}^{0}\mathbf{u}_{sw}\times]^2 \tag{4.3.25} $$

By substituting equations 4.3.25 and 4.3.25 into equation 4.3.23, and utilizing matrices derived from the shoulder joint, the rotation matrix ${}^{4}_{7}\mathbf{R}$ can be reformulated as:

$$ {}^{4}_{7}\mathbf{R} = \mathbf{A}_w \sin\psi + \mathbf{B}_w \cos\psi + \mathbf{C}_w \tag{4.3.26} $$

where constant matrices $\mathbf{A}_w$, $\mathbf{B}_w$, and $\mathbf{C}_w$ are given by:

$$ \mathbf{A}_w = {}^{3}_{4}\mathbf{R}^{T}\ \mathbf{A}_s^{T}\ {}^{0}_{7}\mathbf{R} \tag{4.3.27} $$

$$ \mathbf{B}_w = {}^{3}_{4}\mathbf{R}^{T}\ \mathbf{B}_s^{T}\ {}^{0}_{7}\mathbf{R} \tag{4.3.28} $$

$$ \mathbf{C}_w = {}^{3}_{4}\mathbf{R}^{T}\ \mathbf{C}_s^{T}\ {}^{0}_{7}\mathbf{R} \tag{4.3.29} $$

By reworking the principal equation concerning each joint, we can establish the connection between the arm angle and the wrist joints, as illustrated by:

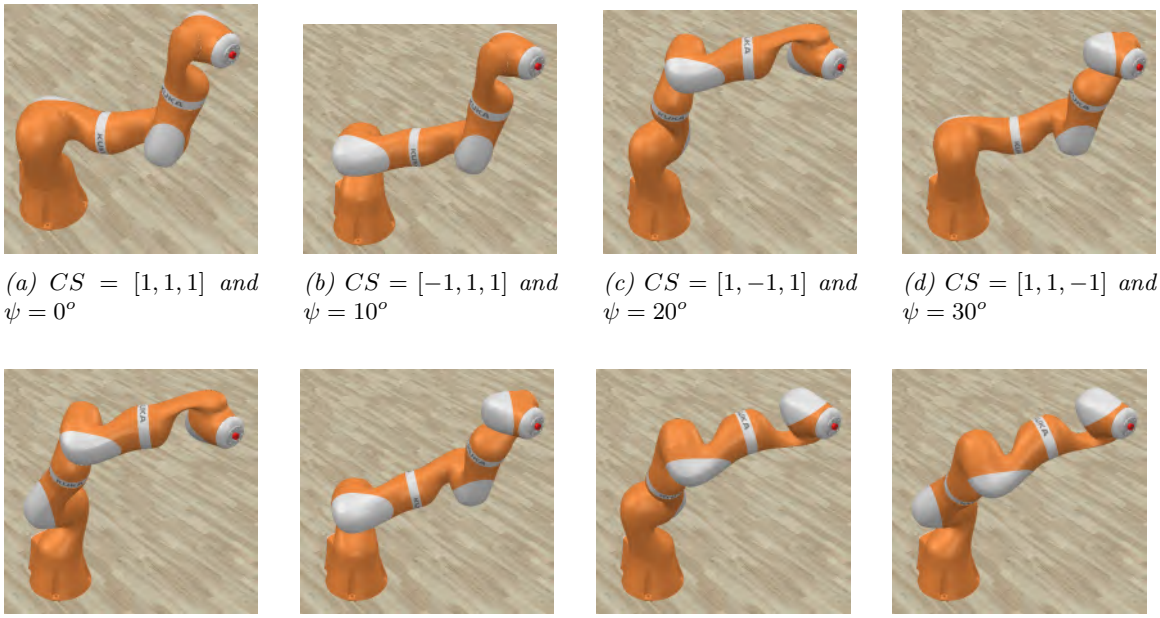
$$ \theta_5 = atan2(CS_6[a_{w23}\sin\psi + b_{w23}\cos\psi + c_{w23}], CS_6[a_{w13}\sin\psi + b_{w13}\cos\psi + c_{w13}]) \tag{4.3.30} $$

$$ \theta_6 = CS_6 \arccos\left(a_{w33}\sin\psi + b_{w33}\cos\psi + c_{w33}\right) \tag{4.3.31} $$

$$ \theta_7 = atan2(CS_6[a_{w32}\sin\psi + b_{w32}\cos\psi + c_{w32}], -CS_6[a_{w31}\sin\psi + b_{w31}\cos\psi + c_{w31}]) \tag{4.3.32} $$

These equations allow for the determination of all joint angles given a specific arm angle and configuration set.

As discussed previously, the IK problem was addressed by assuming predetermined values for the arm angle and configuration set. However, this raises the question of how these parameters should be determined. There might be more than on solution for a given pose. For instance, in Figure 4.5, 8 different solutions are shown for a single pose. All these robot configurations resulted in a same end-effector pose. The CS and ψ parameters are written for each case. Note that, there exist more solutions than those illustrated in this figure. The answer to this inquiry hinges on the project's objectives and requirements, such as obstacle avoidance, singularity avoidance, adherence to joint limits, minimization of time and energy consumption, among others. In our project, our primary focus revolves around avoiding joint limits and singularities to ensure the safety and optimal performance of the



(a) $CS = [1, 1, 1]$ *and* $\psi = 0^o$ *(b)* $CS = [-1, 1, 1]$ *and* $\psi = 10^o$ *(c)* $CS = [1, -1, 1]$ *and* $\psi = 20^o$ *(d)* $CS = [1, 1, -1]$ *and* $\psi = 30^o$

(e) $CS = [-1, -1, 1]$ *and* $\psi = 30^o$ *(f)* $CS = [-1, 1, -1]$ *and* $\psi = 40^o$ *(g)* $CS = [1, -1, -1]$ *and* $\psi = 50^o$ *(h)* $CS = [-1, -1, -1]$ *and* $\psi = 60^o$

Figure 4.5: Redundant Solutions Illustration

robot arm.

Among the various criteria, joint limits pose the most significant concern for our project. Specifically, the *KUKA* robot employed in this thesis features two joints, namely joints number 4 and 6, each with joint limits of $\pm 120°$. These limits are relatively low compared to those of other joints. Critical points, which may include joint limits or singularities, must be carefully managed to prevent undesirable outcomes.

To address this challenge, we adopted the methodology proposed in [74] and subsequently refined it in [76]. This approach entails a comprehensive analysis of the stationary points of two specific functions:

$$\theta_i(\psi) = \arctan 2(CS_K\ [a_n \sin\psi + b_n \cos\psi + c_n], CS_k\ [a_d \sin\psi + b_d \cos\psi + c_d]) \quad (4.3.33)$$

$$\theta_i(\psi) = CS_k \arccos\left(a \sin\psi + b \cos\psi + c\right) \quad (4.3.34)$$

Through a meticulous examination of these two functions at their stationary points and juxtaposing them with individual joint limits, viable intervals can be delineated. Once these intervals are identified, the arm angle ψ can be chosen utilizing diverse methodologies advocated in existing literature. In this thesis, the mean value within the feasible interval is adopted.

For every joint, the feasible intervals for the arm angle are computed, and subsequently,

the union of these intersecting intervals Ψ_j is derived for the j^{th} joint. The permissible interval for Ψ_{all} is determined as the intersection of all these intervals, as illustrated in the subsequent equation:

$$\Psi_{all} = \bigcap_{j=1}^{n_j} \Psi_j \tag{4.3.35}$$

where Ψ_j is

$$\Psi_j = \bigcup_{i=1}^{n_i} \Psi_{j,i} \tag{4.3.36}$$

Joints where the arctan function is employed for calculation are termed pivot joints, while those utilizing the arccos function are classified as hinge joints. These two categories of joints are scrutinized independently.

To correlate the arm angle with joint limits and singular points, the stationary points ψ_0 are determined by differentiation. For pivot joints, upon differentiation and simplification of the expression, the stationary points are obtained as:

$$\psi_0 = 2 \arctan(\frac{a_t \pm \sqrt{a_t^2 + b_t^2 - c_t^2}}{b_t - c_t}) \tag{4.3.37}$$

The constants a_t, b_t, and c_t are defined as follows:

$$\begin{aligned} a_t =& CS_K(b_d c_n - b_n c_d) \\ b_t =& CS_K(a_n c_d - a_d c_n) \\ a_t =& CS_K(b_d c_n - b_n c_d) \end{aligned}$$

Based on the value of $a_t^2 + b_t^2 - c_t^2$, denoted by Δ, three distinct cases emerge:

1. $\Delta > 0$: In this case, the stationary points of the joint angle θ_i are present, and the joint angle reaches its maximum or minimum at two arm angles found by:

$$\psi_0^- = 2 \arctan(\frac{a_t - \sqrt{a_t^2 + b_t^2 - c_t^2}}{b_t - c_t}) \tag{4.3.38}$$

$$\psi_0^+ = 2 \arctan(\frac{a_t + \sqrt{a_t^2 + b_t^2 - c_t^2}}{b_t - c_t}) \tag{4.3.39}$$

2. $\Delta < 0$: In this scenario, there are no stationary points, and the function $\theta(\psi)$ exhibits monotonic behavior, establishing a one-to-one correspondence between ψ and θ. Consequently, this case poses no concerns regarding joint limits and singular points.

3. $\Delta = 0$: In this scenario, a solitary stationary point ψ_{sing} is established by the equation:

$$\psi_{sing} = 2\arctan\left(\frac{a_t}{b_t - c_t}\right) \tag{4.3.40}$$

To manage this particular circumstance, a safety margin is defined to steer clear of approaching this point.

The identical approach is utilized for hinge joints, where two potential scenarios arise depending on the value of $a^2 + b^2 - (c \pm 1)^2$.

1. $a^2 + b^2 - (c \pm 1)^2 \neq 0$**:** In this scenario, two stationary points are evident, and the function $\theta_i(\psi)$ is minimized at one of these points while maximized at the other. The corresponding arm angle at these points is determined using:

$$\psi_0 = 2\arctan\left(\frac{-b \pm \sqrt{a^2 + b^2}}{a}\right) \tag{4.3.41}$$

2. $a^2 + b^2 - (c \pm 1)^2 = 0$**:** In this scenario, a singularity occurs, placing the robot in a singular configuration. This situation arises when $\sin\theta_i = 0$ for $i = 4, 6$. The corresponding arm angle for this case can be determined by:

$$\psi_0 = 2\arctan\left(\frac{a}{b - (c \pm 1)}\right) \tag{4.3.42}$$

Now that the behavior of these two types of functions has been analyzed, a feasible interval can be determined to avoid joint limits and singular points. Assuming that joint limits are defined by:

$$\theta_i^l \leq \theta_i \leq \theta_i^u, \quad (i = 1, 2, \cdots, n_j) \tag{4.3.43}$$

and the global maximum and minimum $\theta_i^{min}, \theta_i^{max}$ are identified, finding feasible regions can be categorized into the following cases:

1. $\theta_i^{min} > \theta_i^u$ or $\theta_i^{max} < \theta_i^l$: In this scenario, there is no feasible region for the arm angle.
2. $\theta_i^{min} < \theta_i^l$ and $\theta_i^l \leq \theta_i^{max} \leq \theta_i^u$: In this case, a specific region including ψ_0^{max} is allowed.
3. $\theta_i^l \leq \theta_i^{min} \leq \theta_i^u$ and $\theta_i^{max} > \theta_i^u$: In this case, a specific region including ψ_0^{min} is allowed.
4. $\theta_i^{min} < \theta_i^l$ and $\theta_i^{max} > \theta_i^u$: The allowed region is found by solving $\theta_i(\psi) = \theta_i^l$ and $\theta_i(\psi) = \theta_i^u$. The interval found is the feasible.
5. Singular cases should also be considered, and in the event of any singular points, that point should be excluded from the feasible interval with a safe threshold.

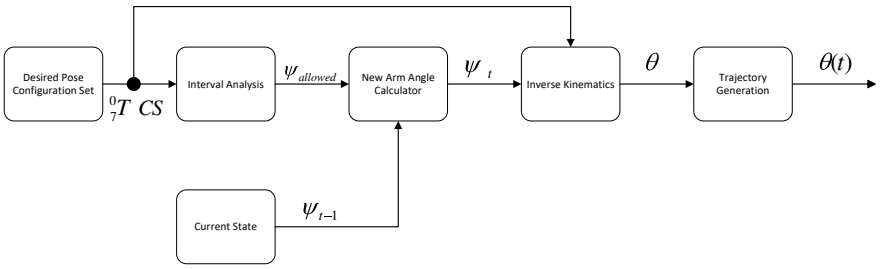


Figure 4.6: Motion Planning Framework

After this process is applied to all the joints, the ultimate permissible range for the arm angle ψ is determined as the intersection of all intervals. The arm angle can be selected from within this interval, preferably close to the previous arm angle.

With the inverse kinematics problem resolved and a method established to determine a suitable arm angle, motion planning for the robot can now be executed using these techniques. In motion planning, the objective is to maneuver the robot's tool to a desired position. It's important to note that in this thesis, only motion control is addressed, and force control is not taken into account. The framework of the proposed motion control algorithm is illustrated in Figure 4.6. In this framework, the desired pose ^0_7T and configuration set CS are provided, and the current arm angle ψ_{t-1} is determined from the simulation or actual state. Subsequently, the interval analysis method identifies a feasible interval for the arm angle $\psi_{allowed}$. Leveraging this feasible interval and the current arm angle, a new arm angle ψ_t is computed using the formula:

$$\psi_t = \frac{\psi_{upper} - \psi_{lower}}{2} \tag{4.3.44}$$

where ψ_{upper} and ψ_{lower} represent the upper and lower limits of the interval within which the current arm angle ψ_{t-1} lies. Once the arm angle is known, along with the desired pose ^0_7T and configuration set CS, the inverse kinematics (IK) method can be utilized to determine a unique solution for the joint angles. This solution represents the desired values for the joints for a specific pose. By employing a cubic trajectory, a trajectory in joint space is generated, enabling the robot to follow this trajectory to attain the desired pose. The use of a cubic trajectory ensures that the trajectory is smooth, and the robot remains stationary at the initial and final times. A cubic trajectory for each joint can be generated using the following formula:

$$q(t) = q_0 + 3(\frac{q_f - q_0}{T^2})t^2 - 2(\frac{q_f - q_0}{T^3})t^3 \tag{4.3.45}$$

where q_0 represents the initial value for the joint, q_f denotes the final value for the joint, and T signifies the time span of the motion.

In this section, the kinematic modeling of the robot is conducted. This method facilitates motion control for the robot, both in simulation and in the actual robot setup.

4.4 Kinematic Modelling of Robotiq3F Gripper

In this section, the gripper used in this thesis is described and modeled in simulation. The component employed is a three-finger adaptive gripper manufactured by *ROBOTIQ*. Initially, the rationale for choosing this particular gripper for this thesis is explained. Subsequently, the gripper itself is described, along with the extraction of the mechanism diagram and joint specifications. Since, to the best of the author's knowledge, this gripper had not been modeled in any simulation environment prior, a novel kinematic model for the gripper is proposed using a state system approach.

The gripper component plays a crucial role in the task of smart grasping. Different objects with varying shapes, sizes, and materials are intended to be grasped, and the gripper is the primary tool to perform this task. Various types of grippers are available in the industry, such as mechanical grippers (including parallel, centric, angular/radial), pneumatic grippers, adhesive grippers, and magnetic grippers. Most mechanical grippers have two or three fingers. Some specialized grippers, inspired by human hands, are also available, such as the anthropomorphic five-finger hand manufactured by *SCHUNK*. Choosing an appropriate gripper depends on the specific requirements of the project.

Alongside mechanical specifications such as dimensions and force, other qualitative indices such as performance and flexibility are important, as discussed in [77]. Performance in grippers can be defined as the efficiency with which a specific task is performed, while flexibility refers to the number of different tasks a gripper can perform. For instance, in Figure 4.7, a comparison is made between a two-finger gripper, a three-finger gripper, and a human hand using these two indices. As shown in the figure, two-finger grippers have a high performance, meaning they are well-suited for specific tasks without requiring adjustments. Three-finger grippers, on the other hand, are ideal for dynamic environments. At the top of the hierarchy are human hands, which excel in both performance and flexibility indices.

In this thesis, the primary goal is to grasp objects of various shapes and sizes, making the flexibility index more crucial than the performance index. Additionally, advanced grippers, such as five-finger grippers inspired by human hands, are too complex and overqualified for the grasping task at hand. Therefore, a three-finger gripper was chosen for this task.

This gripper is an adaptive three-finger gripper manufactured by *ROBOTIQ*. The term "adaptive" indicates that it has multiple grip modes, such as basic mode, wide mode, pinch mode, and scissor mode. It features three fingers, each with three links, as shown in Figure 4.8a. The grip modes are switched by changing the relative angle between fingers

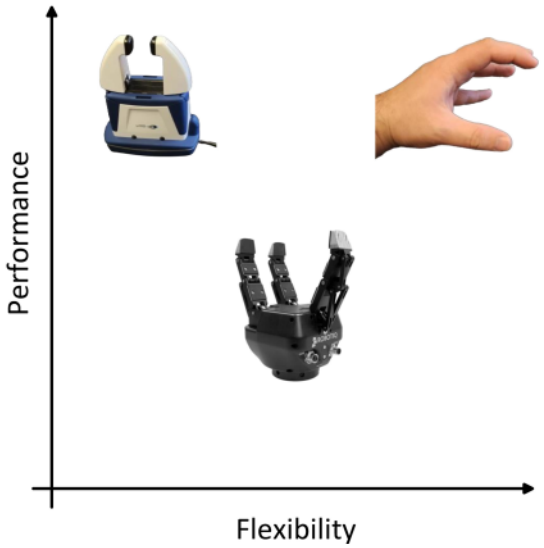
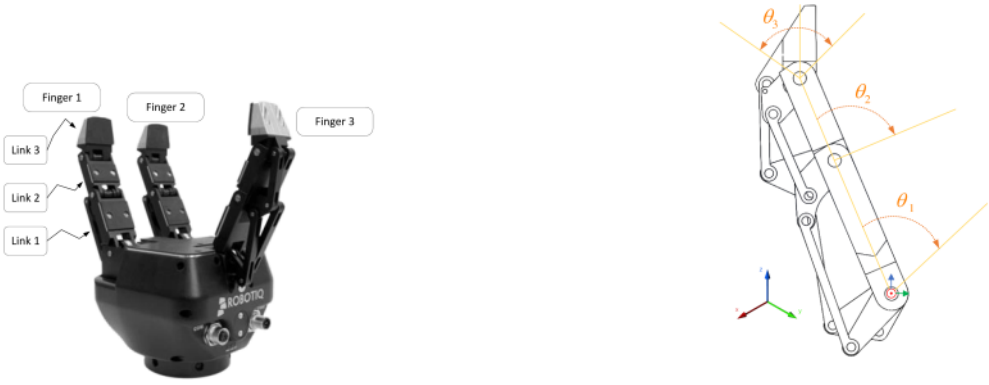


Figure 4.7: Performance and flexibility comparison for a two-finger, three-finger grippers, and human hand

1 and 2. Each finger is actuated with an electric motor and a clutch and brake system. Although the fingers are controlled and actuated independently, their actuation mechanisms are identical. This means that modeling one finger in simulation suffices for modeling the others. Before delving into the gripper modeling, the dynamics of the gripper and its operation are explained.

As mentioned earlier, each finger comprises three links, all actuated by a single motor. The primary links are connected by several minor links, creating a mechanical constraint that limits the motion range of each link. The motion range and direction are illustrated in Figure4.8b. As shown in the figure, all the links can only rotate about their X axis. The rotations of links one, two, and three are denoted by θ_1, θ_2, and θ_3, respectively. Links



(a) General Description *(b) Finger's Mechanism*

Figure 4.8: ROBOTIQ 3-Finger Adaptive Gripper Fingers

one and two can only rotate in one direction, which is closing, while link three can rotate in both directions. Based on observations from the actual gripper, and using the method described later in Section 4.8.3, the joint rotation ranges are as follows:

$$\theta_1 : [0, 67^\circ] \tag{4.4.1}$$

$$\theta_2 : [0, 90^\circ] \tag{4.4.2}$$

$$\theta_3 : [-55^\circ, 43^\circ] \tag{4.4.3}$$

These joint limits will later be used as transition conditions in the modeling process.

With the technical details of the gripper covered, we can now begin the modeling phase. As mentioned previously, CoppeliaSim software is used in this thesis as the simulation environment. To import a machine like a robot or a gripper into this software, it is necessary to generate a Unified Robotics Description Format (URDF) file. A URDF file, along with mesh files, forms a package that provides the physics engine with essential information such as geometry, dimensions, and mass.

One way to generate this package is to extract the exact dimensions from the actual device, create the mesh files for all parts, and manually write the URDF file. However, this method is time-consuming and prone to errors. A more efficient approach is to use a developed toolbox that generates this package from a CAD file. One of the most popular toolboxes for this purpose is the SW2URDF toolbox developed by Brawner [78]. This toolbox, designed for *SOLIDWORKS* software, enables the generation of URDF packages from a CAD file.

The CAD file, provided by the manufacturer, was imported into the CAD software. The toolbox requires all bodies and joints to have defined coordinate systems and axes. During this process, over 40 reference geometry features were defined in the CAD software. This step ensured that all parts and joints were accurately represented and aligned, which is critical for the proper functioning of the simulation.

Once the URDF package was generated, it was imported into the simulation software. Another crucial step after importing was ensuring that all parts were convex. This requirement is important as non-convex shapes can cause instability during simulation for physics engines. Ensuring convexity involved checking and, if necessary, modifying the geometry of the parts to meet this criterion. This step helps in maintaining the stability and reliability of the simulation, preventing potential errors or crashes.

A schematic of the entire process, from CAD modeling to URDF generation and import into CoppeliaSim, is shown in Figure 4.9. This figure illustrates the workflow and highlights the key steps involved in preparing the gripper for simulation. By following this structured

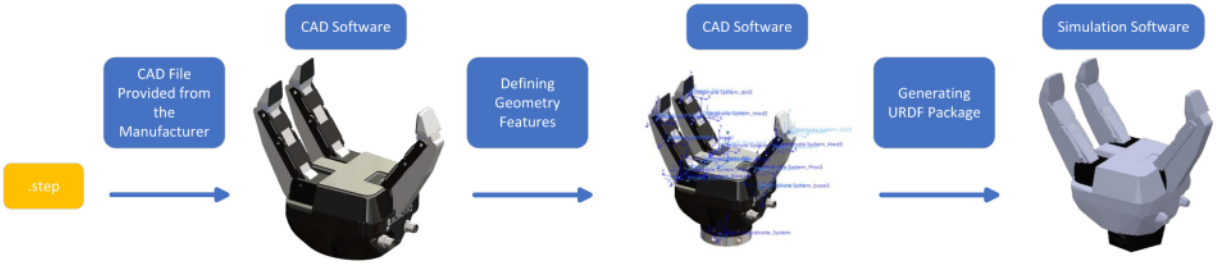


Figure 4.9: From CAD file to URDF package process

approach, we ensure that the gripper model is accurately represented and ready for detailed simulation and analysis. This preparation is essential for subsequent tasks, such as motion planning and control, which rely on a precise and stable model of the gripper.

With a proper simulation model established for the gripper, the joints can now be controlled as desired. The simulation software utilized in this thesis offers various joint control options, including position, force, and speed control. According to the simulation software manual, it is assumed that the robot and the gripper are equipped with joint controllers for position and speed. This assumption is valid since modern robots and grippers typically come with reliable controllers. For the modeled gripper, two methods were employed to control the joint motion: position control and speed control. The joint control aspect of the modeling is explained in the following paragraphs.

As previously mentioned, the Robotiq 3F gripper consists of three fingers, each individually powered by an electric motor and composed of three links. The mechanical structure allows for various link configurations based on joint limits or contact with objects.

As illustrated in the Figure 4.8b, each finger, similar to a human hand, is made up of three links. The movement of these links depends on the position of objects and the joint limits. The model developed here is based on observations of the actual gripper and information from a technical report [79].

First, let's explore the concept of joint limits. Links one and two have a lower joint limit set at zero, meaning they cannot move below this point. In contrast, link number three can move in the opposite direction. This is important because, as link number one begins to close, link number three must either remain vertical or in its initial position and move in the reverse direction of link number one. Without an object within the gripper's grasp, the mechanism is influenced solely by joint limits.

When the close command is sent to the gripper, links one and three start moving at

the same speed but in opposite directions until link number three reaches its lower limit. Afterward, link number one continues to move until it reaches its upper limit. Once link number one stops, link number two begins its motion until it reaches its joint limit. The movement of the finger without an object can be accurately modeled by a piecewise-defined function, as shown in equation 4.4.4, where g represents the motor position and θ_i represents the position of link number i:

$$[\theta_1, \theta_2, \theta_3] = \begin{cases} [V_1 g, 0, -V_1 g] & 0 \leq g \leq 110 \\ [V_1 g, 0, 0] & 110 < g \leq 140 \\ [0, V_2 g, 0] & 140 < g \leq 255 \end{cases} \tag{4.4.4}$$

The motor position is represented by an integer $g \in [0, 255]$, where $g = 0$ indicates a fully open position, and $g = 255$ denotes a fully closed position. Furthermore, V_1 and V_2 are the velocities of links 1 and 2, respectively, and are defined as follows:

$$\begin{aligned} V_1 &= \Theta_1^{max}/140 \\ V_2 &= \Theta_2^{max}/100 \end{aligned} \tag{4.4.5}$$

As illustrated in the equations, modeling the gripper's motion is straightforward when there is no object. However, this model encounters a significant issue. Specifically, the speed of the links in the simulation is slow because they consist of 256 steps, and each step takes time. During the simulation, it took approximately 20 seconds for the gripper to fully close, whereas the actual gripper only took about four seconds.

The behavior of the gripper becomes more complex when an object is within the gripper's hull. This complexity arises because, upon detecting a contact, the motor doesn't stop moving immediately. Depending on the position of the links, various situations can occur. Additionally, the motor continuously applies a gripping torque to the links when it is not in motion to prevent the grasped object from falling. This persistence of constant torque was observed when a consistent electrical current was displayed by the Robotiq User Interface software. When the close command is sent to the gripper, links 1 and 3 start moving, similar to the scenario without an object. The motion continues until one of the following events occurs:

- one of the links reaches the joint limit
- one of the links contacts the object

This sequence continues until the third link either contacts the object or reaches its limit. There is an exception when there is no object, and in this case, the finger stops moving when link number two reaches its limit.

As previously discussed, determining the actual configuration of the links in the presence of an object is not possible unless the exact motor position g_{obj}, where the contact occurred, is known. In this scenario, depending on this value, a mathematical model can be developed [79]. However, in the simulation, especially when training a reinforcement learning agent, the environment is random and uncertain, which prevents the use of this precise model. To address this, it is preferable to model the system using a state machine representation rather than a piecewise-defined function.

A state machine is a computational model that represents the dynamic behavior of a system by defining a set of states, transitions between these states, and the events that trigger these transitions. Each state represents a condition or mode in which the system can exist, and transitions denote how the system moves from one state to another in response to events. State machines are used to model the sequence of actions or behaviors a system can exhibit and are widely employed in software engineering, control systems, and various other fields to describe and analyze the logic and flow of processes.

By adopting a state machine representation, the gripper's behavior can be effectively modeled, accounting for the random and uncertain nature of the simulation environment. This approach provides a more flexible and robust framework for simulating and controlling the gripper's movements in various scenarios, enhancing its performance and reliability in practical applications.

For each finger in the gripper, the state machine graph is depicted in Figure 4.10, designed using Microsoft Visio. As previously mentioned, controlling the joints in position mode is not suitable for this system, and controlling them in velocity mode operates more efficiently. In Figure 4.10, the parameter L_i indicates that link number i has reached its limit, with the special case L_4 indicating that link number three has reached the upper limit. The same terminology is used for contacts, where C_i denotes that link number i has made contact with the object. This state machine is implemented in the simulation software.

To implement this behavior in the simulation software, the Python StateMachine library was utilized. After implementing the system, the output state machine representation was generated, as shown in Figure 4.11. The behavior was implemented as a class, with each finger being an instance of that class. After creating the class, a graph was generated using the built-in methods of the library to verify it against the graph from Visio. Using this state machine, once the close command is sent to the gripper, appropriate actions will be sent to the simulation software depending on the events, such as joint limits and contacts. By using this state machine, after the close command is sent to the gripper, depending on the events which can be joint limits and contacts, suitable action will be sent to the simulation software.

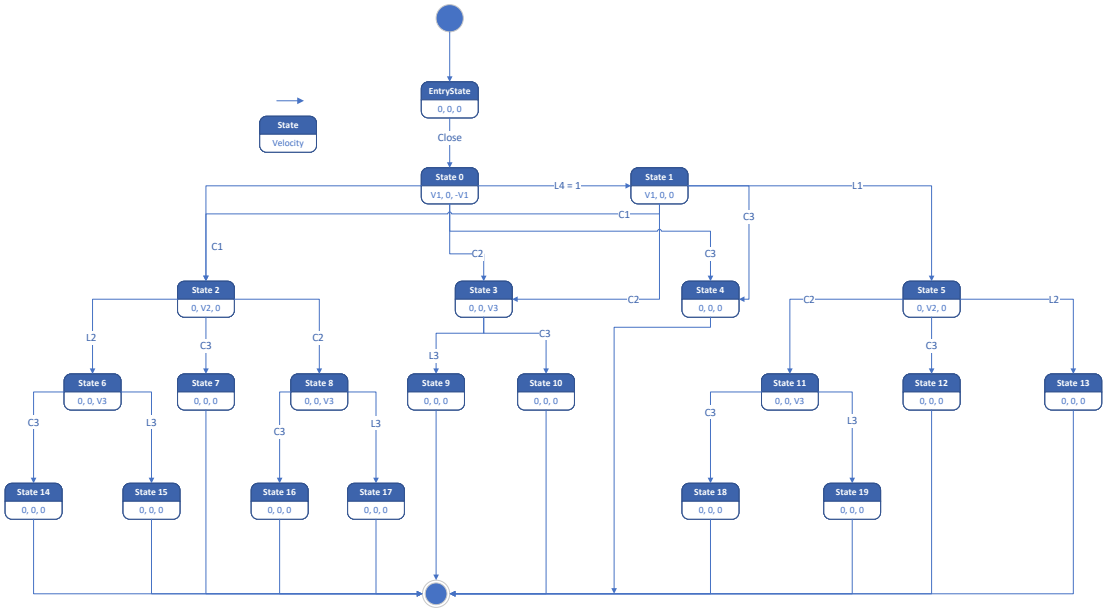


Figure 4.10: State Machine Representation

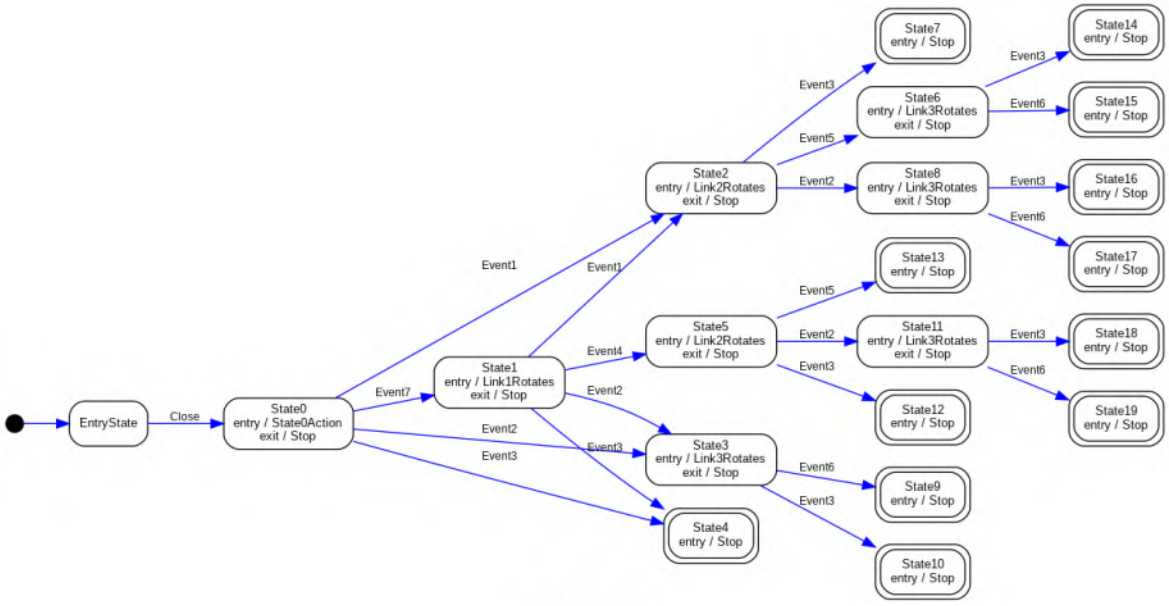


Figure 4.11: State Machine Representation in Python

Now that we have the state machine, two more components are needed to complete the model: moving fingers in velocity mode and detecting events. To move the links, the control mode of the links was changed to velocity mode in Coppeliasim. When a velocity command is sent, the built-in controller in the software starts increasing the velocity until it reaches the desired speed. This type of control has two drawbacks: first, the speed does not change immediately, similar to the real gripper. Additionally, stopping the link presents its own challenges. To stop the joint from moving in velocity mode, an opposite force must be applied. Controlling the magnitude of this force is not straightforward in the physics engine. To address this, Coppeliasim provides a trick to stop the joint immediately when the velocity command is set to zero. While using this trick may reduce the model's accuracy, it simplifies the model and does not affect the final result significantly.

Another important aspect is detecting contact. Modeling an accurate contact requires a powerful engine and extensive studies and research, which are beyond the scope of this thesis. Contact elasticity and damping parameters, along with the torque of the motors, must be adjusted to model a contact. To detect a contact in this model, a proximity sensor was placed at the tip of each link. The range of these sensors was set to the minimum value of 1 mm to model the contact. Therefore, when a contact event occurs, it is not an actual contact, and the object is tangent to the tip's surface. Despite this limitation, where the object may fall during manipulation, in an actual scenario, the object would not fall. This drawback is solely due to contact detection limitations. For demonstration purposes, there is a method in the literature called "fake grasp." In this method, if the object is between the fingers and the distance threshold is satisfied, the object will be attached to the gripper and move with it. The object can be reattached when the open command is sent to the gripper.

4.5 Estimating Observation Data Using Vision Algorithm

In this section, we discuss the vision component of the smart grasping system. As illustrated in Figure 4.1, the vision sensor is one of the three main components utilized in this thesis. The primary function of this component is to observe the environment and provide as much information as possible to the DRL agent. This information typically includes the position and shape of the desired object.

The vision sensor captures both Red, Green and Blue (RGB) and depth images from the environment. These images are then processed using vision algorithms, and the processed information is sent to the DRL agent. The vision sensor employed in this project is the *Xbox Kinect* sensor, manufactured by *Microsoft*. In the following paragraphs, we will explain the methods implemented to retrieve the necessary information.

In the simulation phase, the vision sensor needs to provide information about the height

of the object and a set of two-dimensional points representing the object's perimeter. The Xbox Kinect sensor captures the RGB and depth images, which are then processed to extract these details. The RGB image helps in identifying the object's color and texture, while the depth image provides information about the distance of the object from the sensor.

To extract the object's height, we process the depth image to determine the distance from the sensor to various points on the object. By identifying the closest and farthest points, we can calculate the object's height. Additionally, to outline the object's perimeter, edge detection algorithms are applied to the RGB image to identify the object's boundaries. These boundaries are then converted into a set of two-dimensional points representing the object's perimeter.

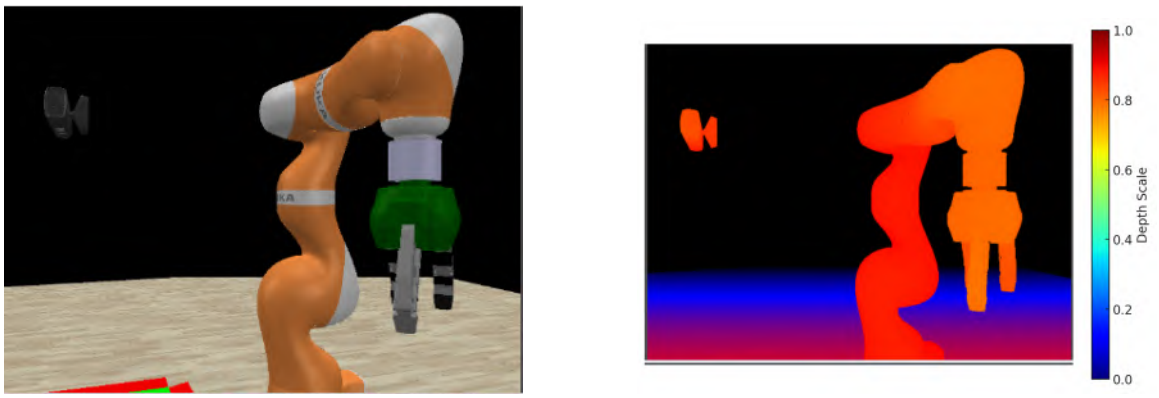
The extracted information is crucial for the DRL agent to understand the environment and make informed decisions about how to grasp the object. By providing detailed data about the object's position, height, and shape, the vision sensor enables the DRL agent to plan and execute effective grasping actions.

In summary, the vision sensor plays a vital role in the smart grasping system by capturing and processing visual information from the environment. This processed information is then used by the DRL agent to accurately identify and grasp objects, ensuring the efficiency and effectiveness of the smart grasping system.

Vision-based approaches are gaining popularity in robotic manipulation due to their capability to provide detailed sensory data about the environment[80]. This sensory data is essential for robots to perform manipulation tasks effectively. Early applications included object detection, path planning, and navigation. More recent research has tackled more complex tasks such as grasping under occlusion and manipulation through human demonstration.

A vision sensor typically consists of two main components: an RGB sensor that captures a regular image and a depth sensor that provides depth information. The vision sensor module used in this thesis was already available in the simulation software's library. This module allows for the capture and processing of both RGB and depth images, enabling a comprehensive understanding of the environment.

For instance, the RGB sensor provides color and texture information, which is useful for identifying objects and their features. The depth sensor, on the other hand, measures the distance to various points in the scene, enabling the determination of an object's position and height. By combining these two types of data, the vision system can create a detailed model of the environment. A sample data output from the vision sensor is illustrated in Figure 4.12. From the RGB data presented in Figure 4.12a, the target object can be detected by identifying its color and texture. Additionally, from the depth data shown in



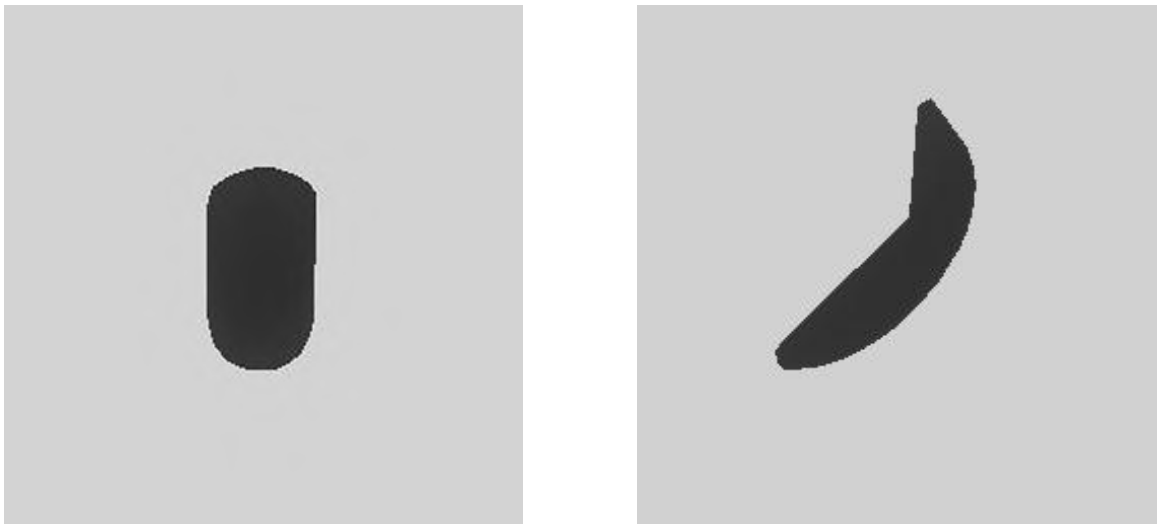
(a) Sample RGB Data *(b) Sample Depth Data*

Figure 4.12: Sample Vision Data in Simulation

Figure 4.12b, the position and shape of the desired object can be estimated by analyzing the distance measurements provided. The vision sensor was imported into the simulation environment and strategically positioned above the area where objects are placed. This placement was crucial to ensure that the object remained within the camera's field of view, allowing for accurate detection and measurement. By combining the RGB and depth data, the system can effectively identify and locate objects within the environment. This integration is essential for the successful implementation of the vision-based manipulation tasks discussed in this thesis. The detailed information provided by the sensor allows the Deep Reinforcement Learning (DRL) agent to make informed decisions, enhancing the overall performance and accuracy of the smart grasping system.

To expedite the training process, the RGB data was omitted, assuming that the object was already detected within the simulation. This was achieved by positioning the camera perpendicular to the ground at a specific height, ensuring the ground could be clearly distinguished from the object and that no other items were present on the ground. This setup was also necessary due to the limited action space defined for the robot.

As will be discussed in the environment section, the gripper's orientation was restricted to rotation in only one direction, ensuring that the gripper always faced perpendicularly to the ground. The data provided by the simulation software is an array containing the depth values of all the pixels. The dimensions of this array are determined by the resolution of the sensor. For example, if the resolution is 256 by 256 pixels, the array will have a dimension of $256 \times 256 = 65,536$. Each cell in this array represents the relative distance between the sensor and the object. A sample of the depth data is illustrated in Figure 4.13 To estimate the height of the object from the depth image, a straightforward technique was employed. Given that the distance between the sensor and the gripper, denoted as D, is known and remains constant, the height of the object can be estimated by subtracting the distance of



(a) Sample depth image of a mouse *(b) Sample depth image of a banana*

Figure 4.13: Sample Depth Image in Simulation

the closest point from this fixed distance. In the simulated environment, this fixed distance D is set to 1.02 meters. The principle behind this height estimation method is depicted in Figure4.14. The parameter $min_{deptharray}$ is determined by identifying the minimum value

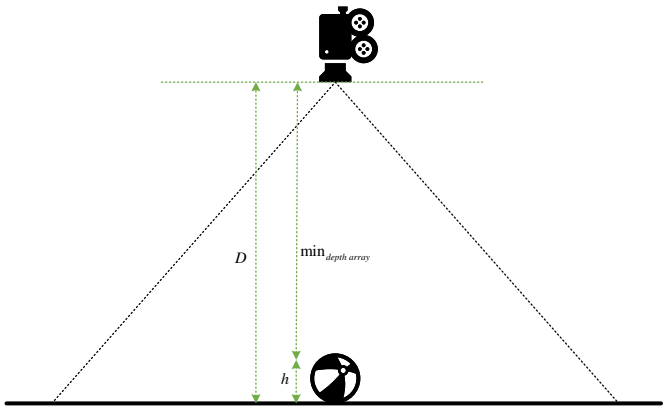


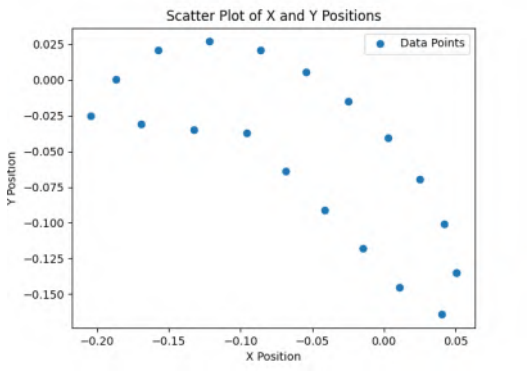
Figure 4.14: State Machine Representation

in the depth array, which corresponds to the closest point to the sensor. By employing this straightforward method, the height of the object, denoted as h, can be estimated using the following formula:

$$h = D - min_{depth\ array} \tag{4.5.1}$$

Here, D represents the fixed distance between the sensor and the gripper, and $min_{deptharray}$ is the minimum depth value obtained from the depth image. This approach effectively

(a) Depth image provided to the algorithm



(b) Output of the algorithm

Figure 4.15: Sample example for the proposed algorithm to find the perimeter

calculates the height of the object based on the difference between the fixed distance and the closest point detected by the sensor. This method ensures a simple and accurate estimation of the object's height, which is essential for the smart grasping system's functionality.

The next piece of information required for the DRL agent is a set of two-dimensional points representing the perimeter of the object. To obtain a uniform set of points outlining the perimeter, a method based on Otsu's[81] edge detection technique is proposed. Using this edge detection tool, the largest contour detected is selected. This contour is then resampled to create a uniform distribution of points along the perimeter. For example, this process is demonstrated for a sample object shown in Figure 4.15.

In this section, the vision component of the simulation framework was discussed. This component plays a crucial role since the DRL agent learns based on the data provided by the vision sensor. The significance of the vision component is even greater in the real-world framework, as the vision algorithm must not only detect the desired object but also estimate its position accurately. By ensuring the vision sensor provides precise and reliable data, the overall effectiveness of the smart grasping system is enhanced.

4.6 Object Dataset

In this section, the dataset of objects used to train the agent during the simulation is discussed. The robot learns the grasping task by attempting to grasp these objects. As mentioned earlier, random environments are more efficient for training because they allow the agent to explore the state space more thoroughly. Following this logic, to improve the overall efficiency of the training, objects with different shapes and sizes were selected. These selected objects form a dataset known as the object dataset. Creating this dataset involves modeling the desired objects in CAD software and then importing these CAD models into

the simulation software. An important consideration during the importing process is that all objects must be convex to ensure the physics engine functions correctly. This section outlines all these steps.

A total of 24 objects were created, including both primitive shapes and real-life objects. First, these objects were modeled in *SOLIDWORKS*, and then converted to OBJ files using *MESHLAB* software. The OBJ format contains geometry information and is compatible with the simulation software, allowing for easy importation. After importing the objects into the simulation software, it was essential to ensure that all the objects in the dataset were convex. Objects targeted for grasping by the gripper must possess a convex structure to ensure stability within the simulation conducted by the physics engine. A shape is considered convex if it contains the line connecting any two points within the shape. While not all objects in the dataset strictly meet this criterion, this challenge was addressed by decomposing the shapes into multiple convex sub-shapes or meshes. Using convex shapes in simulations involving collisions and contact interactions is preferred for several reasons according to[82],[83], and [72]:

- **Efficiency:** Collisions involving convex shapes can be computed more efficiently than those with concave shapes. Convex shapes allow for simpler and faster collision detection algorithms, leading to quicker simulations.
- **Stability:** Convex shapes contribute to more stable simulations. Concave shapes can cause unexpected behaviors and instability in the simulation, especially during collision resolution. Using convex shapes reduces the likelihood of objects getting stuck or behaving unrealistically.
- **Simplicity:** Convex shapes are mathematically simpler to work with. Algorithms for convex shapes are well-established and widely implemented in physics engines. Dealing with concave shapes involves more complex and computationally intensive calculations.
- **Accuracy:** Physics engines can provide more accurate and reliable results with convex shapes. Concave shapes often require approximation methods or decomposition into multiple convex shapes, which can introduce errors.
- **Collision Response:** Resolving collisions with convex shapes is more straightforward. Concave shapes might require complex algorithms for collision response, which can be computationally expensive.

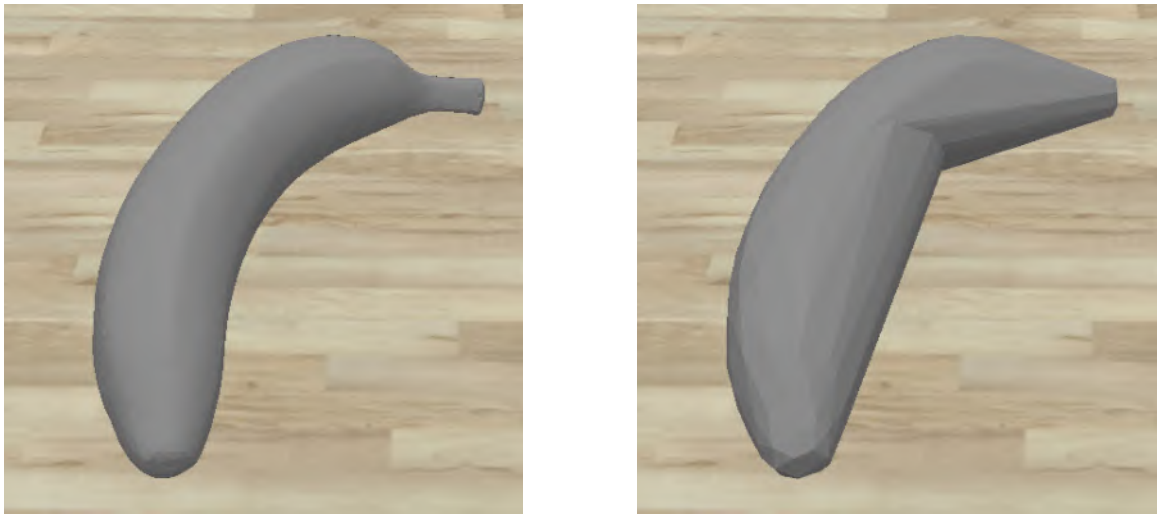
In summary, precise modeling of the interaction between the gripper fingers and the object demands the utilization of convex shapes. CoppeliaSim software provides dependable and stable methods for handling convex shapes. It can either convert the shape into a single

convex form or decompose it into multiple convex shapes. There are two methods available for generating convex shapes in the simulation software:

- **Hierarchical Approximate Convex Decomposition[84] (HACD):** HACD is a method used to decompose 3D models into a set of approximately convex components. Convex shapes are simpler to work with in collision detection and physics simulations. HACD works by iteratively simplifying the input mesh into smaller convex hulls. It starts with a coarse decomposition and then refines it iteratively to obtain a more accurate approximation. The resulting convex shapes can be used in physics engines for collision detection, allowing efficient simulation of interactions with the original complex model.

- **Volumetric Hierarchical Approximate Convex Decomposition[85] (V-HACD):** V-HACD is an extension of HACD that introduces a volumetric representation to enhance the decomposition process. Instead of working directly with the mesh geometry, V-HACD first represents the model as a voxel grid (a 3D grid of uniformly sized cubes). By operating in this voxel space, the algorithm can identify regions of the model with varying densities. Sparse regions (with fewer voxels) are approximated with fewer convex components, while denser regions (with more voxels) are decomposed into more convex shapes. This adaptive approach allows V-HACD to better preserve the geometry details of the original model.

From the dataset of objects, two examples were selected for illustration. The shapes of these objects are depicted in Figures 4.16 and 4.17 before and after the conversion to convex shapes. Through trial and error during the conversion phase, it was determined that the second method is suitable for objects with hollow spaces, while the first method is more appropriate for objects without such spaces. To elaborate, the first method involves a straightforward conversion process, ideal for solid objects without cavities or complex internal structures. This approach ensures that the object's surface is uniformly convex, facilitating accurate and stable simulations. Conversely, the second method, designed for objects with hollow spaces or intricate geometries, involves decomposing the object into multiple convex sub-shapes. This decomposition ensures that even complex objects maintain their structural integrity and interact correctly within the simulation environment.
This process was applied to all the objects in the dataset. To ensure that objects would not change during their addition to the scene, they were saved as CoppeliaSim models in the TTM format after being converted to convex shapes. This step ensures that the objects maintain their integrity and are correctly represented in the simulation environment. The dataset generated after completing these steps is shown in Figure 4.18. By saving the objects in the TTM format, we ensure consistency and accuracy in the simulation, which is crucial for effective training of the DRL agent. This comprehensive approach to dataset

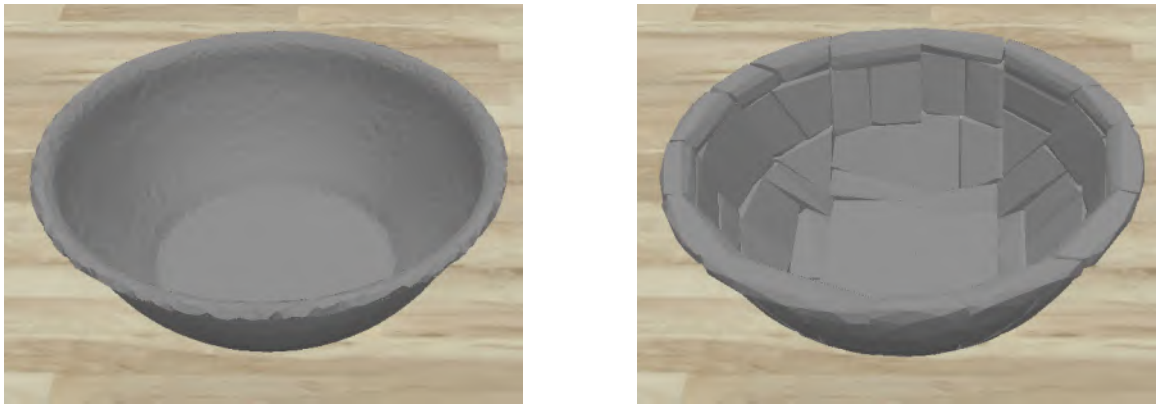


(a) Non-convex Banana *(b) Convex Banana*

Figure 4.16: Conversion to convex shape for a banana using HACD

preparation significantly contributes to the robustness and reliability of the smart grasping system.

In this section, the process of creating the desired dataset was detailed. To ensure the DRL agent learns effectively, a variety of objects with different sizes and shapes were selected. These objects were then imported into the simulation software. To guarantee proper functioning of the physics engine during simulation, all objects were converted into one or more convex shapes. These convex shapes were subsequently saved as simulation models in the TTM format, ensuring they remain unchanged when loaded into the main scene.



(a) Non-convex Bowl *(b) Convex Bowl*

Figure 4.17: Conversion to convex shape for a bowl using V-HACD

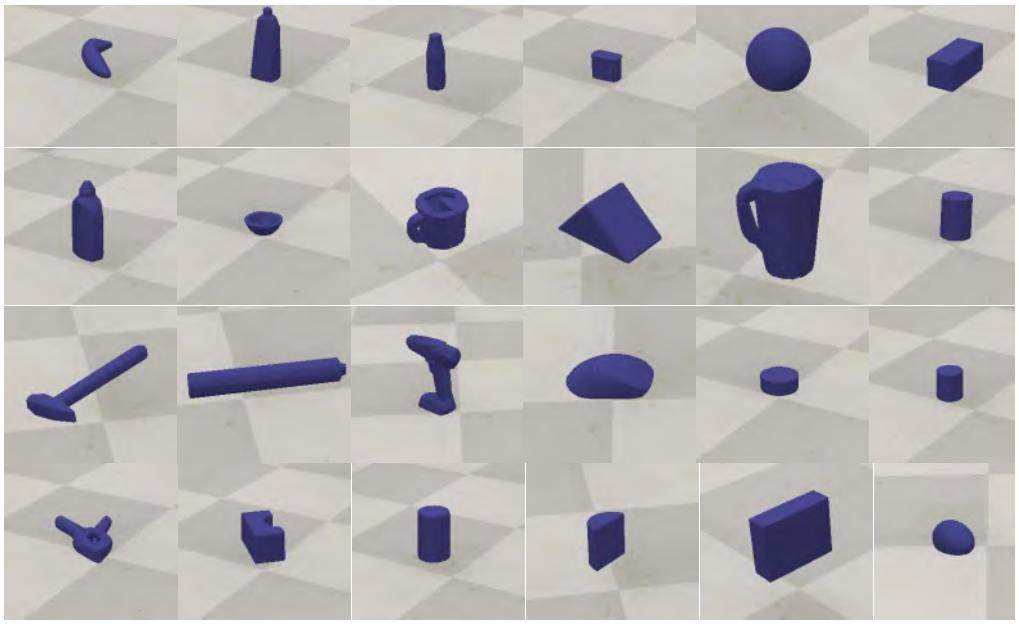


Figure 4.18: Object Dataset

4.7 Creating Simulation Environment For Deep Reinforcement Learning

As previously mentioned, the environment is the only external entity with which the DRL agent interacts. The task is defined within this environment, and the agent aims to learn an optimal policy through interaction to maximize its reward. This environment can either be the real world or a simulated model of it. In this section, the process of creating a novel environment from scratch for the smart grasping task is explained in detail.

First, the interface and framework used to create this environment are described. This includes the software and tools employed to model the various components such as the robot, gripper, and vision sensor. Following this, the training phase is discussed, outlining how the agent interacts with the environment and the iterative process of learning.

The discussion then moves to state space and action space definition, crucial elements for the DRL agent's decision-making process. Reward shaping, which involves designing the reward system to guide the agent towards optimal behavior, is also elaborated on. To enhance training efficiency, techniques such as adding noise to the environment and using parallel environments are explored. These methods help the agent generalize better and learn more robust policies.

As discussed in Chapter 3 and shown in Figure 3.1, the interaction between the agent and environment, known as the DRL process, operates as follows: the agent receives the state s_t from the environment, and based on its policy π, selects an action a_t. aAs a result

of this action, the environment changes, and the agent subsequently receives a new state s_{t+1} and and a reward r_{t+1}. his cycle continues until the environment reaches a terminal state. Consequently, a significant amount of data is transferred over an extended period during the training process.

To effectively manage the interaction between the agent and the environment, it is crucial to employ a standardized Application programming interface (API) for this communication. An API is a method used in software engineering to enable two or more programs or components to interact with each other. In this thesis, one of the popular Python libraries for this purpose, called Gymnasium[86], was utilized. This widely-used library provides a standardized approach to creating environments that can communicate seamlessly with DRL models.

The standard method for creating an environment using Gymnasium involves creating a subclass of the original Gymnasium environment class. Within this subclass, several essential methods must be defined: $reset$, $step$, $get_observation$, get_reward, and termination. Since the environment is considered fully observed, the terms "state" and "observation" are used interchangeably in this thesis. These methods correspond to the components of the DRL process discussed previously.

The reset method initializes the environment to its starting condition and returns the initial observation. This step defines the initial conditions of the environment. For example, in this step, a random object from the object dataset is selected and placed at a random position within the robot's workspace. The step method is invoked whenever the agent takes an action. This method receives the action, executes it, and the environment undergoes a change. After performing the action, the reward is determined, and along with the new state, the method's output is formed.

The $get_observation$ and get_reward methods are responsible for determining the state and reward, respectively, whenever needed. The reward and state are defined within these methods. The reward function, which will be explained in the following paragraph, is a crucial component of this process. Finally, the termination method evaluates whether a termination condition is met. If it is, the episode ends after the step method is called.

Another critical feature of this class is the definition of the action and observation spaces. These spaces are specified in the initialization section of the class. It is important to note that, based on training results, these spaces might undergo modifications. At this stage, the spaces are defined in their general form, with specific details provided alongside the results. The action space is a four-dimensional continuous space, as illustrated in Table 4.4. In this table, the parameters $\Delta_{translation}$ and $\Delta_{rotation}$ are fixed during a training session but can be adjusted before it. These parameters define how much the end-effector

Table 4.4: Type 1 Action Space Definition

Number	Action	Minimum Value	Maximum Value	Range	Name
1	Linear translation of the end-effector in X direction	-1	+1	$\Delta_{trasnlation}$	d_x
2	Linear translation of the end-effector in Y direction	-1	+1	$\Delta_{trasnlation}$	d_y
3	Linear translation of the end-effector in Z direction	-1	+1	$\Delta_{trasnlation}$	d_z
4	Rotational translation of the end-effector in Z direction	-1	+1	$\Delta_{rotational}$	d_r

moves for a given action. For example, if $\Delta_{translation} = 10\,cm$ and the agent takes action number one to the maximum, the end-effector moves 10 cm in the X direction.

The previously discussed action space pertained to translational motion. Another possible type of motion is position-based motion, where the agent determines the gripper's placement rather than the distance it moves along a specific axis. The action space for this type of motion is defined in Table 4.5. In this table, R_x, R_y, R_z, and R_{rZ} represent

Table 4.5: Type 2 Action Space Definition

Number	Action	Minimum Value	Maximum Value	Range	Name
1	Linear position of the end-effector in X direction	-1	+1	$\pm R_z$	p_x
2	Linear position of the end-effector in Y direction	-1	+1	$\pm R_y$	p_y
3	Linear position of the end-effector in Z direction	-1	+1	R_z	p_z
4	Rotational orientation of the end-effector in Z direction	-1	+1	R_{rz}	p_r

the range of position or rotation for each dimension. The implementation of the resulting action space is illustrated in Figure 4.19.

Figure 4.19: Observation space definition when using image

```
import numpy as np
from gym import spaces
self.action_space = spaces.Box(low = -1, high = 1,
shape = (4,), dtype = np.float32)
```

For the objectives of this thesis, two distinct observation spaces have been considered, and training will be undertaken using both spaces, followed by a comparative analysis of the outcomes. Modifications to these spaces may be introduced based on the results observed during the training process. The initial observation space comprises a depth image along with several vectors, while the secondary observation space involves considering 2D points representing the object's perimeter. Part of the observation space includes a vector detailing the positions of the object and the gripper, as illustrated in Table 4.6. The extended portion

Table 4.6: Observation vector primary vector

Num	Observation	Min	Max	Unit
1	Component X of Gripper's absolute position	$-\infty$	∞	meter
2	Component Y of Gripper's absolute position	$-\infty$	∞	meter
3	Component Z of Gripper's absolute position	$-\infty$	∞	meter
4	Gripper's spinning angle	$-\infty$	∞	radian
5	Component X of Object's absolute position	$-\infty$	∞	meter
6	Component Y of Object's absolute position	$-\infty$	∞	meter
7	Component Z of Object's absolute position	$-\infty$	∞	meter
8	Component X of relative position between object and gripper	$-\infty$	∞	meter
9	Component Y of relative position between object and gripper	$-\infty$	∞	meter
10	Component Z of relative position between object and gripper	$-\infty$	∞	meter
11	Component X of closest distance between object and gripper	$-\infty$	∞	meter
12	Component Y of closest distance between object and gripper	$-\infty$	∞	meter
13	Component Z of closest distance between object and gripper	$-\infty$	∞	meter
14	Object's height	$-\infty$	∞	meter

of the observation space incorporates either a depth image or a set of 2D points appended to the primary observation vector. When employing a depth image, a vision sensor is typically positioned on the ground. At the beginning of the simulation, the depth image captured by this sensor is extracted and undergoes preprocessing. Initially, the image array is reshaped to match the camera's width and height, followed by multiplication by 255 to standardize the image array. Consequently, the image becomes a one-channel array with a specific

resolution. The resulting observation space is depicted in Figure 4.20. When integrating

Figure 4.20: Observation space definition when using image

```
import numpy as np
from gym import spaces
self.observation_space = spaces.Dict(
{"image": spaces.Box(low=0, high=255,
shape=(256,256), dtype=np.uint8),
"vector": spaces.Box(low = -np.inf,
high= np.inf, shape=(14,), dtype=np.float32)})
```

the image directly into the observation space, the reinforcement learning model employs the "MultiInputPolicy". In this approach, the observation space is structured as a dictionary where the 'image' key contains the image itself, and the 'vector' key encompasses other scalar values.

Alternatively, another approach involves extracting crucial details from the depth image and then integrating them into the observation space. This methodology is detailed in Section4.5, and the constructed observation space is depicted in Figure 4.21.

Figure 4.21: Observation space definition when using 2D Points

```
import numpy as np
from gym import spaces
self.observation_space = spaces.Box(low = -np.inf, high = np.inf,
shape = (obsvDimension,), dtype = np.float32)
```

The subsequent component of the environment concerns the reward function, which evaluates the effectiveness of the agent's actions. Typically, this function is defined within the step method. However, if it becomes intricate or is required by other methods, it is advisable to define it separately. Reward functions commonly fall into two categories: dense and sparse. Dense rewards are provided at each time step or action, offering detailed feedback on the action's quality and aiding the agent in comprehending its decisions' impact. Sparse rewards, on the other hand, are less frequent and are typically linked to specific events, conveying limited information about trajectory outcomes, whether positive or negative. Despite their potential to slow down training and provide minimal information, sparse rewards are crucial for indicating whether key objectives have been accomplished.

To harness the benefits of both reward types, this thesis adopts a combined approach. For dense rewards, two distinct terms are employed: distance reward and time penalty. Sparse rewards encompass three terms: successful grasp, grasp quality score, and collision penalty. Unlike other components in the DRL process, the reward function requires tuning during training to enhance performance. Hence, a parameterized reward function is utilized, with multiple candidate functions explored.

The primary focus lies on the distance reward term within the reward function, which incentives the agent to approach the target object. This term calculates the distance between the gripper and the object at each time step and maps it to the reward function space using mathematical functions. Specifically, two functions are employed: the inverse hyperbolic tangent and Sigmoid functions, expressed as follows:

$$r_d = 1 - \tanh\left(K|\mathbf{P}_{\text{Object}} - \mathbf{P}_{\text{Gripper}}|\right) \tag{4.7.1}$$

$$r_d = 1 - \frac{1}{1 + e^{-K|\mathbf{P}_{\text{Object}} - \mathbf{P}_{\text{Gripper}}|}} \tag{4.7.2}$$

In these equations, the expression $|\mathbf{P}_{\text{Object}} - \mathbf{P}_{\text{Gripper}}|$ denotes the Euclidean distance between the gripper's palm center and the object's center of mass, where K serves as the distance mapping coefficient. This parameter governs how distance values are translated in the reward calculation.

Moving on to the subsequent component of the dense reward framework, the time penalty imposes a cost on the agent for each time step exceeding a predefined threshold within an episode. This penalty aims to prompt the agent to complete tasks efficiently. Mathematically, this term is represented as:

$$r_t = -1 \qquad \textbf{if} \quad t_{elapsed} > T_{max.time} \tag{4.7.3}$$

In the equation, $t_{elapsed}$ represents the total duration of the episode, and $T_{max.time}$ denotes the threshold beyond which penalization begins.

The next aspect crucial to the reward system is the successful grasp reward r_s, which is granted to the agent upon successfully completing the grasping task. A significant challenge in implementing this reward is defining what constitutes a successful grasp and devising a reliable method to detect it. In existing literature, a commonly proposed technical solution involves attempting to lift the object when the gripper is closed. If the object's height increases during this lifting phase, it indicates a successful grasp. However, applying this method in our specific environment proved unfeasible due to limitations inherent in the gripper model employed.

To address this challenge, a novel method was developed to accurately define and detect successful grasps. This method also facilitates distinguishing between two distinct phases of the grasping process. In Phase 1, or the approach phase, the agent moves towards the object and positions the gripper optimally for grasping. A proximity sensor installed on the gripper's palm detects the presence of the object, signaling the initiation of Phase 2: gripping. Once the gripper fully closes, a successful grasp is confirmed if there are at least three contact points between the gripper and the object. This criterion is expressed

mathematically as:

$$r_s = \begin{cases} 1 & \textbf{if} \quad \text{Successful Grasp} \\ 0 & \textbf{otherwise} \end{cases} \tag{4.7.4}$$

The subsequent component in the sparse reward framework is the grasp quality score r_q. This metric assesses the quality of the grasp by assigning a numerical score based on how well the grasp is executed. Various methods for evaluating grasps have been proposed in the literature[87], but due to constraints in the action space, the most viable option was the scoring method introduced by Chinellato et al.[88]. This approach evaluates grasp quality by measuring the distance between the centroid of the contact polygon and the object's center of mass. However, this measure required adaptation to fit our specific environment. Modifications were implemented to integrate it effectively, resulting in the proposed function outlined in Figure 6.

Algorithm 6 Proposed Grasp Quality Score Function

Given object's center of mass C
Given centroid of the contact polygon CM
Given object height H

1: $d = Dist(CM, C)$
2: **if** $d < H/2$ **then**
3: $\quad r_q = -2\frac{d}{H} + 1$
4: **end if**
5: **else:** $r_q = 0$

The final component of the reward function is the collision penalty r_c. Throughout the training process, unintended collisions can occur between the gripper and the ground, or between the gripper and the object. These collisions are undesirable as they can disrupt task execution. To mitigate such incidents and provide feedback to the agent, a penalty term is incorporated. If the gripper makes contact with the ground or strikes the object in a manner that displaces it, the agent incurs a penalty.

An additional consideration in designing the reward function is that not all terms contribute equally to achieving the desired goal. For example, achieving proximity to the target (distance reward) holds greater significance than minimizing the time taken (time penalty). Hence, the reward function terms are parameterized accordingly:

$$\text{Reward} = w_d r_d + w_t r_t + w_c r_c + r_s r_q \tag{4.7.5}$$

These weights undergo tuning throughout the training phase to optimize outcomes.

Another critical aspect of the environment involves specifying termination conditions. As discussed in the RL process, an episode continues until certain conditions prompt its

conclusion. Defining termination conditions within the environment ensures that when these conditions are met, the episode concludes. The termination conditions in this environment are outlined below:

- **Maximum Time Step:** When the number of elapsed time steps in an episode reaches a specified threshold known as the maximum number of time steps, the episode terminates.
- **Collision:** If an unintended collision occurs between the gripper and the ground or the object, this condition triggers termination of the episode.
- **Successful Grasp:** A successful grasp, defined by the criteria established in reward shaping, results in a successful episode termination.

These termination conditions are implemented within a dedicated method in the environment and are evaluated after each invocation of the step method.

Another crucial component of the smart grasping framework is the agent itself. Chapter 3 provided the necessary background on (DRL), but did not delve into the specifics of implementing these algorithms. Implementing modern DRL algorithms demands a strong foundation in mathematics and software engineering, along with considerable time investment.

To circumvent these challenges, leveraging existing and reliable implementations of these algorithms is advisable. One widely used library for this purpose is Stable-Baselines3 (SB3), as introduced in [89]. This library offers robust implementations of various DRL algorithms designed to integrate seamlessly with Gymnasium environments. Each algorithm within SB3 faithfully reproduces the original research papers' methodologies. Moreover, it provides flexibility for users to customize algorithms or policies according to specific requirements.

Now that all components of the simulation framework have been addressed, the framework itself can be constructed. The detailed diagram of the simulation framework is depicted in Figure 4.22. This diagram illustrates the creation of three main Python scripts comprising over 3000 lines, essential for managing the entire training process. The first component depicted in the diagram is the simulation scene developed within the CoppeliaSim software environment. This scene incorporates essential elements such as the robot, gripper, camera, and various objects. Implemented as an XML file, this scene is visually represented in the graphical interface shown in the diagram. Following the simulation scene is the CoppeliaSimClass Python script. This script functions as a Python class specifically designed to interact with the simulation scene using the ZeroMQ [90] remote API. It encapsulates commands for controlling the simulation environment, including robot movement, gripper

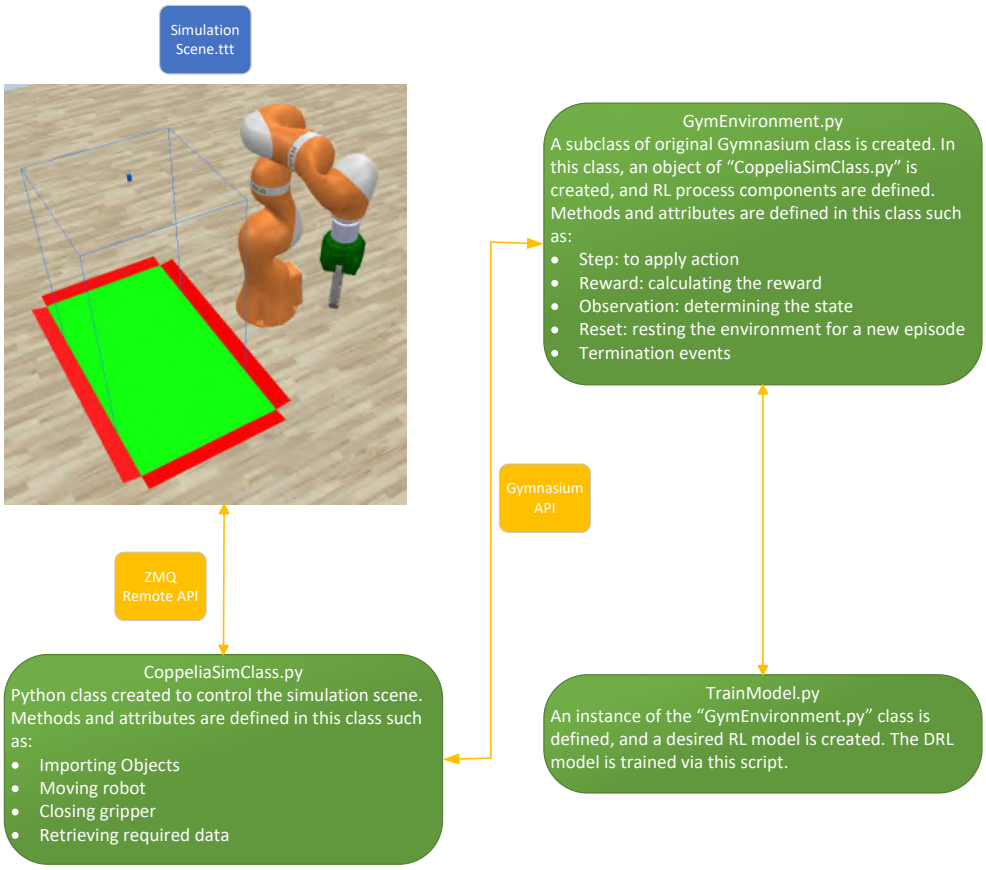


Figure 4.22: Detailed Simulation Framework Diagram

operation, object importation, and data retrieval.

The next script is the GymEnvironment, a subclass of the Gymnasium class. Within this class, an instance of the CoppeliaSim class is instantiated, and various components of the DRL process are established. These components include defining methods for handling steps, computing rewards, generating observations, and resetting the environment for new episodes.

The final script, TrainModel, interfaces with the GymEnvironment through the Gymnasium API. This script involves creating the desired DRL model, which is then trained and saved for future use. The training process is depicted in detail in Figure 7, outlining the steps and algorithms employed during model learning. As depicted in Figure 7, the training process commences with initializing the model network and establishing remote connections between the simulation scene and the environment. Subsequently, for a specified number of training steps and at the start of each episode, a random object is positioned at a randomized location within the scene. Depth information relevant to the object is then

Algorithm 7 Training Process

Initializing model networks
Establishing remote connections

for n in range number of training time steps **do**
    Import a random object to a random position
    Get required depth information
    while *not* *terminationCondition* **do**
        Get state from environment S_t
        Agent takes action a_t
        Action a_t is performed in the environment
        Agent gets S_{t+1} and r_{t+1}
        Update network parameters Θ
    end while
    if *Evaluation* then
        Perform evaluation and save the model
    end if
end for

captured. Following this setup phase, the DRL process initiates by retrieving the state s_t, from the environment and executing action a_t. This action alters the state of the environment, prompting the agent to receive a new state s_{t+1} and a corresponding reward r_{t+1}. Based on these outcomes, adjustments are made to the parameters of the model networks. This iterative process continues until the episode concludes. Upon completing the episode, if it is an evaluation phase, the model is assessed and subsequently saved. The entire training cycle persists until the maximum number of time steps is reached.

4.8 Validation Methods

In this section, the validation methods used to verify the simulation model, addressing the second objective, are explained. To mitigate Sim2Real challenges, it is essential to validate the components and their models. First, the method used to validate the final assembly of the robot and the gripper is detailed. Next, the method used to validate the kinematic model of the robot is explained, and the approach used to control the real robot is discussed. Finally, the methods employed to validate the proposed novel model of the gripper are examined, along with the approach for controlling the gripper.

4.8.1 Final Assembly Validation

In this section, the validation process of the final assembly of the robot and the gripper is described. The gripper and the robot are two distinct devices in the simulation, with the gripper mounted on the robot using a flange and a coupling. The flange is provided by the robot's manufacturer, and the coupling is provided by the gripper's manufacturer. The gripper was specifically designed for use with another company's robots, resulting in

the coupling lacking an accurate drawing. An accurate drawing was necessary to determine the relative distance between the gripper's frame and the tool frame defined in the robot's software. Initially, this offset was determined using the available drawings, but it was later discovered that the dimensions used were inaccurate. To resolve this issue, the relative position between the gripper's frame and the robot's built-in frame needed to be precisely determined. Traditional measuring tools were inadequate for this task because the exact locations of the robot's frames were unknown or inaccessible. Therefore, a modern measuring tool based on a laser tracker system was employed to achieve the required accuracy.

The laser tracker used was the **FARO** Vantage Laser Tracker, designed for fast and accurate measurements in large-scale applications. This system operates based on a spherical coordinate system, where the position of each point is represented using two angles and one distance. These parameters are then converted to the Cartesian coordinate system to determine the XYZ values. To measure the spherical coordinate system parameters, the laser system tracks a target by capturing and following a laser beam it emits. The laser beam, emitted by the tracker and reflected by a prism inside the target, must remain within the source. The tracker has two motors, and the encoders in these motors determine the angles. To measure the distance, the travel time of the reflected laser beam is calculated. Once the spherical coordinates are measured, the XYZ coordinates are determined.

To find the relative position of the gripper's frame with respect to the robot's world frame, several geometric features were defined. These features are depicted in Figure4.23. First, the XY plane of the robot's base frame was considered, and several points were taken to establish this plane. Then, by measuring three points on the circular base of the robot, the center of the base was determined. The next step was to define the direction of one axis. The X-axis was established using the edge of the flange between the robot and the table. With the defined plane, the center, and the direction of the X-axis, the robot's world frame could be constructed. The next geometric feature to define was the gripper's body frame. Since the offset is in the Z-direction, finding the exact center of the frame was unnecessary; any point on the XY plane sufficed. With the coordinate system and the defined point on the gripper, the position of this point in the defined frame was determined using the laser tracker software. This data was then compared to the data provided by the robot's software to determine the actual offset value. This offset value was later used to modify the simulation model and validate the robot's kinematics model.

4.8.2 Validation of the Kinematic Model of the Robot

In the simulation environment established for the reinforcement learning agent, a kinematic model of the robot was used to move the robot to the desired position and orientation. This kinematic model must be validated before the training phase begins to ensure that the discrepancy between the simulation environment and the actual environment is minimized.

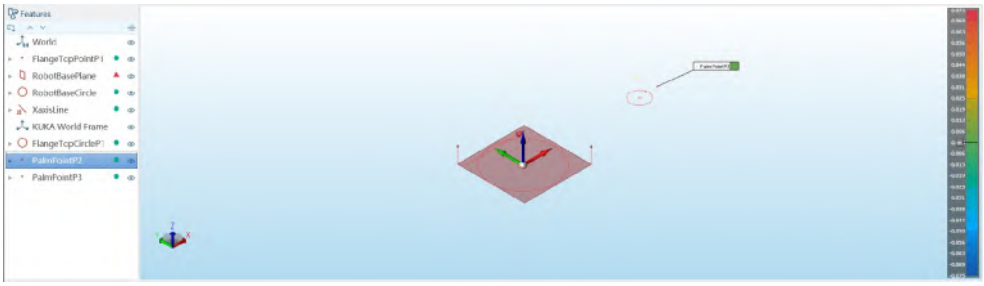


Figure 4.23: Geometric features in CAM2

The method to validate this model involves moving the end-effector along a specified path and then comparing the joint and end-effector positions between the simulation and the actual robot. To perform the validation, the simulation model and the real robot run simultaneously. The joint positions and the end-effector's pose are then compared between the simulation model and the actual robot. The robot followed two different paths: the action space representation and a circular path. Data gathered from the robot and the captured video provide both quantitative and qualitative validation of the kinematic model.

Before outlining the steps to validate the kinematic model, the approach used to control the actual robot is discussed. During the validation phase, or during the Sim2Real phase, we must be able to control the robot, which involves sending commands to and receiving signals from the robot. For the KUKA robot used in this thesis, there are two main methods to control the robot:

1. Using the smart pad provided by the manufacturer
2. Using an external computer

The first method is suitable for training and testing purposes. In this method, a human operator can manually send commands or read sensor values using the graphical interface on the smart pad. While this method is straightforward, it requires a human operator, which is not feasible for this project's application where high-frequency data transfer between the robot and the agent is needed. Therefore, the second method, controlling the robot using an external computer, was used.

In the second method, an external computer is used to control the robot, eliminating the need for the smart pad. To achieve this, SunriseWorkbench, a software provided by the manufacturer, is utilized. This software allows the installation of an application on the robot that facilitates a connection between the robot and the external computer. Once this application is installed, a connection can be established whenever the robot needs to be controlled.

On the external computer, the established connection can be managed using Matlab

or Python. Consequently, the robot can be controlled through a Python script, simplifying the implementation of novel applications. For instance, the reinforcement learning agent can be trained with the real robot instead of in the simulation. The application installed on the robot, developed by [91], provides a library to utilize the robot's features. However, it doesn't support all the robot's functionalities necessary to control the robot with the desired tools installed. For instance, the library supports only a specific kind of flange, which differs from the flange mounted on the robot in our lab. Additionally, it does not allow modification of coordinate frames or selection of different frames. Another limitation is that the features of the flange cannot be utilized, meaning the gripper cannot be controlled through the robot controller. These limitations can pose significant issues when evaluating the performance of a trained model.

The KUKA robot boasts numerous powerful functionalities, making it an excellent choice for collaborative tasks where the robot must work safely alongside humans. These features include both motion control and force control. In this project, only motion control is required. For motion control, there are three distinct methods to move the robot:

1. **Point-to-Point (PTP) mode:** In this mode, the final point is given either in joint space or task space, and the robot determines the optimal trajectory to reach that point.

2. **Direct Servo mode:** In this mode, the robot follows the given trajectory in real-time, either in joint space or task space. The controller aims to achieve the desired positions and orientations as accurately and quickly as possible.

3. **Impedance mode:** The robot follows the given path while behaving like a spring-damper system, allowing it to handle interactions with objects.

The choice among these options depends on the specific application. If the robot's environment is secure with a low likelihood of contacting other objects, then the PTP mode is the optimal choice. Additionally, if the path is predefined and there is a low probability of collision, the Direct Servo mode is preferred. In scenarios where humans work alongside the robot and there are other objects in the vicinity, the impedance mode should be activated to enhance safety. For this project, given that the robot's task space is confined to the action space and no human is intended to work alongside the robot, the PTP mode has been selected.

By opting for the PTP mode in joint space, the actual robot follows the given trajectory at each step. Essentially, the programming scripts for the simulation and the actual robot remain identical, with the sole distinction being that the commands are directed to the physical robot instead of the simulation.

Table 4.7: Compared Kinematic Model Parameters

Parameter	Symbol	Unit
Joint Number 1 Position Error	E_{θ_1}	rad
Joint Number 2 Position Error	E_{θ_2}	rad
Joint Number 3 Position Error	E_{θ_3}	rad
Joint Number 4 Position Error	E_{θ_4}	rad
Joint Number 5 Position Error	E_{θ_5}	rad
Joint Number 6 Position Error	E_{θ_6}	rad
Joint Number 7 Position Error	E_{θ_7}	rad
End-effector Position Error Along the X-axis of the Robot Base Frame	E_{P_x}	m
End-effector Position Error Along the Y-axis of the Robot Base Frame	E_{P_y}	m
End-effector Position Error Along the Z-axis of the Robot Base Frame	E_{P_z}	m
End-effector Orientation Error: XYZ Euler Angle Roll	E_α	rad
End-effector Orientation Error: XYZ Euler Angle Pitch	E_β	rad
End-effector Orientation Error: XYZ Euler Angle Yaw	E_γ	rad

The next step is to develop a method to validate the kinematic model. In this case, two different trajectories are defined. The first trajectory is termed "action space," where the action space is divided into numerous points along the X, Y, and Z directions, and the end-effector moves through all these points. The second trajectory is a planar circular path, where the end-effector follows a circular path. Once both the simulation model and the actual robot complete the given trajectories, the kinematic data between the two are compared to achieve quantitative validation. Additionally, a video is captured from both models to provide qualitative validation. For quantitative validation, the differences between each parameter of the kinematic model at each step are calculated to determine the model's error. Subsequently, the Root Mean Squared Error (RMSE) is calculated for each parameter across all the steps. The parameters used for comparison are shown in Table4.7.

The next step involves determining or reading these validation parameters. The process of reading sensor data varies for different parameters. For the joint positions, both the simulation and the robot provided accurate position data from the joints, which can be easily compared. However, obtaining the end-effector pose presented some challenges. Pose data from the simulation model was retrieved using the built-in functions of the physics engine. For the actual robot, the desired frame differed from the robot's defined frame.

The frame defined by KUKA was located at the center of gravity of the electric flange installed on the robot. This frame differed from the gripper's frame, and the installed package on the KUKA could only read the default frame data. An attempt was made to define a custom frame for KUKA's software, but using the software provided by the company requires expert knowledge in software engineering. To solve this issue, the transformation matrix between the default frame and the gripper's frame was found using the laser tracking system discussed in the previous section. This transformation matrix is shown in Equation 4.8.1.

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & d \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.8.1}$$

In this matrix, θ is the rotation along the X-axis of the tool frame, and d is the distance along the Z-axis between the flange's center of gravity and the gripper's palm, which equals 124.02 mm. The pose given by the robot was transformed into the palm's pose using this transformation matrix.

The next step is determining the error. For each step, the error for each parameter is determined by subtracting the simulation value from the actual value. These errors are plotted with respect to the step for each parameter. Then, for each parameter, the RMSE value is determined using the following equation:

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_{real} - x_{simulation})^2} \tag{4.8.2}$$

where N is the number of steps, x_{real} is the value of the actual parameter, and $x_{simulation}$ is the value of the simulation parameter.

4.8.3 Gripper Validation

This section outlines the methods employed to validate the accuracy and reliability of the simulation model for the Robotiq3F gripper. Validation is a critical step before utilizing the model in further research, as it ensures the simulated behavior closely resembles the real-world gripper's performance. Without proper validation, the results obtained from the simulation could be misleading and unreliable, potentially impacting subsequent research conclusions.

The validation approach combines qualitative and quantitative methods applied across two distinct scenarios. The first scenario focuses on the gripper's behavior when no object is grasped by its fingers. A close command is sent to the gripper in both the simulation environment and the physical experiment setting. Videos are captured from both the simu-

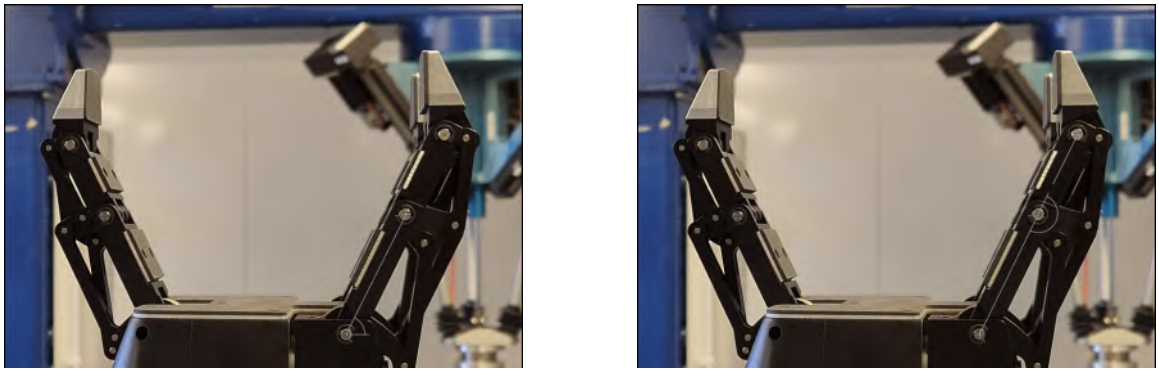
lated and real gripper during this closing motion. Simultaneously, joint positions within the simulation are recorded at specific intervals. Once the gripper is fully closed in both environments, the captured video from the real gripper is analyzed to retrieve the corresponding joint positions. This retrieval process involves capturing frames at a predetermined frame rate from the video. Subsequently, joint positions for all links of the gripper are measured meticulously using image processing software like GIMP. The second scenario mirrors the first but introduces an object with specified dimensions and a defined relative pose within the gripper and experiment setup. Following object placement, the gripper is closed again, and the same data measurement process, involving video capture, frame extraction, and joint position measurement, is applied to both the simulated and real gripper data.

Beyond simply comparing data between the simulation and experiment, uncertainty within the measured data is also investigated. Two primary types of numerical data are considered in this analysis: joint positions acquired from the simulation and experiment. Simulated joint positions are considered deterministic and error-free, as the physics engine's accuracy has been previously validated through rigorous testing[72]. Conversely, joint positions measured from the experiment can be considered probabilistic and potentially contain errors due to various factors.

To exemplify the concept of measurement uncertainty, consider the process for analyzing a single image captured during the gripper's closing motion. Two primary sources of error can potentially influence the accuracy of the measured joint positions. The first source of error arises from potential misalignment between the camera used to capture the video of the real gripper and the gripper itself. This error, prevalent in many vision applications and known as "tilted camera," can introduce inaccuracies into the measurement process. To mitigate this error source, a level is used during the experiment setup to minimize the angle between the camera's field of view and the surface of the object being grasped (or, in the case of the first scenario, the surface of the gripper finger). The second source of error stems from the measurement process itself, where an operator selects a specific point on the image corresponding to a joint on the gripper. This selection process inherently introduces human error due to the limitations of human perception and hand-eye coordination.

The number of points selected per image can vary depending on the specific link of the gripper being measured. Three angles are measured for each image, corresponding to the three individual fingers of the gripper. The image processing software allows for measuring angles based on either two or three points. Selecting two points yields more accurate results for horizontal or vertical orientations, while three points are necessary for angles between lines. As illustrated in Figure 4.24, only two points are required for link one (the base link of the gripper), while three points are needed for link two (the first joint connecting a finger to the base link). Consequently, based on the measurement methodology, higher accuracy is

anticipated for link one measurements compared to link two measurements. To address the



(a) Two-point Measurement *(b) Three-point Measurement*

Figure 4.24: Experimental Angle Measurement Method

issue of measurement error and improve the overall reliability of the validation process, five measurements are taken for each link at each step (i.e., for each captured video frame). The validation process utilizes the average of these five measurements as the representative joint position for that specific point in time. Additionally, the standard deviation is employed as an uncertainty metric to quantify the measurement accuracy. A lower standard deviation indicates a higher degree of consistency among the five individual measurements, suggesting greater confidence in the average value used for validation.

An additional challenge arose during model validation: the data acquisition frequencies differed between the simulation and experiment, hindering direct error calculation for each step. For example, in the "without object" scenario, 202 data points were recorded from the simulation, while only 22 points were captured from the experiment. To address this discrepancy, both datasets were linearly resampled using interpolation techniques.

4.9 Conclusion and Contributions

This chapter details the methods developed and implemented throughout this research. The chapter is structured into three key areas: robot motion planning, gripper modeling, and environment creation. A novel motion planning algorithm was proposed for the robot. This research introduces the first-ever implementation of the Shimizu solution[74] for non-offset redundant manipulators in Python and the CoppeliaSim simulation environment. This implementation allows the robot to efficiently navigate its workspace and achieve desired goals. This chapter also presents a groundbreaking kinematic model for the Robotiq3F gripper. This model is the first of its kind and utilizes a state machine to accurately represent the gripper's behavior during various grasping tasks. Following the development of the individual models, all components, including the robot motion planning algorithm and the gripper kinematic model, were combined into a single simulation software. This integration

resulted in the creation of a novel reinforcement learning (RL) environment specifically designed for smart grasping tasks. This environment allows researchers to explore and develop grasping strategies for three-finger grippers within a simulated setting. To ensure the reliability and effectiveness of the developed environment for smart grasping tasks, validation methods are proposed in this section. These methods will be used to assess the accuracy and performance of the entire system.

Chapter 5

Results

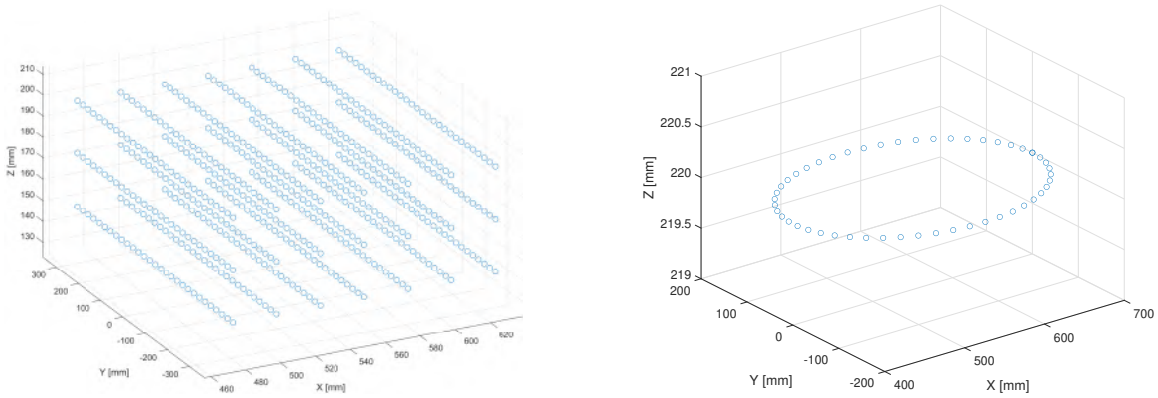
This chapter presents the results of three parts: the robot kinematic model validation, the Robotiq 3F-gripper kinematic model validation, and the deep reinforcement learning model. Each section will detail the obtained results and their corresponding discussion.

The initial section concentrates on validating the robot's kinematic model. Chapter 4 established the kinematic model for the robot, followed by a validation method to confirm its accuracy. This section provides the validation results accompanied by a thorough discussion. The subsequent section presents the validation results for the Robotiq3F gripper kinematic model. Finally, the third section delves into the training results of the deep reinforcement learning model. This section will detail the training process, including the chosen parameters and performance metrics used to evaluate the model's effectiveness. The model's ability to achieve the set goals and its overall performance was explored.

5.1 Robot Kinematic Model Validation Results

This section presents the validation results of the robot's kinematic model, comparing simulation model's accuracy compared to the actual robot. As detailed in Section 4.8.2, two predefined paths, shown in Figure 5.1, were used for validation: a linear motion across the action space and a circular path. The simulation model and the physical robot were instructed to follow these trajectories simultaneously. Subsequently, kinematic validation parameters from both the simulation and the experiment were compared. The RMSE was calculated for all steps to quantify the discrepancies.

Figure 5.2 illustrates the error between the simulation model and the experiment for the action space path. Figure 5.2 illustrates the error between the simulation model and the experiment for the action space path. Figure 5.2c focuses on the difference in joint positions between the two models. This figure suggests that the joint position control in both models



(a) Action Path Trajectory *(b) Circle Path Trajectory*

Figure 5.1: Designed trajectories

accurately follows the given trajectory, with minimal discrepancies observed. Figure 5.2a depicts the difference in end-effector position over the steps. Similarly, a separate plot compares the end-effector orientations of the two models.

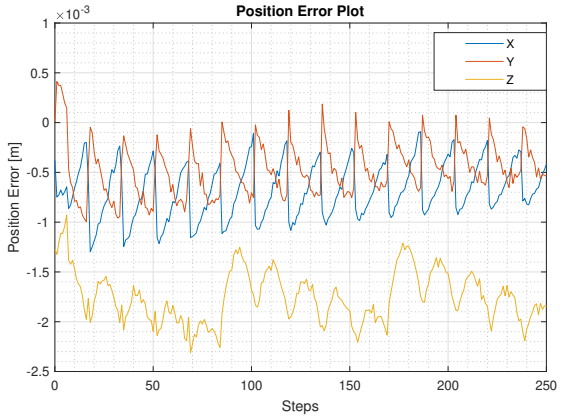
Table 5.1 summarizes the RMSE values for these comparisons. The results indicate RMSE values of 0.0030 $[rad]$ for angle-based parameters and 0.0018 $[m]$ for position-based parameters. These values are acceptable for grasping tasks according to acceptable error values suggested by [92] and [93].

Continuing the previous part's analysis, Figure 5.3 presents the error between the simulation model and the physical robot for the circular path validation. Table 5.2 summarizes the corresponding RMSE values. The results indicate that the model achieves an RMSE of 0.0029 $[rad]$ for angle-based parameters and 0.0017 $[m]$ for position-based parameters. These values are acceptable for grasping tasks according to acceptable error values suggested by [92] and [93].

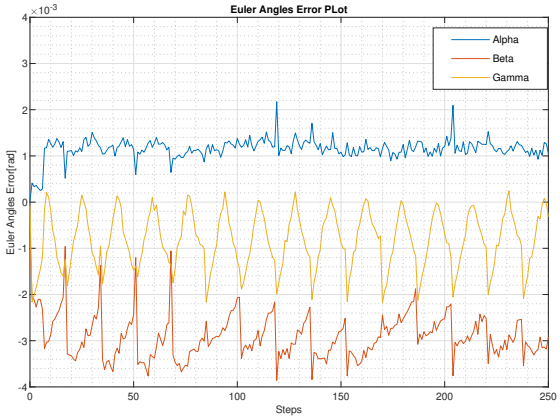
5.2 Robotiq 3F-Gripper Kinematic Model Validation Results

This section presents the validation results for the gripper kinematic model, as outlined in the previous chapter. The validation employed two methods: qualitative and quantitative assessments using various scenarios.

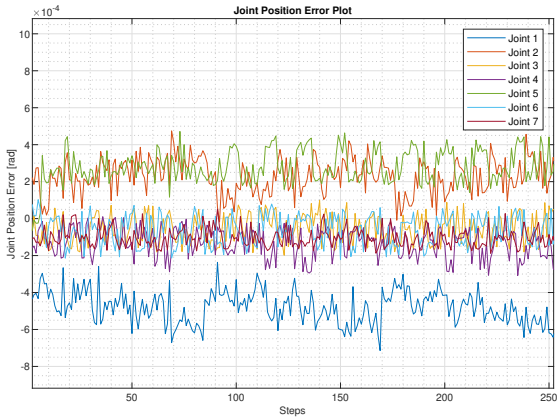
The first scenario, designed for simplicity, involved sending a command to the gripper to be closed in the absence of any object. Subsequent scenarios incorporated two distinct objects for grasping: a book presented in two different orientations, and a bottle. These objects were positioned identically relative to the gripper in both the simulation and the experiment. Following the issuance of a close command to both models, the necessary data was captured. This data encompassed joint positions and image sequences depicting the



(a) Error in end-effector position for linear motion in action space



(b) Error in end-effector orientation for linear motion in action space



(c) Error in joint position for linear motion in action space

Figure 5.2: Error in kinematic parameters for linear motion in action space

finger movements.

Continuing the gripper validation process, we examine the scenario where no object is present (empty gripper). Figure 5.4 compares the image sequences captured from the simulation model and the actual gripper during a closing motion. Qualitative evaluation of these images suggests a high degree of similarity between the simulated and real-world gripper behavior. As previously mentioned, quantitative validation was also conducted. Figure 5.5 presents a comparison of joint positions between the two models. The joint positions are plotted over time steps for both the simulation and the physical gripper. As evident from the figure, the numerical values for corresponding joint positions, as well as the steps at which movement commences or ceases, align closely between the models. This close agreement in joint positions indicates the accuracy of the simulation model for the empty gripper case.

Next, we consider the scenario involving a normal text book. A book with identical

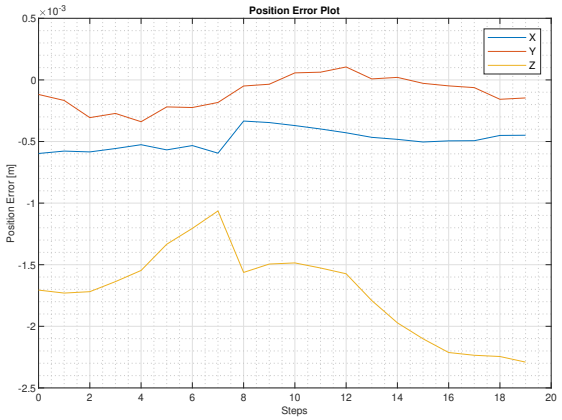
Table 5.1: RMSE Value For Kinematic Model Parameters In Action Space Motion

Parameter	RMSE
Joint Number 1 Position Error $[rad]$	0.0004
Joint Number 2 Position Error $[rad]$	0.0002
Joint Number 3 Position Error $[rad]$	0.0000
Joint Number 4 Position Error $[rad]$	0.0001
Joint Number 5 Position Error $[rad]$	0.0003
Joint Number 6 Position Error $[rad]$	0.0001
Joint Number 7 Position Error $[rad]$	0.0001
End-effector Position Error Along the X-axis of the Robot Base Frame $[m]$	0.0007
End-effector Position Error Along the Y-axis of the Robot Base Frame $[m]$	0.0005
End-effector Position Error Along the Z-axis of the Robot Base Frame $[m]$	0.0018
End-effector Orientation Error XYZ Euler Angle Roll $[rad]$	0.0011
End-effector Orientation Error XYZ Euler Angle Pitch $[rad]$	0.0030
End-effector Orientation Error XYZ Euler Angle Yaw $[rad]$	0.0010

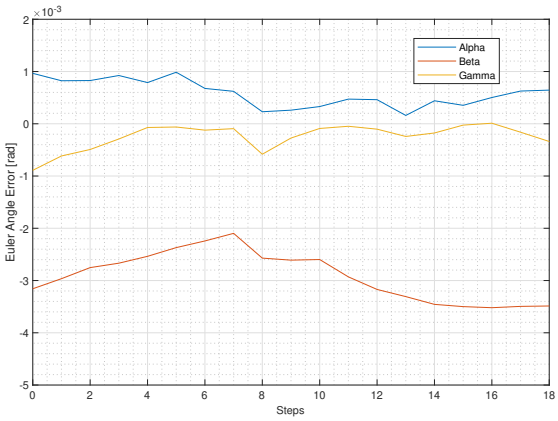
dimensions in both the simulation and physical setup was positioned at the same relative location. Following this placement, a close command was issued. The validation procedure mirrored that of the previous section, with captured image sequences presented in Figure 5.6. Qualitative evaluation of these images suggests a similar configuration of the grippers during the closing motion in the presence of the object. Consistent with the prior analysis, quantitative validation was conducted. Figure 5.7 depicts a comparison of joint positions between the two models over time steps. The close correspondence between the joint position plots for the simulation and physical gripper reinforces the accuracy of the model in grasping scenarios.

The next scenario explores grasping a book in a rotated orientation compared to the previous orientation. The captured image sequences for this scenario are presented in Figure 5.8. As with the previous scenario, joint position plots over time steps are provided in Figure 5.9 for comparison between the simulation and physical gripper.

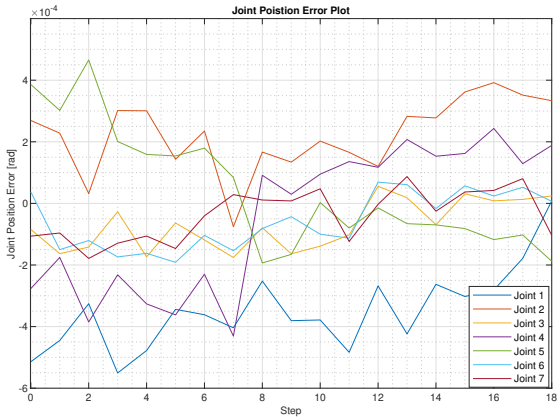
Completing the validation process, we examine grasping a water bottle. The captured image sequences for this scenario are presented in Figure 5.10. Consistent with the prior analyses, joint position plots over time steps are provided in Figure 5.11 for comparison between the simulation and physical gripper. This final scenario allows assessment of the model's ability to handle objects with different shapes.



(a) Error in end-effector position for circle path motion



(b) Error in end-effector orientation for circle path motion



(c) Error in joint position for circle path motion

Figure 5.3: Error in kinematic parameters for circle path motion

To comprehensively evaluate the gripper model's accuracy, RMSE values were calculated for each joint in all validation scenarios. These values, presented in Table 5.3, quantify the average difference between the corresponding joint positions in the simulation and the physical gripper over time steps. As observed in the table, the maximum RMSE value of 8.3 degrees pertains to link 2 in the "without object" scenario. This could be attributed to [brief explanation, e.g., slight differences in initial gripper finger positions or minor sensor noise in the real gripper]. However, it's important to note that all other RMSE values remain below 4 degrees. This range is generally considered acceptable for grasping tasks, as small position discrepancies typically do not significantly hinder the gripper's ability to grasp an object securely.

Furthermore, the consistently low RMSE values across various scenarios, including those involving objects with different shapes and orientations, demonstrate the model's versatility. This suggests that the simulation accurately captures the essential kinematic behavior of the gripper, making it a valuable tool for planning and testing grasping strategies in diverse

Table 5.2: RMSE Value For Kinematic Model Parameters In Circle Path

Parameter	RMSE
Joint Number 1 Position Error $[rad]$	0.0004
Joint Number 2 Position Error $[rad]$	0.0002
Joint Number 3 Position Error $[rad]$	0.0001
Joint Number 4 Position Error $[rad]$	0.0002
Joint Number 5 Position Error $[rad]$	0.0001
Joint Number 6 Position Error $[rad]$	0.0001
Joint Number 7 Position Error $[rad]$	0.0000
End-effector Position Error Along the X-axis of the Robot Base Frame $[m]$	0.0005
End-effector Position Error Along the Y-axis of the Robot Base Frame $[m]$	0.0001
End-effector Position Error Along the Z-axis of the Robot Base Frame $[m]$	0.0017
End-effector Orientation Error XYZ Euler Angle Roll $[rad]$	0.0006
End-effector Orientation Error XYZ Euler Angle Pitch $[rad]$	0.0029
End-effector Orientation Error XYZ Euler Angle Yaw $[rad]$	0.0003

Table 5.3: RMSE Values

Scenario	Link 1 [deg]	Link 2 [deg]	Link 3 [deg]
Without Object	2.9	8.3	1.5
Bottle	2.1	0.2	3.3
Book 1	1.5	0.1	2.6
Book 2	2.7	0.4	3.3

situations.

5.3 Deep Reinforcement Learning Model Training Results

This section explores the outcomes of training a deep reinforcement learning model. 55 models were trained, and the training process revealed several challenges, including getting stuck in local optima, slow training speed, and a potentially misleading reward function. By addressing these issues, we observed improvements in the training results. The performance of the trained models was evaluated using a combination of metrics like success rate and mean reward per episode. To gain further insights, we analyzed various plots generated during training, including rollout reward, actor loss, and mean evaluation

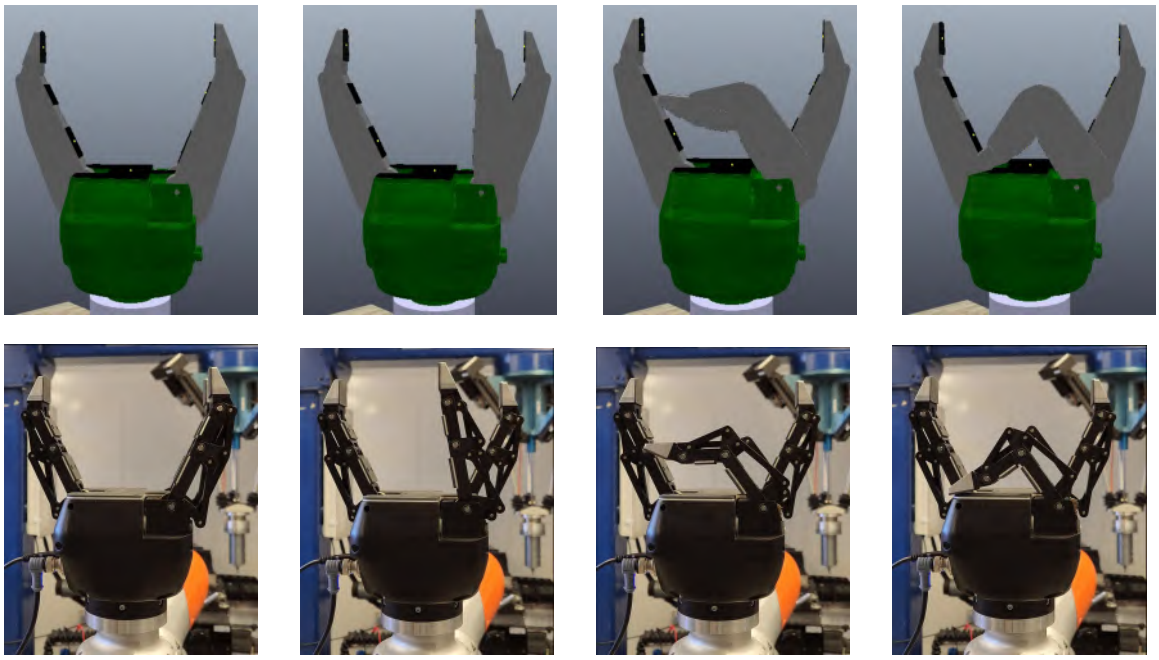


Figure 5.4: Comparison of image sequences: Closing motion of an empty gripper in simulation (top) and experiment (bottom)

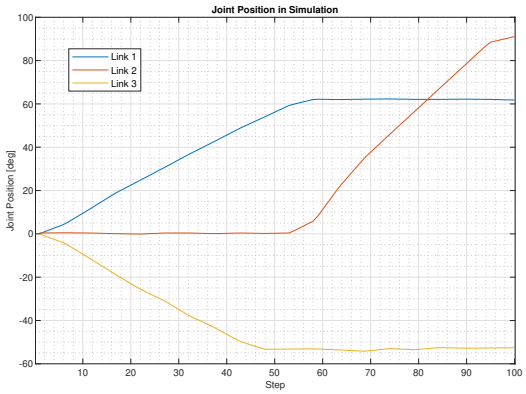
reward. A detailed analysis of these metrics and plots will be provided to understand the best model's performance and the overall effectiveness of the training process.

One of the initial challenges encountered during training was the slow training speed inherent to DRL algorithms. DRL algorithms typically require significant time for training as they generate data through trial and error within the simulated environment. A common technique to address this challenge is parallel learning. In parallel learning, the agent interacts with multiple environments simultaneously, effectively multiplying the experience collected per unit time. This technique was implemented in our training process, and the results are presented in Table 5.4. This table compares training runs with different numbers

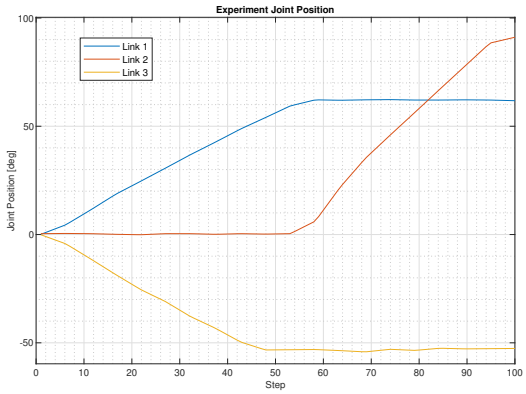
Table 5.4: Comparison of Training with Single vs. Parallel Environments

Number of env.	Episodes	Time Elapsed	Total Time Steps	Episode /second	Time step /second
1	2805	84180	48097	0.033	0.571
16	20000	151200	20000	0.132	0.132

of environments based on total elapsed time, total time steps, average number of episodes per second, and average number of time steps per second. The results show that while increasing the number of environments decreases the average time steps per second (more time spent per individual step due to managing multiple environments), it also increases the average number of episodes per second (more experience collected overall). This indicates



(a) Gripper links position for Without Object case in simulation



(b) Gripper links position for Without Object case in experiment

Figure 5.5: Gripper links position for Without Object case

that using multiple environments accelerates training by gathering more experience in a shorter total wall-clock time.

The next issue encountered was that the roll-out mean reward was oscillating and not increasing, while the actor loss was on an upward trend. This indicated that the agent was either not exploring the space effectively or was stuck in a particular region, leading to a lack of improvement as shown in the diagrams of Figure 5.12. As shown in this Figure 5.12a, the mean reward collected at each episode over training is oscillating. Additionally, Figure 5.12b illustrates the gradual increase in actor loss. From these observations, it can be inferred that the agent was not exploring the environment adequately and was not learning effectively.

Possible reasons for this issue could include the agent getting stuck in a local optimum, where the actor network finds a local optimum point and remains there. Another potential cause could be a misleading reward function, where the agent cannot focus on an optimal policy because the received rewards are not informative. To address this issue, two techniques were employed: adding action noise and revising the reward function. In terms of action noise, two different types of noise were used: Gaussian noise and Ornstein-Uhlenbeck noise.

After experimenting with different algorithms and reward functions, the following options yielded the best results. The reward function for the best-trained model is presented in Equation 5.3.1.

$$\text{Reward} = 7.5(1 - \tanh(2d)) + 0.5r_t + r_c + r_s + r_s r_q \tag{5.3.1}$$

As shown in this equation, the hyperbolic tangent function with a distance coefficient,

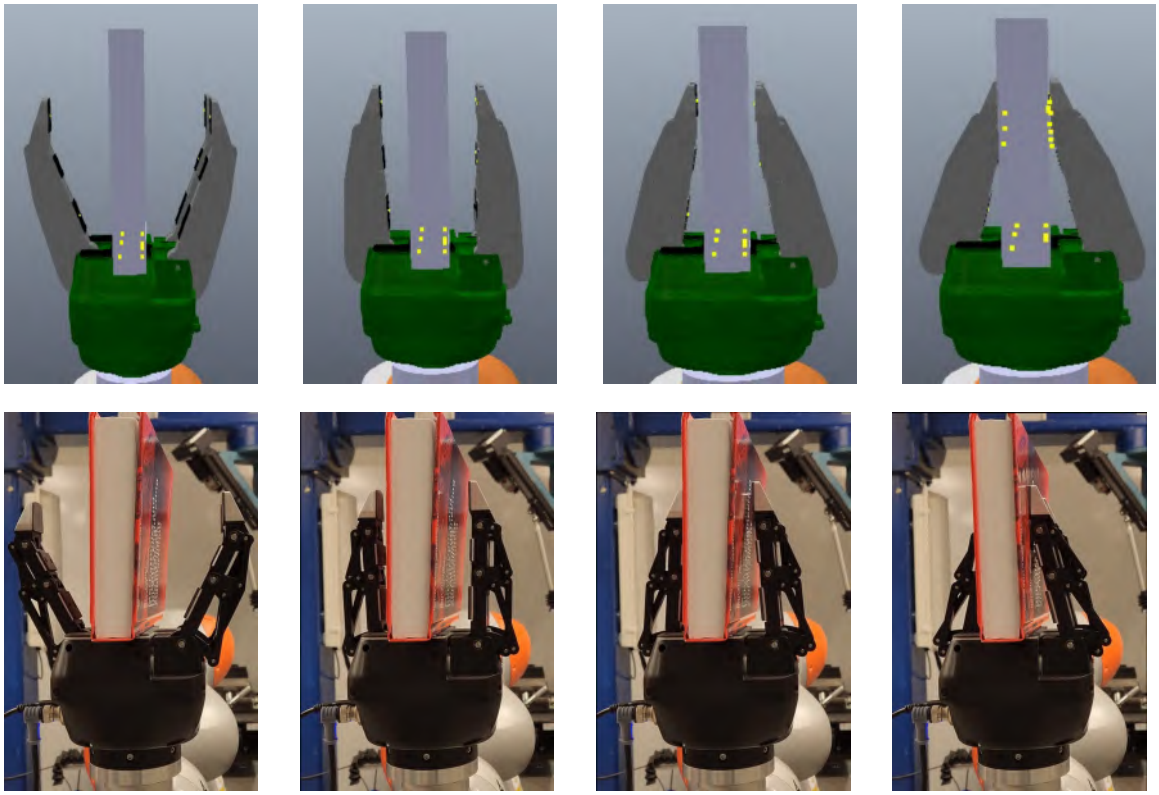
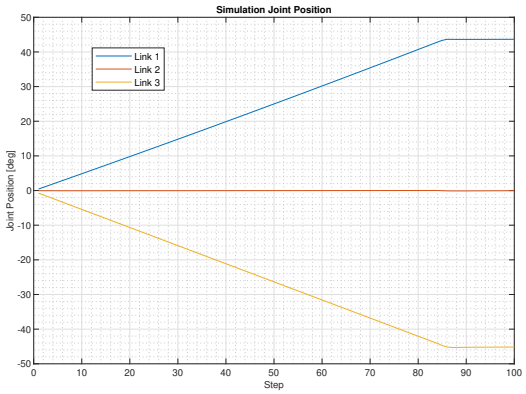


Figure 5.6: Comparison of image sequences: Closing motion of the gripper in simulation (top) and experiment (bottom) for Book 1 scenario

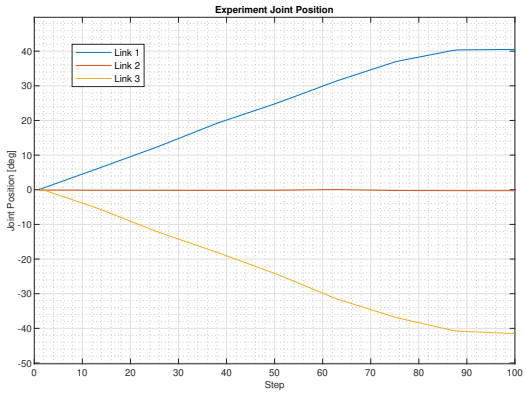
where each reward term was weighted, produced the best results. Additionally, the options and hyperparameters chosen are shown in Table 5.5. This table indicates that the best model was found using the TD3 algorithm. Sensitivity analysis was performed by varying one parameter while keeping others constant, followed by performance evaluation. The training lasted for 32 hours, with four parallel environments operating simultaneously. The translational action space, where the agent decides how much the gripper moves in each direction, performed better than the position-based action space. Regarding action noise, the normal Gaussian action noise with a range of 0.1 was more effective. Furthermore, the observation space where the perimeter of the object was represented with 2D points outperformed the scenario where a depth image without any processing was provided.

Also, the hyper-parameters and the architecture of the neural networks used in this model, which was chosen according to the original paper[67], are reported here. In Table 5.6 are reported, and in Figure 5.13.

In this section, the diagrams of the best-trained model are presented. The first diagram is the actor network loss, depicted in Figure 5.14a. As shown, the actor network loss gradually decreases, indicating that the actor network is selecting actions leading to states with higher values. Additionally, it is observable from this diagram that the values oscillate



(a) Gripper links position for the Book 1 case in simulation



(b) Gripper links position for the Book 1 case is experiment

Figure 5.7: Gripper links position for the Book 1 case

slightly. This oscillation occurs because the agent is still exploring the environment to discover new states with higher values. The next diagram to consider is the critic network loss, as shown in Figure 5.14b. The critic network loss diagram illustrates the performance of the critic network in predicting the expected reward, essentially estimating the relationship between states, actions, and rewards. For the best model, this metric does not show a clear trend of decreasing or increasing; instead, it oscillates around a fixed value. This indicates that the agent is actively exploring the environment. Although a decreasing trend would be ideal, a fixed or oscillating loss does not necessarily imply poor performance. It simply reflects the dynamics of the network's training process.

The next diagram to consider is the rollout length, shown in Figure 5.15a which depicts the duration of an episode until termination. As illustrated, the episode length stabilizes after a certain period, indicating that the agent's behavior has become consistent. This stability suggests that the agent has learned a reliable policy. Following this, the rollout reward diagram, presented in Figure 5.15b, is crucial for evaluating the agent's performance. As discussed in Chapter Three, the primary objective of a RL problem is to maximize the reward. This diagram displays the reward accumulated during the training process. The trend in this diagram provides insights into how well the agent is performing and improving over time. As shown in the plot, the reward gradually increases and then stabilizes. This indicates that the agent's performance is improving as it learns to perform the task more effectively. Similar diagrams for evaluation episodes are illustrated in Figure 5.16. These diagrams reveal the same observations: the length of the episodes becomes steady over time, the agent learns to accumulate more rewards, and it finds a policy that maximizes the reward.

It is important to distinguish between rollout episodes and evaluation episodes. Rollout episodes occur frequently whenever an episode is terminated. Conversely, evaluation

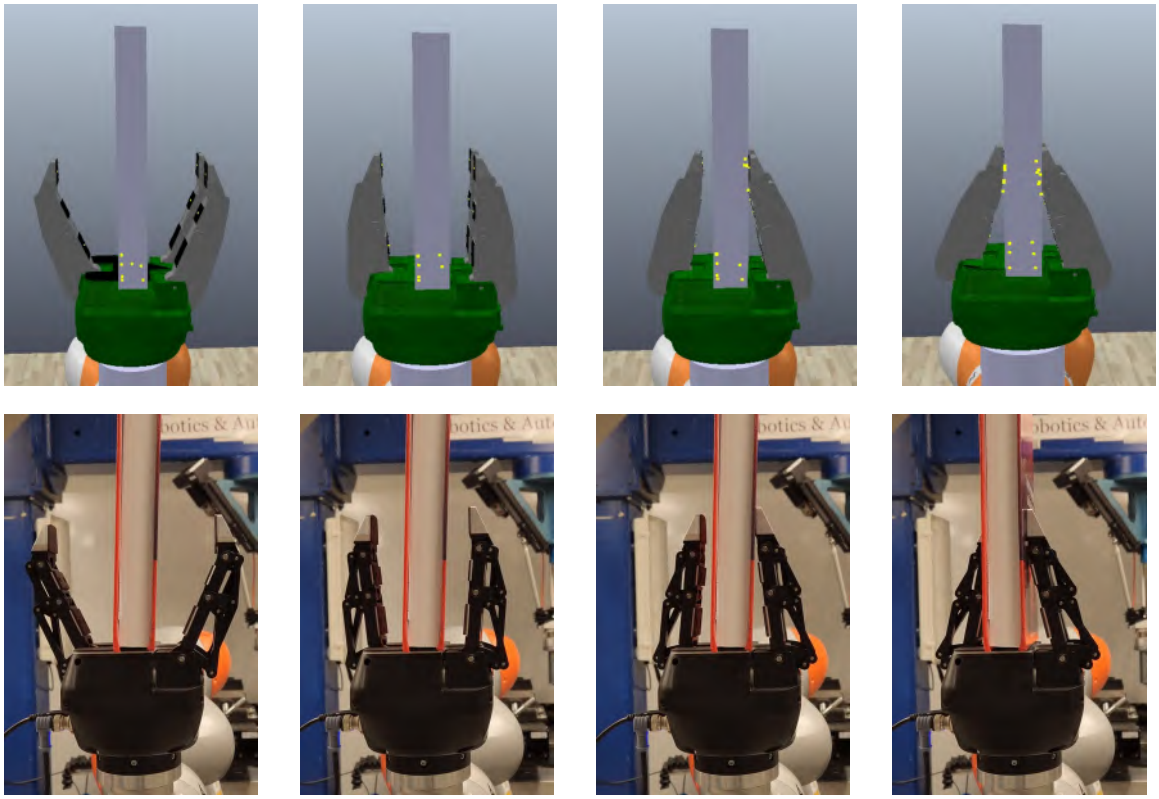
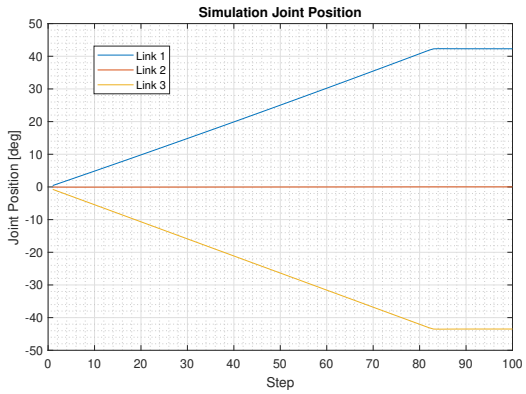


Figure 5.8: Comparison of image sequences: Closing motion of the gripper in simulation (top) and experiment (bottom) for Book 2 scenario

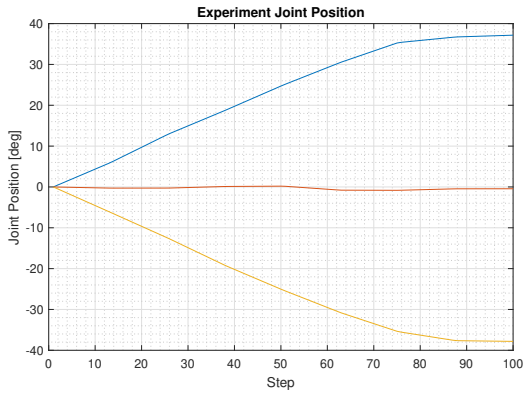
episodes occur less frequently. During these episodes, the most recently trained model is evaluated and compared with previous evaluation models, and the best-performing model is saved. This process ensures that the agent is consistently improving and adapting to find the optimal policy.

These diagrams collectively highlight the progress and effectiveness of the trained model, demonstrating how the agent learns to interact with the environment and optimize its actions to achieve higher rewards. The steady rollout length and the pattern in the rollout reward diagram indicate the agent's ability to stabilize and maximize its performance through continuous learning and adaptation.

To qualitatively evaluate the best model, images of some successful grasps are included here. As mentioned earlier, during the training process, approximately 24 objects were used, with each episode selecting a random object placed at a random location. The gripper then attempted to grasp this object. The following images are captured from the evaluation episodes. The successful grasps are shown in Figure 5.17. These images demonstrate that the model was able to grasp various objects positioned at different locations effectively. The model performed well for objects with relatively medium or high height. However, the agent struggled with objects of low height, as shown in Figure 5.18. In these images, the agent



(a) Gripper links position for the Book 2 case in simulation



(b) Gripper links position for the Book 2 case is experiment

Figure 5.9: Gripper links position for the Book 2 case

successfully approached the object and achieved a nearly perfect position in the XY plane but failed to select the optimal height for a successful grasp. This limitation highlights an area for further refinement in the training process to improve the model's performance with low-height objects.

5.4 Conclusion and Contributions

In this chapter, all the results obtained were thoroughly discussed. First, the validation results of the robot's kinematic model were presented. As indicated by the diagrams and RMSE values, the simulation was successfully validated. Additionally, the IK solution for the robot was implemented for the first time within a novel smart grasping environment.

The chapter continued by showcasing the results of validating the new kinematic model of a three-finger gripper. This model was validated both quantitatively and qualitatively, demonstrating its ability to accurately mimic the behavior of an actual gripper. Finally, the results of the DRL model training were discussed.

During the initial stages of training, several challenges were encountered, including local optima and slow training speed. These challenges were effectively addressed. Subsequently, the specifications of the best model were reported. The most successful model was identified using the TD3 algorithm, which enabled the agent to grasp various objects placed in different locations.

The simulation model was developed from scratch, and the findings demonstrated its effectiveness. Therefore, this novel framework provides other researchers with a robust environment to apply smart grasping techniques.

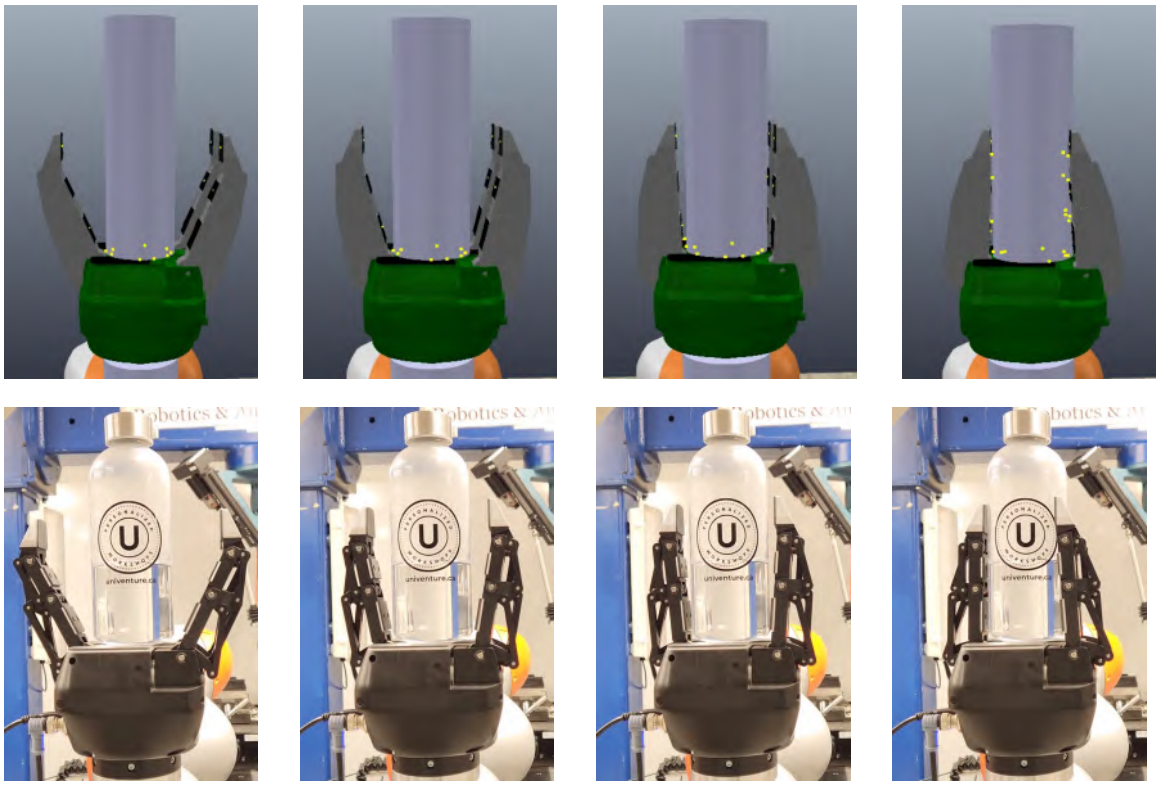
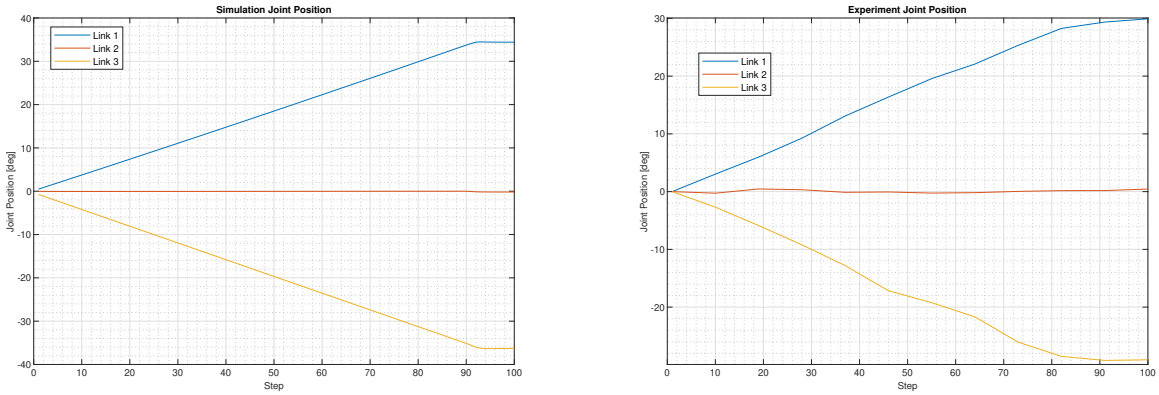


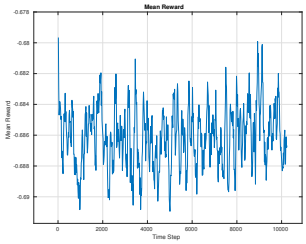
Figure 5.10: Comparison of image sequences: Closing motion of the gripper in simulation (top) and experiment (bottom) for Bottle scenario



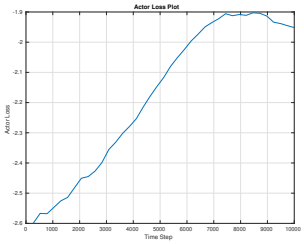
(a) Gripper links position for the Bottle case in simulation

(b) Gripper links position for the Bottle case is experiment

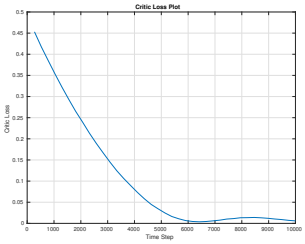
Figure 5.11: Gripper links position for the Bottle case



(a) Mean Reward



(b) Actor Loss



(c) Critic Loss

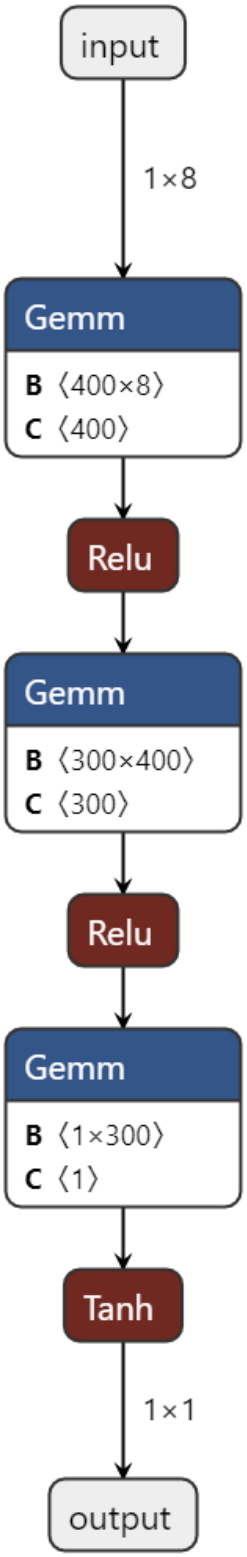
Figure 5.12: Performance of Models During Early Training Stages

Table 5.5: Best Trained Model Hyper parameters

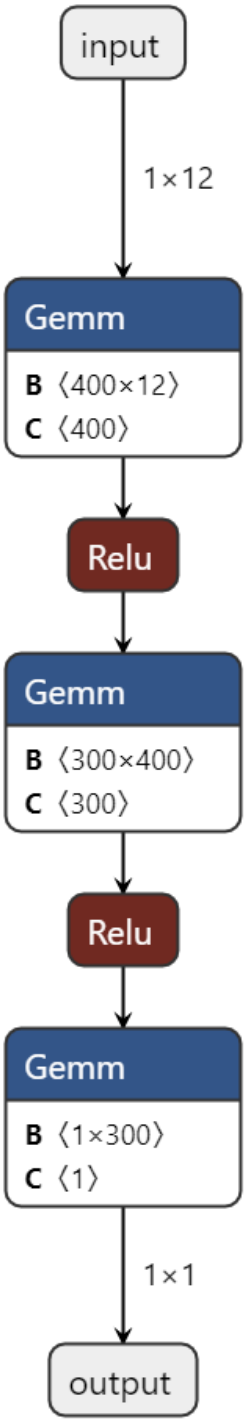
Algorithm	TD3
Date	May 8, 2024
Elapsed Time	32 Hours
Total time steps	244,000
Total Episodes	9560
Device	Hp Desktop
Number of Environments	4
Sync Mode	Sync
Object Random Position	Enabled
Environment Device	Gripper
Batch Size	512
Noise and Domain	Normal 0.1
Action Space	Transnational $XYZR$
Observation Space	Primary vector and 2D points
Mean Reward	59.44 ± 2.98
success Rate %	58 ± 4.9

Table 5.6: TD3 hyper-parameters

Critic Learning Rate	10^{-3}
Critic Regularization	None
Actor Learning Rate	10^{-3}
Actor Regularization	None
Optimizer	Adam
Target Update Rate	5.10^{-3}
Discount Factor	0.99

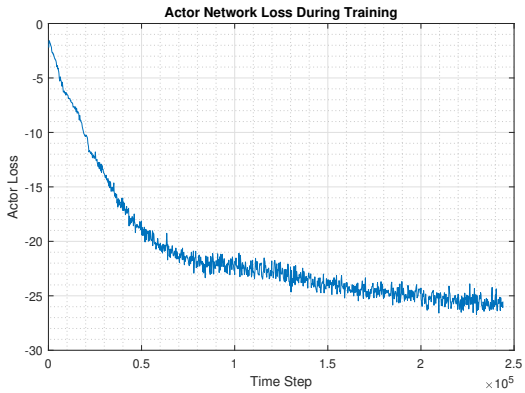


(a) Actor Network Architecture

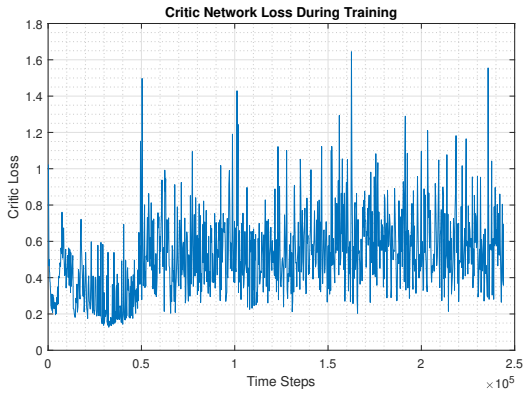


(b) Critic Network Architecture

Figure 5.13: Neural Networks Architecture

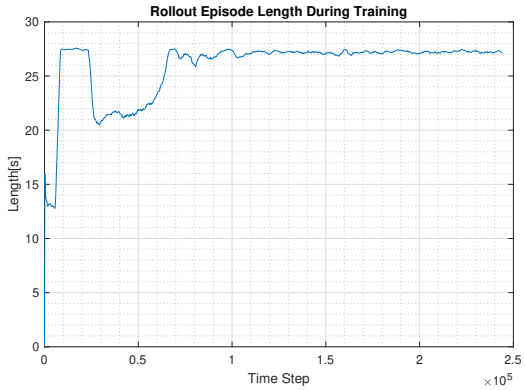


(a) Best Model: Actor Network Loss Diagram

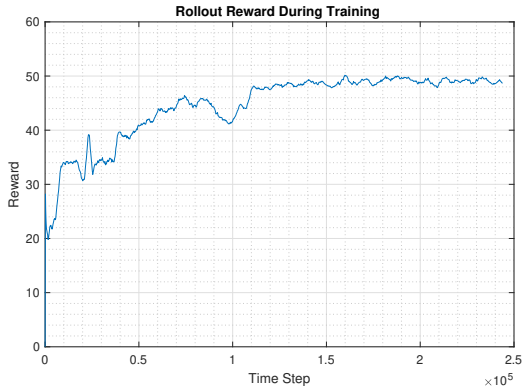


(b) Best Model: Critic Network Loss Diagram

Figure 5.14: Best Model Networks Loss

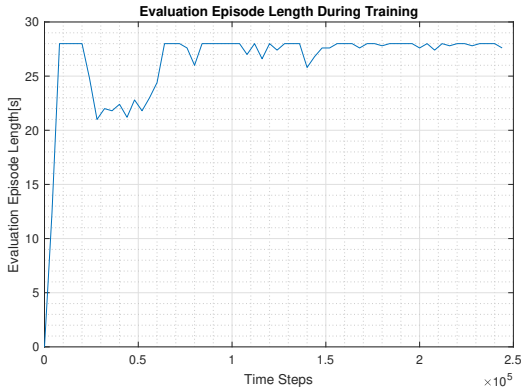


(a) Best Model: Rollout Episode Length

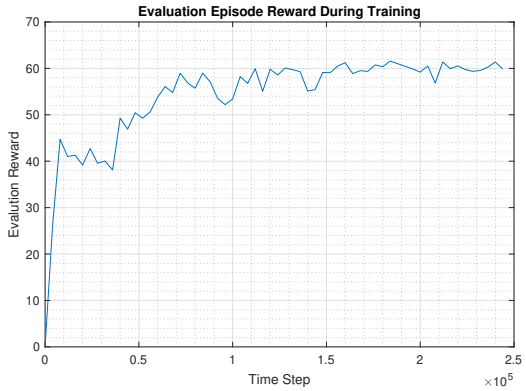


(b) Best Model: Rollout Episode Reward

Figure 5.15: Best Model Rollout Diagrams



(a) Best Model: Evaluation Episode Length



(b) Best Model: Evaluation Episode Reward

Figure 5.16: Best Model Evaluation Diagrams

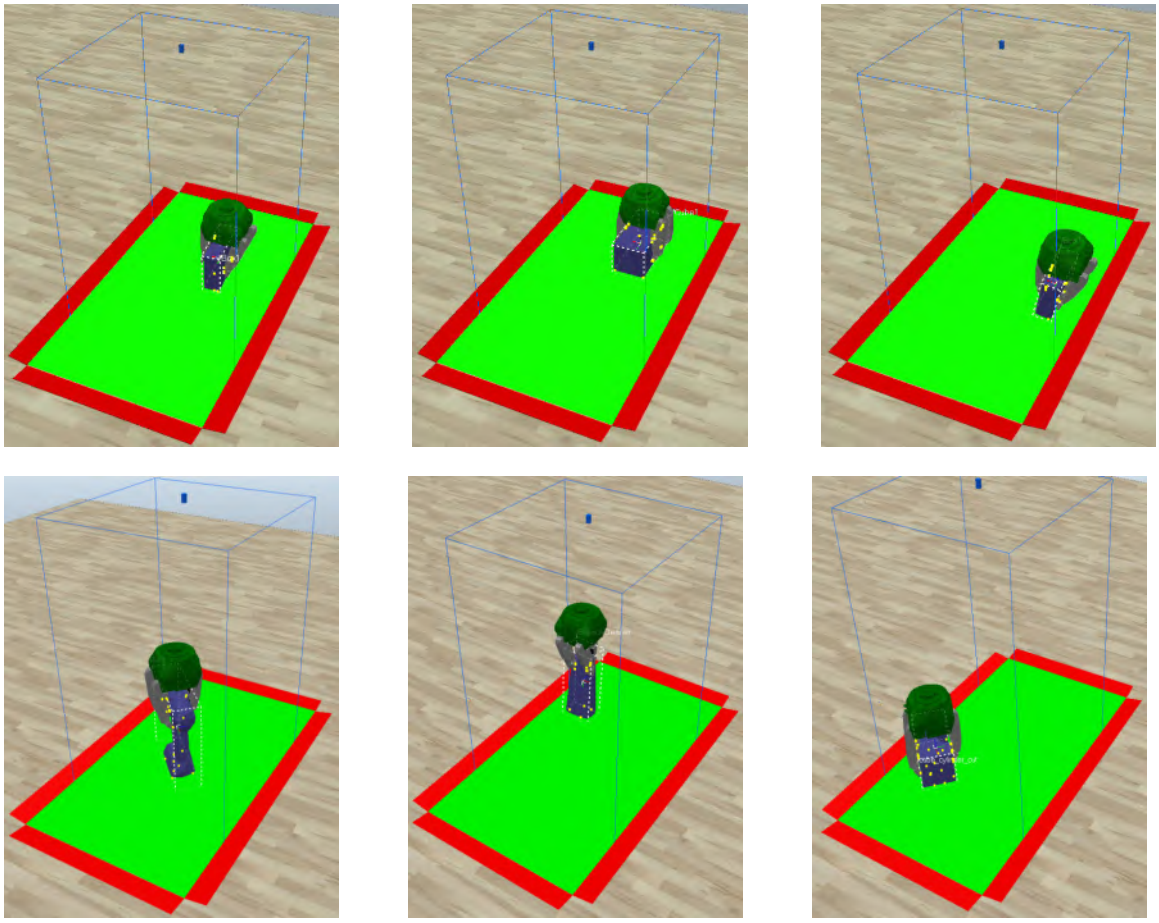


Figure 5.17: Sample Successful Grasps

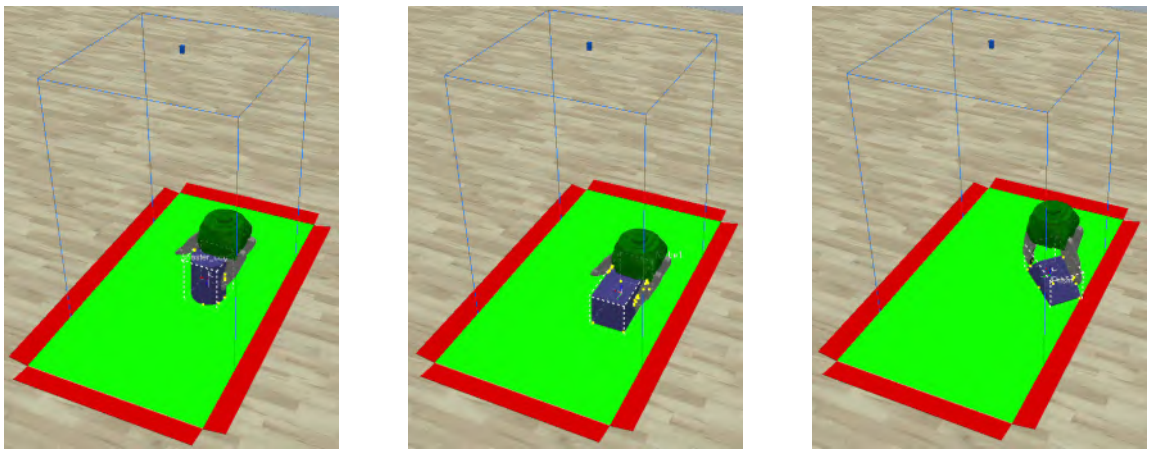


Figure 5.18: Sample Unsuccessful Grasps

Chapter 6

Conclusions and Future Work

In this chapter, the overall conclusions and results of this thesis are presented. Following this, a comparison is made between this thesis and other works in the same context. Subsequently, the limitations of this research are discussed. Finally, potential future works to enhance this study are explored.

The conclusions summarize the key findings, highlighting the contributions and the effectiveness of the proposed methodologies. The comparative analysis positions this research within the broader field, examining similarities and differences with other studies.

The limitations section acknowledges the constraints and challenges encountered during the research, providing a critical assessment of areas where the work fell short or could be further refined.

The discussion on future work outlines potential directions for extending and improving the research. By identifying these future research avenues, the thesis sets a foundation for continued exploration and development in the field.

6.1 Conclusions and Contributions

In this work, a novel smart grasping environment was created from scratch, allowing a DRL agent to interface with this environment to learn a policy for optimally grasping unknown objects. The environment includes a manipulator, a gripper, and a vision sensor.

The manipulator was a serial redundant robot that had several solutions for a given pose. To address this issue, a motion planning algorithm was proposed, which includes an IK solution and a redundancy resolution part. The redundancy resolution was performed by considering joint limits and singular points. The proposed motion planning algorithm

was able to precisely follow the given trajectory, and the kinematic simulation model was successfully validated.

The next main part of the environment was the gripper, which was modeled for the first time in simulation. Using a state machine, the behavior of the gripper was modeled, and then this model was successfully validated by comparing the simulation model behavior to the real device.

After integrating all these parts into the environment, the model training process began. Three DRL algorithms were used with different observation and reward spaces. The best results were found using the TD3 algorithm. To increase the training speed, a parallel environment was used. However, it was found that increasing the number of environments further would sacrifice sample efficiency. Regarding the reward function, weighting reward terms helped the agent focus on important parts of the task. Additionally, using a Sigma function with a proper coefficient for the distance helped the agent approach the target faster and more precisely. For observations, including an unprocessed depth image was not effective; instead, a set of 2D points representing the object's perimeter worked better. The translational action space performed better than the position-based action space. There were no sudden changes in the gripper's position, and the changes in the observation vector values were smooth. This approach was inspired by human logic when grasping objects and applied similarly to the agent, enabling it to learn a better policy.

After applying all these techniques, the agent was successfully able to grasp different objects placed at random locations. The agent effectively grasped objects with relatively high heights but struggled with low-height objects, resulting in a lower overall success rate.

Finally, the contributions of this thesis are outlined as follows:

- **3F-Gripper Modeling:** A novel kinematic simulation model for an adaptive 3F-Gripper was developed using a state machine. This model allows researchers to enhance their simulated grasping systems by using a 3F-gripper. Additionally, it enables the employment of broader grasp quality functions for grasp planning, providing a more comprehensive approach to evaluating grasp stability and efficiency.

- **3F-Gripper for Smart Grasping:** This thesis utilized a 3F-gripper and introduced a grasp quality score reward term for the first time to achieve optimal grasps. This innovation can assist other researchers in developing more efficient grasping strategies by leveraging these features, thus contributing to advancements in the field of robotic grasping.

- **DRL Simulation Environment:** A novel simulation environment for a DRL agent was created from scratch. This environment is scalable and adaptable to various

smart grasping tasks, providing a flexible platform for testing and refining DRL-based grasping algorithms.

6.2 Discussion: Comparison and Limitation

In this section, this work is compared with similar studies in this field, and the differences are discussed. Subsequently, the limitations of this work are detailed.

Works on DRL-based smart grasping systems can be categorized based on the desired tasks, such as grasping moving objects, grasping unknown objects, and grasping in cluttered environments. As shown in Table 2.1 in Chapter 2, the main focus of these works, including this thesis, is on grasping unknown objects. Common metrics for comparing these works include success rate and experimental results. Although this thesis shows a lower success rate and lacks extensive experimental results compared to other works, it includes a significant focus on grasp evaluation. Additionally, this thesis introduces the use of a 3F-gripper for the first time to achieve better grasps. Another novel aspect, compared to other works, is the creation of a kinematic simulation model of an adaptive 3F-gripper.

Regarding limitations, this work has several constraints similar to any research endeavor:

- **Limited Action Space:** The action space in this work was restricted to rotation along a single axis. This decision was inspired by human grasping behavior, where objects are approached from above before lowering the hand to grasp. Although this limitation does not impede the possibility of a successful grasp, it restricts the application of various grasp quality scores. For achieving optimal and stable grasps, a broader set of grasp quality scores should be utilized.

- **Contact Detection Method for Gripper:** The contact detection method for the simulation model of the gripper was based on a distance threshold. Proximity sensors were placed at each fingertip, and contact was detected when the distance was below a specified threshold. Consequently, there was always a gap of less than $1mm$ between the object and the fingertip. This limitation meant that the grasp detection method relied on contact rather than the actual lifting of the object.

- **Kinematic model:** Another limitation of the gripper model was the lack of control over the applied force. In the actual gripper, the force can be adjusted to handle soft and brittle objects. In the model developed in this thesis, the applied force was either fixed or absent, limiting the gripper's ability to handle delicate objects.

6.3 Future Work

In this section, potential future works to enhance this research are discussed.

- **Real-Time Vision Data:** In this work, vision data was retrieved at the beginning of each episode, after which the vision part of the observation remained fixed. Additionally, the object's position was determined using the physics engine function, which is not feasible in the real world. Another limitation is that if there is any change in the object's position or orientation during an episode, the vision data does not update. To address this, vision data can be updated at each time step, providing the agent with real-time information. This would enable the agent to handle unexpected events more effectively. Moreover, the agent could use vision data to estimate the object's position, facilitating better implementation in real-world scenarios. Positioning the camera on the wrist could further enhance system mobility.

- **Gripper Model:** As discussed in the limitations section, the current gripper model lacks an accurate contact detection method and force control capabilities. These shortcomings can be addressed by using a more reliable engine for contact simulation, such as **MuJuCo**[70]. Additionally, the gripper's adaptive feature, which allows for changing the relative angle between fingers to accommodate different shapes, could be better utilized.

- **Camera Noise:** In future research, using Gaussian noise in the camera sensor's observation space for reinforcement learning (RL) can help make RL agents more robust and better suited for real-world applications. By introducing random noise to the input images, this approach simulates real-world imperfections like blur, distortion, or lighting variations, allowing agents to learn in more realistic settings. Gaussian noise improves generalization by preventing overfitting to idealized data and helps agents handle uncertainty in perception, which is crucial for autonomous systems like robots or self-driving cars operating in imperfect environments.

- **Grasp Condition:** In this work, the grasp condition is based on the relative distance between the gripper and the object. This condition could be improved by allowing the agent to decide when to grasp, thereby making the process more autonomous and efficient.

- **Imitation Learning:** Another potential improvement is the implementation of imitation learning. In DRL, the agent begins by interacting with the environment to gain experience without any preloaded data in the buffer. Imitation learning involves showing the agent how to perform the task, then allowing it to practice the skill, even in new states. This approach ensures the agent has an understanding of an optimal policy before training, enabling it to refine this skill during training and apply it to

unseen states.

Bibliography

[1] Devika Kannan, Parvaneh Gholipour, and Chunguang Bai. "Smart manufacturing as a strategic tool to mitigate sustainable manufacturing challenges: a case approach". In: *Annals of Operations Research* 331 (2023), pp. 543–579. DOI: 10.1007/S10479-023-05472-6/FIGURES/5.

[2] Hyoung Seok Kang et al. "Smart manufacturing: Past research, present findings, and future directions". In: *International Journal of Precision Engineering and Manufacturing - Green Technology* 3 (2016), pp. 111–128. DOI: 10.1007/S40684-016-0015-5/METRICS.

[3] Lorenzo Gagliardini et al. "A reconfiguration strategy for Reconfigurable Cable-Driven Parallel Robots". In: *Proceedings - IEEE International Conference on Robotics and Automation* (2015), pp. 1613–1620. DOI: 10.1109/ICRA.2015.7139404.

[4] Klaus Dieter Thoben, Stefan Alexander Wiesner, and Thorsten Wuest. "Industrie 4.0" and smart manufacturing-a review of research issues and application examples". In: *International Journal of Automation Technology* 11 (2017), pp. 4–16. DOI: 10.20965/IJAT.2017.P0004.

[5] Rabab Benotsmane, László Dudás, and György Kovács. "Trajectory optimization of industrial robot arms using a newly elaborated "whip-lashing" method". In: *Applied Sciences (Switzerland)* 10 (2020), pp. 1–18. DOI: 10.3390/APP10238666.

[6] João Pedro Carvalho de Souza et al. "Robotic grasping: from wrench space heuristics to deep learning policies". In: *Robotics and Computer-Integrated Manufacturing* 71 (Oct. 2021). DOI: 10.1016/j.rcim.2021.102176.

[7] Ashutosh Saxena, Justin Driemeyer, and Andrew Y. Ng. "Robotic Grasping of Novel Objects using Vision". In: *The International Journal of Robotics Research* 27 (Feb. 2008), pp. 157–173. DOI: 10.1177/0278364907087172.

[8] Bruno Siciliano and Oussama Khatib. "Springer Handbook of Robotics, Grasping". In: *Springer Handbook of Robotics* (2016), pp. 955–988. DOI: 10.1007/978-3-540-30301-5.

[9] Julian Ibarz et al. “How to train your robot with deep reinforcement learning: lessons we have learned”. In: *International Journal of Robotics Research* 40 (2021), pp. 698–721. DOI: 10.1177/0278364920987859.

[10] Tuomas Haarnoja et al. “Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor”. In: *35th International Conference on Machine Learning, ICML 2018* 5 (2018), pp. 2976–2989.

[11] Yun Hui Liu, Miu Ling Lam, and Dan Ding. “A complete and efficient algorithm for searching 3-D form-closure grasps in the discrete domain”. In: *IEEE Transactions on Robotics* 20 (2004), pp. 805–816. DOI: 10.1109/TRO.2004.829500.

[12] Sahar El-Khoury and Anis Sahbani. “On computing robust n-finger force-closure grasps of 3D objects”. In: *Proceedings - IEEE International Conference on Robotics and Automation* (2009), pp. 2480–2486. DOI: 10.1109/ROBOT.2009.5152272.

[13] Andrew T. Miller et al. “Automatic grasp planning using shape primitives”. In: *Proceedings - IEEE International Conference on Robotics and Automation* 2 (2003), pp. 1824–1829. DOI: 10.1109/robot.2003.1241860.

[14] Antonio Morales et al. “Integrated grasp planning and visual object localization for a humanoid robot with five-fingered hands”. In: *International Conference on Intelligent Robots and Systems* (2006), pp. 5663–5668. DOI: 10.1109/IROS.2006.282367.

[15] João Pedro Carvalho de Souza et al. “Reconfigurable Grasp Planning Pipeline with Grasp Synthesis and Selection Applied to Picking Operations in Aerospace Factories”. In: *Robotics and Computer-Integrated Manufacturing* 67 (2021), p. 102032. DOI: 10.1016/J.RCIM.2020.102032.

[16] L. Ingber. “Very fast simulated re-annealing”. In: *Mathematical and Computer Modelling* 12.8 (1989), pp. 967–973. DOI: https://doi.org/10.1016/0895-7177(89)90202-1.

[17] Siddarth Jain and Brenna Argall. “Grasp detection for assistive robotic manipulation”. In: *IEEE Int Conf Robot Autom* (2016), pp. 2015–2021. DOI: 10.1109/ICRA.2016.7487348.

[18] Corey Goldfeder et al. “Grasp planning via decomposition trees”. In: *Proceedings 2007 IEEE International Conference on Robotics and Automation* (2007), pp. 4679–4684. DOI: 10.1109/ROBOT.2007.364200.

[19] Jacopo Aleotti and Stefano Caselli. “A 3D shape segmentation approach for robot grasping by parts”. In: *Robotics and Autonomous Systems* 60 (2012), pp. 358–366. DOI: 10.1016/j.robot.2011.07.022.

[20] V. Rakesh et al. “Optimizing force closure grasps on 3D objects using a modified genetic algorithm”. In: *Soft Compute* 22 (2018), pp. 759–772. DOI: 10.1007/s00500-016-2377-6.

[21] Matei T. Ciocarlie and Peter K. Allen. "Hand posture subspaces for dexterous robotic grasping". In: *The International Journal of Robotics Research* 28 (2009), pp. 851–867. DOI: 10.1177/0278364909105606.

[22] Sergey Levine et al. "End-to-End Training of Deep Visuomotor Policies". In: *Journal of Machine Learning Research* 17 (2015), pp. 1–40.

[23] Yevgen Chebotar et al. "Path integral guided policy search". In: *Proceedings - IEEE International Conference on Robotics and Automation* (2017), pp. 3381–3388. DOI: 10.1109/ICRA.2017.7989384.

[24] Sehoon Ha, Joohyung Kim, and Katsu Yamane. "Automated Deep Reinforcement Learning Environment for Hardware of a Modular Legged Robot". In: *2018 15th International Conference on Ubiquitous Robots, UR 2018* (2018), pp. 348–354. DOI: 10.1109/URAI.2018.8442201.

[25] Jemin Hwangbo et al. "Learning agile and dynamic motor skills for legged robots". In: *Science Robotics* 4 (2019). DOI: 10.1126/SCIROBOTICS.AAU5872/SUPPL_FILE/AAU5872_SM.PDF.

[26] Joonho Lee et al. "Learning quadrupedal locomotion over challenging terrain". In: *Science Robotics* 5 (2020), p. 5986. DOI: 10.1126/SCIROBOTICS.ABC5986/SUPPL_FILE/ABC5986_SM.PDF.

[27] Nate Kohl and Peter Stone. "Policy gradient reinforcement learning for fast quadrupedal locomotion". In: *Proceedings - IEEE International Conference on Robotics and Automation* (2004), pp. 2619–2624. DOI: 10.1109/ROBOT.2004.1307456.

[28] Laura Smith, Ilya Kostrikov, and Sergey Levine. "A Walk in the Park: Learning to Walk in 20 Minutes With Model-Free Reinforcement Learning". In: *Computer Science* (2022).

[29] Yun Jiang, Stephen Moseson, and Ashutosh Saxena. "Efficient grasping from RGBD images: Learning using a new rectangle representation". In: *Proceedings - IEEE International Conference on Robotics and Automation* (2011), pp. 3304–3311. DOI: 10.1109/ICRA.2011.5980145.

[30] Jeffrey Mahler et al. "Dex-Net 2.0: Deep Learning to Plan Robust Grasps with Synthetic Point Clouds and Analytic Grasp Metrics". In: *Robotics: Science and Systems* 13 (2017). DOI: 10.15607/rss.2017.xiii.058.

[31] Andreas ten Pas and Robert Platt. "Using geometry to detect grasp poses in 3D point clouds". In: vol. 2. Springer Science and Business Media B.V., 2018, pp. 307–324. DOI: 10.1007/978-3-319-51532-8_19.

[32] Raphael Pelossof et al. "An SVM learning approach to robotic grasping". In: *IEEE International Conference on Robotics and Automation (ICRA)* 2004 (2004), pp. 3512–3518. DOI: 10.1109/robot.2004.1308797.

[33] Ian Lenz, Honglak Lee, and Ashutosh Saxena. "Deep learning for detecting robotic grasps". In: *The International Journal of Robotics Research* 34 (2015), pp. 705–724. DOI: 10.1177/0278364914549607.

[34] Joseph Redmon and Anelia Angelova. "Real-time grasp detection using convolutional neural networks". In: *Proceedings - IEEE International Conference on Robotics and Automation* (2015), pp. 1316–1322. DOI: 10.1109/ICRA.2015.7139361.

[35] Di Guo et al. "A hybrid deep architecture for robotic grasp detection". In: *Proceedings - IEEE International Conference on Robotics and Automation* (2017), pp. 1609–1614. DOI: 10.1109/ICRA.2017.7989191.

[36] Fu Jen Chu, Ruinian Xu, and Patricio A. Vela. "Real-world multiobject, multigrasp detection". In: *IEEE Robotics and Automation Letters* 3 (2018), pp. 3355–3362. DOI: 10.1109/LRA.2018.2852777.

[37] Ghazal Ghazaei et al. "Dealing with Ambiguity in Robotic Grasping via Multiple Predictions". In: *Computer Vision – ACCV 2018* 11364 LNCS (2018), pp. 38–55. DOI: 10.1007/978-3-030-20870-7_3.

[38] Andreas ten Pas et al. "Grasp Pose Detection in Point Clouds". In: *International Journal of Robotics Research* 36 (2017), pp. 1455–1473. DOI: 10.1177/0278364917735594/ASSET/IMAGES/LARGE/10.1177_0278364917735594-FIG16.JPEG.

[39] Shuai Zhang et al. "A visual imitation learning algorithm for the selection of robots' grasping points". In: *Robotics and Autonomous Systems* 172 (2024). DOI: 10.1016/j.robot.2023.104600.

[40] Jae Bong Yi et al. "Anthropomorphic Grasping of Complex-Shaped Objects Using Imitation Learning". In: *Applied Sciences* 12 (2022). DOI: 10.3390/app122412861.

[41] Dmitry Kalashnikov et al. "QT-Opt: Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation". In: *2nd Conference on Robot Learning (CoRL 2018), Zurich, Switzerland* (2018). DOI: 10.48550/arxiv.1806.10293.

[42] Shirin Joshi, Sulabh Kumra, and Ferat Sahin. "Robotic Grasping using Deep Reinforcement Learning". In: *IEEE International Conference on Automation Science and Engineering* (2020), pp. 1461–1466. DOI: 10.1109/CASE48305.2020.9216986.

[43] Yiwen Chen, Zhaojie Ju, and Chenguang Yang. "Combining Reinforcement Learning and Rule-based Method to Manipulate Objects in Clutter". In: *Proceedings of the International Joint Conference on Neural Networks* (2020). DOI: 10.1109/IJCNN48605.2020.9207153.

[44] Natanael Magno Gomes et al. "Deep Reinforcement Learning Applied to a Robotic Pick-and-Place Application". In: *Communications in Computer and Information Science* 1488 CCIS (2021), pp. 251–265. DOI: 10.1007/978-3-030-91885-9_18.

[45] Pengzhan Chen and Weiqing Lu. "Deep reinforcement learning based moving object grasping". In: *Information Sciences* 565 (July 2021), pp. 62–76. DOI: 10.1016/J.INS.2021.01.077.

[46] Abdulrahman Al-Shanoon et al. "Learn to grasp unknown objects in robotic manipulation". In: *Intelligent Service Robotics* 14 (2021), pp. 571–582. DOI: 10.1007/S11370-021-00380-9.

[47] Hiba Sekkat et al. "Vision-based robotic arm control algorithm using deep reinforcement learning for autonomous objects grasping". In: *Applied Sciences (Switzerland)* 11 (2021). DOI: 10.3390/APP11177917.

[48] Marwan Qaid Mohammed et al. "Deep reinforcement learning-based robotic grasping in clutter and occlusion". In: *Sustainability (Switzerland)* 13 (2021). DOI: 10.3390/SU132413686.

[49] Haoxiang Lang Abdulrahman Al-Shanoon. "Learn to Grasp Unknown-Adjacent Objects for Sequential Robotic Manipulation". In: *Journal of Intelligent and Robotic Systems: Theory and Applications* 105 (Aug. 2022). DOI: 10.1007/S10846-022-01702-4.

[50] Feng Xiong Xiangxin Ji et al. "Grasping Control of a Vision Robot Based on a Deep Attentive Deterministic Policy Gradient". In: *IEEE Access* 10 (2022), pp. 867–878. DOI: 10.1109/ACCESS.2021.3137821.

[51] Asad Ali Shahid et al. "Continuous control actions learning and adaptation for robotic manipulation through reinforcement learning". In: *Autonomous Robots* 46 (2022), pp. 483–498. DOI: 10.1007/S10514-022-10034-Z.

[52] Tianya You et al. "A Proposed Priority Pushing and Grasping Strategy Based on an Improved Actor-Critic Algorithm". In: *Electronics (Switzerland)* 11 (2022). DOI: 10.3390/ELECTRONICS11132065.

[53] Rui Li, Shimin Liu, and Xiaojie Su. "Learning Form Closure Grasping with a Four-Pin Parallel Gripper". In: *Applied Sciences (Switzerland)* 13 (Feb. 2023). DOI: 10.3390/APP13042506.

[54] Ya Ling Chen, Yan Rou Cai, and Ming Yang Cheng. "Vision-Based Robotic Object Grasping—A Deep Reinforcement Learning Approach". In: *Machines* 11 (Feb. 2023). DOI: 10.3390/MACHINES11020275.

[55] Muhammad Babar Imtiaz, Yuansong Qiao, and Brian Lee. "Prehensile and Non-Prehensile Robotic Pick-and-Place of Objects in Clutter Using Deep Reinforcement Learning". In: *Sensors* 23 (2023), p. 1513. DOI: 10.3390/S23031513.

[56] Yu Huang et al. "A novel robotic grasping method for moving objects based on multi-agent deep reinforcement learning". In: *Robotics and Computer-Integrated Manufacturing* 86 (2024), p. 102644. DOI: 10.1016/J.RCIM.2023.102644.

[57] Xuan Zheng, Shuaiming Yuan, and Pengzhan Chen. "Robotic Autonomous Grasping Strategy and System for Cluttered Multi-class Objects". In: *International Journal of Control, Automation and Systems* 22.8 (2024), pp. 2602–2612. DOI: 10.1007/s12555-023-0358-y.

[58] Birhanemeskel Alamir Shiferaw et al. "Synergistic Pushing and Grasping for Enhanced Robotic Manipulation Using Deep Reinforcement Learning". In: *Actuators* 13.8 (2024), p. 316. DOI: 10.3390/act13080316.

[59] Yanhu Chen et al. "A synchronous multi-agent reinforcement learning framework for UVMS grasping". In: *Ocean Engineering* 307 (2024), p. 118155. DOI: 10.1016/j.oceaneng.2024.118155.

[60] Michael L. Littman Leslie Pack Kaelbling and Timothy W. Moore. "Reinforcement Learning: A Tutorial". In: *WL/AACF* 8 (1996), pp. 237–285.

[61] C.J.C.H. Watkins. "Learning From Delayed Rewards". PhD thesis. University of Cambridge, 1989.

[62] Christopher J.C.H. Watkins and Peter Dayan. "Technical Note: Q-Learning". In: *Machine Learning* 8 (1992), pp. 279–292. DOI: 10.1023/A:1022676722315/METRICS.

[63] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature 2015 518:7540* 518 (Feb. 2015), pp. 529–533. DOI: 10.1038/nature14236.

[64] Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson. "Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problems". In: *IEEE Transactions on Systems, Man and Cybernetics* SMC-13 (1983), pp. 834–846. DOI: 10.1109/TSMC.1983.6313077.

[65] David Silver et al. "Deterministic policy gradient algorithms". In: *31st International Conference on Machine Learning, ICML 2014* 1 (2014), pp. 605–619.

[66] Timothy P. Lillicrap et al. "Continuous control with deep reinforcement learning". In: *4th International Conference on Learning Representations, ICLR 2016* (2015).

[67] Scott Fujimoto, Herke Van Hoof, and David Meger. "Addressing Function Approximation Error in Actor-Critic Methods". In: *35th International Conference on Machine Learning, ICML 2018* 4 (2018), pp. 2587–2601.

[68] Alain Suero Julio Jerez. *Newton Dynamics*. URL: https://newtondynamics.com/forum/newton.php.

[69] Russ Smith. *Open Dynamics Engine*. URL: https://www.ode.org/.

[70] Emanuel Todorov, Tom Erez, and Yuval Tassa. "MuJoCo: A physics engine for model-based control". In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2012, pp. 5026–5033. DOI: 10.1109/IROS.2012.6386109.

[71] Viktor Makoviychuk et al. *Isaac Gym: High Performance GPU-Based Physics Simulation For Robot Learning*. 2021. arXiv: 2108.10470 [cs.RO].

[72] Yunfei Bai Erwin Coumans. *Pybullet, a python module for physics simulation for games, robotics and machine learning*. URL: https://pybullet.org/wordpress/.

[73] E. Rohmer, S. P. N. Singh, and M. Freese. "CoppeliaSim (formerly V-REP): a Versatile and Scalable Robot Simulation Framework". In: *Proc. of The International Conference on Intelligent Robots and Systems (IROS)*. www.coppeliarobotics.com. 2013.

[74] Masayuli Shimizu et al. "Analytical inverse kinematic computation for 7-DOF redundant manipulators with joint limits and its application to redundancy resolution". In: *IEEE Transactions on Robotics* 24 (2008), pp. 1131–1142. ISSN: 15523098. DOI: 10.1109/TRO.2008.2003266.

[75] J. Hollerbach. "Optimum Kinematic Design For A Seven Degree Of Freedom Manipulator". In: (1985).

[76] Carlos Faria et al. "Position-based kinematics for 7-DoF serial manipulators with global configuration control, joint limit and singularity avoidance". In: *Mechanism and Machine Theory* 121 (2018), pp. 317–334. DOI: 10.1016/J.MECHMACHTHEORY.2017.10.025.

[77] Mathieu Bélanger-Barrette. *What does Flexibility in Automated Manufacturing Means?* URL: https://blog.robotiq.com/bid/70628/What-does-Flexibility-in-Automated-Manufacturing-Means.

[78] Stephen Brawner. *SolidWorks to URDF Exporter*. URL: https://github.com/ros/solidworks_urdf_exporter?tab=readme-ov-file.

[79] Giulia Franchi and Kris Hauser. *Use of Hybrid Systems to model the RobotiQ Adaptive Gripper*. Tech. rep. School of Informatics and Computing, Indiana University, Bloomington, 2014.

[80] Md Tanzil Shahria et al. "A Comprehensive Review of Vision-Based Robotic Applications: Current State, Components, Approaches, Barriers, and Potential Solutions". In: *Robotics 2022, Vol. 11, Page 139* 11 (2022), p. 139. DOI: 10.3390/ROBOTICS11060139.

[81] Nobuyuki Otsu. "Threshold Selection Method From Gray-level Histograms". In: *IEEE Trans Syst Man Cybern* SMC-9 (1979), pp. 62–66. DOI: 10.1109/TSMC.1979.4310076.

[82] Christer Ericson. *Real-Time Collision Detection: A volume in The Morgan Kaufmann Series in Interactive 3D Technology*. Morgan Kaufmann, 2005.

[83] David M Bourg Bryan Bywalec. *Physics for Game Developers, 2nd Edition*. O'Reilly Media, Inc., 2013.

[84] Khaled Mamou and Faouzi Ghorbel. "A simple and efficient approach for 3D mesh approximate convex decomposition". In: *Proceedings - International Conference on Image Processing, ICIP* (2009), pp. 3501–3504. DOI: 10.1109/ICIP.2009.5414068.

[85] Eric Lengyel. *Game Engine Gems 3*. CRC Press, 2016. DOI: 10.1201/B21177/GAME-ENGINE-GEMS-3-ERIC-LENGYEL.

[86] Mark Towers et al. *Gymnasium*. Mar. 2023. DOI: 10.5281/zenodo.8127026. URL: https://zenodo.org/record/8127025 (visited on 07/08/2023).

[87] Máximo A. Roa and Raúl Suárez. "Grasp quality measures: review and performance". In: *Autonomous Robots* 38 (2015), pp. 65–88. DOI: 10.1007/S10514-014-9402-3.

[88] Eris Chinellato et al. "Visual quality measures for characterizing planar robot grasps". In: *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews* 35 (2005), pp. 30–41. DOI: 10.1109/TSMCC.2004.840061.

[89] Antonin Raffin et al. "Stable-Baselines3: Reliable Reinforcement Learning Implementations". In: *Journal of Machine Learning Research* 22.268 (2021), pp. 1–8. URL: http://jmlr.org/papers/v22/20-1364.html.

[90] Stephen Brawner. *ZeroMQ: An open-source universal messaging library*. URL: https://zeromq.org/.

[91] M. Safeea and P. Neto. "Model-based hardware in the loop control of collaborative robots: Simulink and Python based interfaces". In: *International Journal of Computer Integrated Manufacturing* 0.0 (2023), pp. 1–13. DOI: 10.1080/0951192X.2023.2177744.

[92] Tong Li et al. "Robot Grasping System and Grasp Stability Prediction Based on Flexible Tactile Sensor Array". In: *Machines* 9.6 (2021). DOI: 10.3390/machines9060119.

[93] Paola Ardón, Mauro Dragone, and Mustafa Suphi Erden. "Reaching and Grasping of Objects by Humanoid Robots Through Visual Servoing". In: *Haptics: Science, Technology, and Applications*. Ed. by Domenico Prattichizzo et al. Cham: Springer International Publishing, 2018, pp. 353–365. DOI: 10.1007/978-3-319-93399-3_31.