Pontifícia Universidade Católica
do Rio de Janeiro

Matheus do Nascimento Santos

AUV auto-docking approach based on reinforcement learning and visual servoing

Dissertação de Mestrado

Thesis presented to the Programa de Pós–graduação em Engenharia Elétrica, do Departamento de Engenharia Elétrica da PUC-Rio in partial fulfillment of the requirements for the degree of Mestre em Engenharia Elétrica.

Advisor: Prof. Wouter Caarls

Rio de Janeiro
October 2023

Pontifícia Universidade Católica
do Rio de Janeiro

Matheus do Nascimento Santos

AUV auto-docking approach based on reinforcement learning and visual servoing

Thesis presented to the Programa de Pós–graduação em Engenharia Elétrica da PUC-Rio in partial fulfillment of the requirements for the degree of Mestre em Engenharia Elétrica. Approved by the Examination Committee:

Prof. Wouter Caarls
Advisor
Departamento de Engenharia Elétrica – PUC-Rio

Prof. Leonardo Alfredo Forero Mendoza
Universidade do Estado do Rio de Janeiro - UERJ

Prof. Antonio Candea Leite
Norwegian University of Life Sciences - NMBU

Rio de Janeiro, October 13th, 2023

Matheus do Nascimento Santos

Graduated in Control and Automation Engineering from the Faculdade de Ciência e Tecnologia Área1 (Salvador-BA, Brazil).

Bibliographic data

Santos, Matheus do Nascimento

AUV auto-docking approach based on reinforcement learning and visual servoing / Matheus do Nascimento Santos; advisor: Wouter Caarls. – 2023.

139 f: il. color. ; 30 cm

Dissertação (mestrado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Engenharia Elétrica, 2023.

Inclui bibliografia

1. Engenharia Elétrica – Teses. 2. Acoplagem Automática. 3. Redes Neurais Convolucionais. 4. Aprendizado por Reforço. 5. AUV. 6. Simulação 3D. I. Caarls, Wouter. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Engenharia Elétrica. III. Título.

CDD: 621.3

To my parents, sister, and grandmother,
for their support and encouragement.

Acknowledgments

This Section is dedicated to all the people who helped me to overcome this interesting challenge.

First, I would like to thank God and my family who supported me during the hard days, without them, nothing would be possible.

Second, to my advisor, Professor Wouter Caarls, who helped proof, review, and provide guidance for accomplishing our initial promising goals. Without you, this work would have failed a long time ago. Thanks for accepting me as your student, I could learn a lot from you.

To my wonderful mother, Maria Elena Ribeiro, and sister, Isabel Nascimento. Thank you for your humor, for your love, for your strength and support. You make me who I am.

I could not fail to mention my grandmother, Matildes Umbuzeiro, aunt Maria Amelia Ribeiro, and father, Manoel da Lapa, who even living far from me were always there, praying and wishing the best I could have. Thank you, Guys, I love you.

Add CNPq, CAPES, PUC-RJ, the open source community, and esteemed Professors such as David Martins, Jacira Lucas, Oberdan Pinheiro, Marco Reis, and Rosely Bervian, who influenced my journey toward the knowledge roads. Furthermore, my educational foundation in Salvador's public schools instilled in me a sense of hope and resilience, even in the face of challenging circumstances.

To all my professors from PUC-RJ, and the examining board.

To the Technology Innovation Institute, for the support throughout the experiments phase.

To Rosianita Balena and Oceaneering, for the inspiration and the opportunity to face this challenge.

And finally, to all of my friends, Raimundo, Linder, Carlinhos, Leone, Sena, Samantha, Karen, Tayan, Mitro, Aline, Eduardo Lopes, Marco Xaud, Luciana Reys, Ivan Tarifa, Geovane Mimoso, Yuri Oliveira, Diogo Martins, and Gustavo Neves. Thank you for the good times!

From all my friends, I would like to share this achievement, especially, with Josmar Brito. You are part of it also. I hope you can be in a better place laughing and spreading happiness, as you have always done. We miss you.

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

Abstract

Santos, Matheus do Nascimento; Caarls, Wouter (Advisor). **AUV auto-docking approach based on reinforcement learning and visual servoing**. Rio de Janeiro, 2023. 139p. Dissertação de Mestrado – Departamento de Engenharia Elétrica, Pontifícia Universidade Católica do Rio de Janeiro.

In the growing field of underwater robotics, Automated Underwater Vehicles (AUVs) are becoming more important for a range of uses, such as exploration, mapping, and inspection. This dissertation focuses on studying the main challenges of AUV auto-docking, considering a customized 3D simulated environment. The research breaks down this challenging task into two main parts: cage pose estimation and AUV control strategy. Using a mix of traditional and new methods, including fiducial-based systems, Convolutional Neural Networks (CNN), and Reinforcement Learning (RL), the study carries out experiments to check system performance and limitations.

A significant aspect of this dissertation is using a 3D simulated environment to facilitate the development and testing of auto-docking algorithms for AUVs. This environment simulates crucial underwater dynamics, robotic sensors, and actuators, allowing for experimenting with different pose estimation techniques and control strategies. Additionally, the establishment of an RL-friendly 3D simulated environment stands as a relevant contribution, offering a reusable platform that not only validates the auto-docking algorithms developed in this study but also serves as a foundation for future RL-based underwater applications.

In summary, the dissertation explores a range of scenarios to evaluate the efficacy of various auto-docking techniques. It initially utilizes visual servoing along with a traditional PID controller, followed by the introduction of more advanced methods like CNN-based pose estimators and Reinforcement Learning controllers. These methods are assessed both individually and in hybrid combinations to gauge their suitability and limitations for understanding the main challenges behind the AUV auto-docking.

Keywords

Auto-docking; Convolutional Neural Networks; Reinforcement Learning; AUV; 3D Simulation.

Resumo

Santos, Matheus do Nascimento; Caarls, Wouter. **Técnica de acoplagem automática de AUV baseada em aprendizado por reforço e servovisão**. Rio de Janeiro, 2023. 139p. Dissertação de Mestrado – Departamento de Engenharia Elétrica, Pontifícia Universidade Católica do Rio de Janeiro.

No campo em crescimento da robótica subaquática, Veículos Subaquáticos Automatizados (AUVs) estão se tornando cada vez mais importantes para uma variedade de usos, como exploração, mapeamento e inspeção. Esta dissertação foca em estudar os principais desafios da acoplagem automática de AUVs, considerando um ambiente 3D simulado personalizado. A pesquisa divide essa tarefa em duas partes principais: estimativa da pose da garagem e estratégia de controle do AUV. Utilizando uma mistura de métodos tradicionais e novos, incluindo sistemas baseados em marcos fiduciais, Redes Neurais Convolucionais (CNN) e Aprendizado por Reforço (RL), o estudo realiza experimentos para verificar o desempenho e as limitações do sistema.

Um aspecto significativo desta dissertação é o uso de um ambiente 3D simulado para facilitar o desenvolvimento e o teste de algoritmos de acoplagem automática para AUVs. Este ambiente simula dinâmicas subaquáticas, sensores robóticos e atuadores, permitindo experimentar diferentes técnicas de estimativa de pose e estratégias de controle. Além disso, o estabelecimento de um ambiente 3D simulado amigável para RL representa uma contribuição relevante, oferecendo uma plataforma reutilizável que não apenas valida os algoritmos de acoplagem automática desenvolvidos neste estudo, mas também serve como base para futuras aplicações subaquáticas baseadas em RL.

Em resumo, a dissertação explora uma série de cenários para avaliar a eficácia de várias técnicas de acoplagem automática. Inicialmente, ela utiliza servo-visualização junto com um controlador PID tradicional, seguido pela introdução de métodos mais avançados, como estimadores de pose baseados em CNN e controladores de Aprendizado por Reforço. Esses métodos são avaliados tanto individualmente quanto em combinações híbridas para medir sua adequação e limitações para entender os principais desafios por trás da acoplagem automática de AUVs.

Palavras-chave

Acoplagem Automática; Redes Neurais Convolucionais; Aprendizado por Reforço; AUV; Simulação 3D.

Table of contents

List of figures

List of tables

List of Abreviations

2D – 2-Dimensional

3D – 3-Dimensional

ABE – Autonomous Benthic Explorer

AC – Actor-Critic

ADCP – Acoustic Doppler Current Profiler

AER – Azimuth-Elevation-Range

ANN – Artificial Neural Network

APS – Acoustic Positioning System

AUV – Autonomous Underwater Vehicle

CCD – Charge-Coupled Device

CNN – Convolutional Neural Network

CTD – Conductivity, Temperature, Depth

CUDA – Compute Unified Device Architecture

CURV – Cable-Controlled Underwater Recovery Vehicles

CVI – Close Visual Inspection

DFKI – Deutsches Forschungszentrum für Künstliche Intelligenz

DQN – Deep Q-Network

DDPG – Deep Deterministic Policy Gradient

DVL – Doppler Velocity Log

ECEF – Earth-Centered Earth Fixed

ENU – East-North-Up

FC – Fully Connected

FOV – Field Of View

GPS – Global Positioning System

GPU – Graphics Processing Unit

HDR – High Dynamic Range

HDRP – High Definition Render Pipeline

HROV – Hybrid Remotely Operated Vehicle

HVS – Hybrid Visual Servoing

IFREMER – Institut Français de Recherche pour l'Exploitation de la Mer

IBVS – Image-Based Visual Servoing

IMU - Inertial Measurement Unit

INS – Inertial Navigation System

LAUV – Light Autonomous Underwater Vehicle

LBL – Long Baseline

LGPL – GNU Lesser General Public License

LTS – Long Term Support

MBARI – Monterey Bay Aquarium Research Institute

NED – North-East-Down

NOC – National Oceanography Centre

OGRE – Object-Oriented Graphics Rendering Engine

OPEX – Operational Expenditures

OSFR – Open Source Robotics Foundation

OSG – Open Scene Graph

PBVS – Position-Based Visual Servoing

REP – ROS Enhancement Proposal

RELU - Rectified Linear Unit

RGB – Red, Green, Blue

RGBA – Red, Green, Blue, Alpha

RL – Reinforcement Learning

ROS – Robot Operating System

ROV – Remotely Operated Vehicle

RUR – Rossum's Universal Robots

SENAI CIMATEC – Serviço Nacional de Aprendizagem Industrial Centro

Integrado de Manufatura e Tecnologia

SIA – Subsea Asset Integrity Assurance

SLAM – Simultaneous Localization and Mapping

SSH – Secure Shell

TD – Time Difference U.S.A – United States of America

USBL – Ultra-Short-Baseline Localization

"I am the vine; you are the branches. If you remain in me and I in you, you will bear much fruit; apart from me you can do nothing."

John 15:5, *Holy Bible.*

1
Introduction

1.1
Robotics

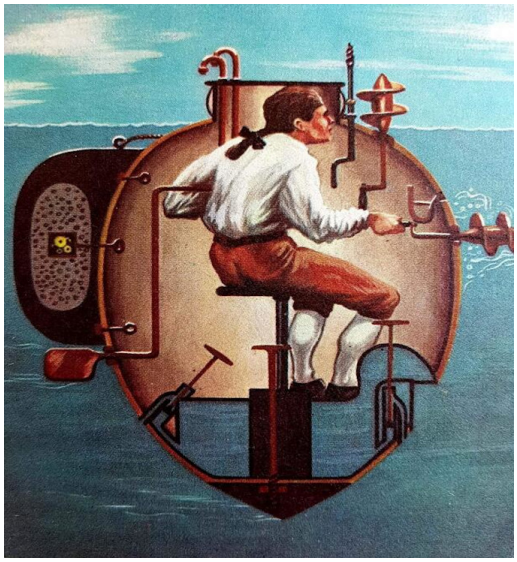
Robots have been incorporated into daily life over the last half-century: what was once only science fiction has now become a reality. Today, everyone living in the developed world benefits from the advances in robotics in everyday life [1]. Specifically, mobile robots have several applications in industries and factories [2]. These include transporting parts between gantries, conveyors, air tubes, and other processes, transportation among non-sequential processes, long-distance deliveries along winding and trafficked paths, and individualized item positioning at designated stations. However, a number of complex issues due to the unstructured and hazardous conditions make it difficult to operate in certain environments such as underwater, air, and space, even though today's technologies have allowed humans to land on the moon and robots to travel to Mars.

1.2
Marine Robotics

Considering those challenging environments, the underwater one is generally overlooked as most of the research focuses its attention on land and atmospheric issues. Consequently, the scientific society has not been able to explore the full depths of the ocean, and its abundant living and non-living resources [3].

In that context, marine robotics systems can help society to better understand marine and other environmental issues, protect the ocean resources of the Earth from pollution, support Oil and Gas companies to perform tasks at hazardous high depths, and efficiently utilize them for human welfare [4].

Considering the more contemporary innovations, one of the pioneers of the Autonomous Underwater Vehicles (AUVs) industry worth mentioning is the Autonomous Benthic Explorer (ABE) vehicle, illustrated in Figure 1.1. Designed for deep-sea mapping and data collection, it was launched for the first time in 1997 [5].



(a) Turtle MSV illustration.

(b) ABE launching.

Figure 1.1: Pioneer underwater vehicles. (a) Turtle, the first manned subversive vehicle in history [5], (b) ABE, one of the first industrial AUVs [5].

From the main types of subsea robots (gliders, autonomous surface vehicles, benthic crawlers, etc.), we are most interested in autonomous underwater vehicles, about which the number of projects and design proposals has been growing fast, recently.

1.3 Autonomous Underwater Vehicles

AUVs are unmanned, untethered robot submarines that operate fully independently to carry out pre-programmed operations and surveys [6]. According to [4], AUVs endurance typically ranges from a few hours to several days, although rapid technological developments are now bringing long-range operations within the possibilities, with endurance stretching to weeks or even months [7, 8].

Autonomous Underwater Vehicles are widely used in many fields of application: they are employed for scientific purposes (e.g., exploration and surveillance of archaeological sites), to complete industrial tasks at high depths (for instance they are exploited in the Oil and Gas industry), to carry out reconnaissance and patrolling missions in the military field, or even to conduct search and rescue duties [9].

Depending on their depth rating and size, AUVs can be equipped with a range of sensors (Conductivity, Temperature, Depth sensor, Acoustic Doppler Current Profilers, chemical sensors, photo cameras, sonars, magnetometers, gravimeters, etc.). However, the lack of tether, and hence of direct power input,

limits the sensor power consumption and duration of activity [9]. Therefore, the battery duration of AUVs is generally a limiting factor for a mission. This restriction adds the need for a surface support vessel to launch and recover the vehicle, something which increases the cost of the mission and makes the operational outcome more dependent on sea conditions [10].

In order to overcome this problem, a docking system is an alternative, since the AUV can dock and recharge its battery [11]. The docking system can be defined as a permanent structure on the seafloor, where the vehicle could charge the batteries and transfer the results of a mission, which would reduce the need for frequent launch and recovery operations at the surface, making the technology more cost-effective, safe, and robust. Also, it would enable the possibility of permanently residing AUVs ready for subsea operations, which would further extend the capabilities of the AUV technology. To this end, autonomous docking is required [10, 12, 13].

1.3.1 AUV Docking Systems

According to [14], underwater docking systems are what enable unmanned charging and data collection for AUVs, and have been actively studied since the last century [15]. Conventional AUV docking systems are usually custom-designed to match up with specific AUV outlines and structures [16, 17, 18].

The idea of a docking station garage aims to provide the required interfaces to support the AUV during the execution of the target missions. Docking station garages perform as fully integrated service stations with improved capabilities, not merely as locations for charging and data transfer. These garages may hold AUVs in a secure setting, greatly reducing the dangers involved in open-water deployments. Additionally, these garages are made to be modular and adjustable, allowing them to house a variety of AUV sizes and types. They act as central nodes where numerous AUVs can dock concurrently for upkeep, data synchronization, and energy replenishment, streamlining operations and lowering overhead. Currently, there are many concepts for AUV garages, three variants are commonly used [15]:

– **Framed modular garage**: This concept consists of a prismatic frame constructed from tubular beams forming the corners, connected as an open structure. The advantage of this solution is a low added mass relative to the internal storage volume enveloped. However, in the simplest implementation, this solution would not allow a smooth AUV re-entry and would need to be designed specifically for each vehicle cross-

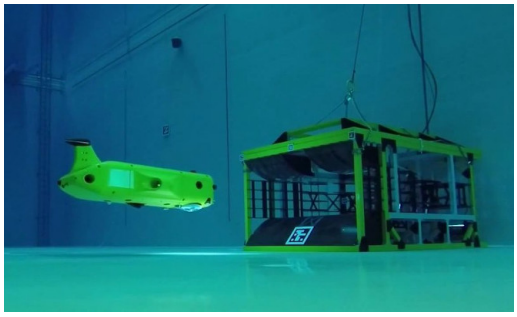
section. Figure 1.2 illustrates an example of a framed modular docking station built for the FlatFish AUV project [19].

– **Tubular garage**: This concept consists of an open tube of a diameter larger than that of the AUV. It is convenient for cylindrical or torpedo-shaped AUVs, for which the lost volume is minimum and the added inertia associated with the trapped water and the hydrodynamic added mass is limited. This concept could have two add-on variants to make docking easier and to allow adaptation to a variety of AUV diameters. First, a propeller could be added to help the deceleration of non-hovering AUVs just prior to and during mating with the garage in the case of bottom support, generating an artificial current against the approaching vehicle. Second, by a pumping-out effect, a guiding bottle would help stop non-hovering vehicles after they had entered the garage. Since torpedo-shaped AUVs are very popular due to their excellent hydrodynamic characteristics, most docking systems are tailored for such kinds of AUVs [20, 21, 22].

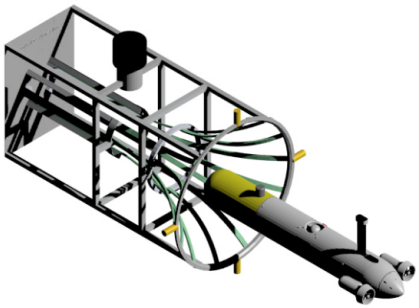
  These docking stations utilize cone-shaped entrances to provide the AUVs with positioning tolerance and collision guidance. Figure 1.2 depicts the docking system designed for the SPARUS II AUV [23].

– **Flying Garage**: A novel concept of docking station garages that can be deployed in areas where the seabed is not accessible [24]. Figure 1.2 shows a flying garage designed for the DeepLeng AUV [25].

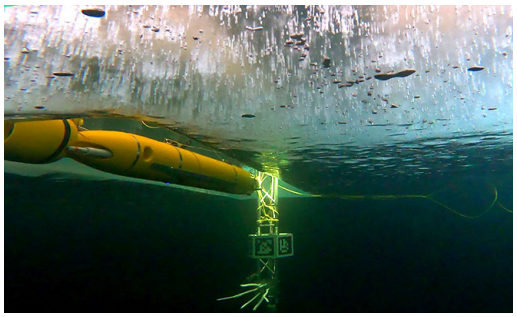
One of the challenges associated with docking systems is related to the localization of this structure by the AUVs. In general, some steps are considered in order to localize and dock the AUV properly. This challenge is defined by the literature as homing [15]. The homing operational phase includes a number of aspects, the principal ones being relative position identification, navigation control, and target detection. In some cases, the solutions will be predicated on the capability of the AUVs, while in others, specific solutions will have to be identified [16, 17].



(a) Flatfish framed modular docking station.



(b) SPARUS II AUV torpedo-shaped docking station design concept.



(c) AUV DeepLeng Flying docking station.

Figure 1.2: Docking station garages examples: (a) FlatFish framed modular garage, developed by SENAI CIMATEC and DFKI [19], (b) Toperdo docking station concept of Sparus II AUV. An acoustic modem on top (black) and 4 light beacons (yellow) at the front [23], and (c) AUV DeepLeng Flying docking station [25].

Different techniques can be applied in order to perform homing, such as using acoustic systems [23], electromagnetic [20], optics [21], and image-based. According to [26], cruising-type AUVs go straight ahead and enter the cylinder-shaped or corn-shaped seafloor station. On the other hand, hovering-type AUVs land at the seafloor station from above because the shape of hovering-type AUVs is more complicated than cruise-type AUVs. In order to complete the docking procedure, the AUV must estimate the relative position of the station in real-time with sufficient accuracy. In underwater fields, acoustic signals and visible light are available for relative positioning (Section 2.3.2 details it). In general, the acoustic signal is available up to long distances but with low resolution. The visible light has opposite properties [26]. Therefore, image-based systems are considered since they can deliver equivalent and even more efficient results, saving costs when compared to the other techniques.

To align with the terminology commonly used in relevant applications, the term “docking garage” will be referred to from now on as the “cage”. Throughout this work, more details are provided about the homing task since

the primary goal of it is to propose a homing localization approach.

1.4 Motivation

As described in the previous Section, one of the concerns associated with the usage of docking stations is related to the AUV's approach to localize the station and the required steps to docking it correctly on the target structure. For harsh underwater environments, the self-localization of AUVs is the basis for accomplishing the aforementioned tasks, which could guarantee the proposed work.

Significant resources are required to ensure proper self-localization of AUVs without available reference signals provided by the Global Positioning System (GPS) or the acoustic positioning systems such as Long Baseline (LBL) [14]. The state-of-the-art applications consider onboard sensors based on visual feedback due to low cost and performance, which in most cases attend the imposed requirements [27, 28, 29].

However, the localization using vision-based algorithms applies visual landmarks to create visual maps of the environment, which can result in a complex task due to the abrupt dynamic light conditions, decreasing visibility with depth and turbidity, and image artifacts like aquatic snow [30]. Moreover, underwater images are essentially characterized by their low visibility once the light is exponentially attenuated as it travels in the water and the scenes result in poorly contrasted and hazy [31]. The extent to which the robot navigates, the map grows in size and complexity, increasing the computational complexity to solve the target algorithms.

Given the mentioned challenges associated with underwater image processing, the main performance impacts are related to the computational overhead that makes real-time processing difficult. This is where machine learning comes into play as a promising solution. Machine learning algorithms are adept at handling complex data and can be trained to better understand and interpret underwater imagery [32]. They can adapt to the unique challenges of the underwater environment and offer improved accuracy and robustness in navigation tasks [31]. Recent localization research developments have witnessed dramatically increasing performance in autonomous driving [33], autonomous micro aerial vehicles [34], augmented reality, and virtual reality [31], which promoted the rapid development of novel machine learning-based algorithms. According to [2], inspired by the machine learning algorithms on the ground and in the air, the underwater approach has been increasing the number of academic research in this field, especially in laboratory circumstances with

high accuracy and robustness.

Therefore, one of the primary motivations behind this research is to simplify the docking homing process for AUVs by studying and implementing machine learning techniques. Conventional approaches often struggle with unique and challenging underwater conditions, such as fluctuating lighting and limited visibility. Implementing machine learning might offer a more efficient and robust solution to these challenges. Moreover, considering machine learning techniques for this research not only addresses immediate challenges in AUV auto-docking but also lays the foundation for future research and advancements in this field.

However, in machine learning applications, environment interpretation, based on images is usually formulated as a pixel labeling task. Given an underwater image, the goal is to produce either a complete semantic segmentation of the image into classes such as marine life, water, static objects, and dynamic objects or a binary classification of the image for a single object class [35]. As training machine learning and deep learning models require huge amounts of carefully labeled data, this task, also referenced as data labeling or data annotation, can impose an unfeasible scenario for a complex application due to the occurrence of noisy labeling.

To address the issue of noisy labeling, the scarceness of specialized automated tools for data labeling in underwater environments, and the unfeasibility of obtaining a sizable dataset for training, this work proposes, as its primary focus, the encouragement of applying Reinforcement Learning (RL) algorithms into the underwater environment to provide an agent capable of performing autonomous tasks by using its policy along with visual servoing techniques to guide a simulated AUV during the docking homing process, also referenced in this work as auto-docking task.

Finally, the implementation of the RL paradigm in simulated underwater environments, as demonstrated in this work, serves as an important stepping stone for the field since the use of simulation-based RL allows for a risk-free, cost-effective way to iteratively improve the autonomous task capabilities, by evaluating the agent's performance under different conditions. In addition, it encourages researchers to adopt and experiment with RL paradigms for underwater robotics, speeding up the solution's prototyping and testing and validation cycles.

1.5 Goals

Therefore, the main goals of this work field of research are to investigate, explore, and conduct experiments towards the AUV's auto-docking task on a 3-Dimensional (3D) simulated environment. A key component of this research is the application of machine learning methods, specifically targeting the challenge of localization — a fundamental aspect of AUV's navigation subject.

In the context of this research, the task of auto-docking for AUVs focused on the localization aspect is broken down into two main challenges. The first is determining the relative position and orientation of the docking garage, or "cage", with respect to the AUV, referred to as **cage pose estimation**. The second involves proposing a **control strategy** to guide the hovering AUV toward the identified cage pose effectively. Figure 1.3 depicts the high-level architecture proposed to achieve the described goals.

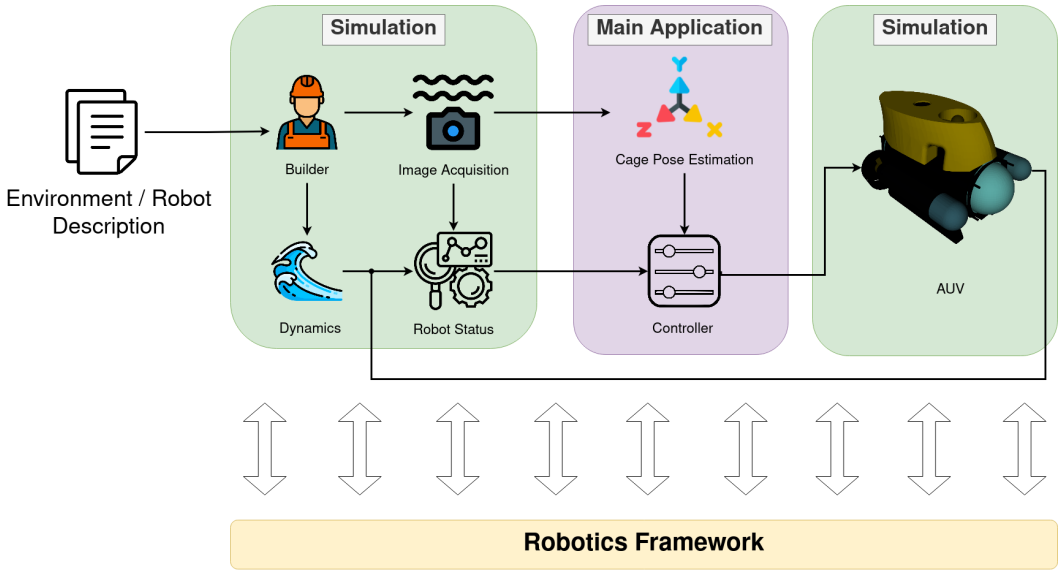


Figure 1.3: Proposed AUV auto-docking system high-level architecture design.

1.6 Proposed solution

As depicted in Figure 1.3, the system is composed of three main subsystems: simulation bringup, the main application, and the AUV simulation. The first one is responsible for receiving the environment and robot description file and setting up the simulation environment accordingly. The environment setting up covers the configuration of image acquisition simulated sensors, the underwater dynamics physics, and all related components required to make the simulation environment meet these work requirements.

The second subsystem is responsible for the functionalities of cage pose estimation and the AUV controlling.

The last subsystem consists of the AUV's simulated onboard sensors and actuators. The main workflow starts off from the simulation environment setting up and proceeds to the image acquisition step. The data acquired is processed and forwarded to the cage pose estimation algorithms, in order to identify the next action taken by the controller expecting to localize the target object (docking cage). One of the goals of this work is to identify the docking station by testing different machine learning methods, since this technique may present different result performances based on the application scope.

Figure 1.4 delineates the sequence of operations required to execute the auto-docking task, in accordance with the specifications outlined in the scope of this research project.

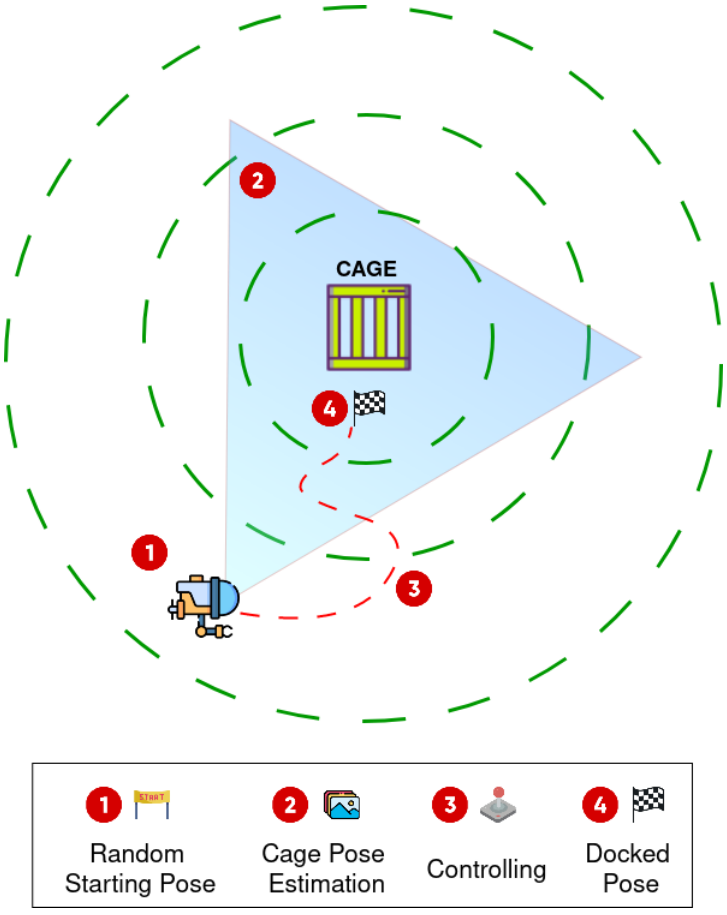


Figure 1.4: Proposed AUV auto-docking steps to perform the task.

As illustrated in Figure 1.4, the auto-docking task comprises four main steps: starting pose, cage pose estimation, controlling, and docked pose. The first one consists of a random starting position within a predefined range, which should guarantee that the cage can be within the AUV camera's Field

Of View (FOV). This approach will guarantee the evaluation of the capability of generalization of the obtained results.

The second step consists of the cage pose estimation, which should receive the image stream from the AUV camera and output the target cage pose relative to the robot. This modular component should be self-contained, with a well-defined interface, aiming at making possible the interchanging between different techniques.

The third step is responsible for taking the controlling actions to guide the AUV toward the cage estimated pose. As the cage pose estimation component, it should have a modular architecture, to make possible the evaluation of different techniques.

The last step consists of reaching the docked pose, which, in this work, is considered as a pose outside the cage since the main challenge is regarding localization, not the fine controlling to avoid collisions.[1]

Furthermore, the work aims to investigate the relevant metrics to evaluate the auto-docking process performance in order to make feasible the application of the proposed approach in a real field test.

1.7 Contributions

From the content presented in the following Chapters, this work goes through the required steps to carry out experiments toward auto-docking. During those steps, the following contributions can be highlighted, as an overview:

– **3D Simulated Underwater Environment for Auto-docking:** One of the major contributions is the creation of a 3D simulated environment tailored for auto-docking applications. The simulation includes all the necessary resources and underwater dynamics required for representative testing. In that context, a unique addition to the UUV Simulator branch is an underwater LED plugin (illustrated in Figure 4.8), specifically implemented to enhance the realism and utility of the simulated environment for auto-docking tasks. This addition provides an extra layer of complexity and is useful for scenarios requiring visual cues or markers. Additionally, the simulated environment is compatible with Reinforcement Learning algorithms, allowing for real-time training and testing.

[1]Usually, the docked pose is inside the docking garage with all required connectors plugged to it. However, other controlling techniques can be considered to tackle this specific challenge, as presented in [22].

- **Supervised Learning-Based Pose Estimation:** A Convolutional Neural Network was trained for pose estimation using a dataset that did not require labeling. This streamlines the process and significantly reduces the overhead associated with manual data labeling. Moreover, transfer learning techniques were applied from architectures that initially were trained to interact with another type of paradigm. More details are provided in Section 3.1.2.
- **Reinforcement Learning Encouragement:** This work figures as an encouragement for future underwater research considering reinforcement learning methods to solve complex tasks. As an example of a complex task, as presented in this work, a Reinforcement Learning agent is proposed to control the AUV under different scenarios. Then, an assessment of the agent's performance is done, considering a PID controller as a reference, aiming to illuminate the strengths and weaknesses of each approach in an auto-docking context, providing valuable insights for future work.
- **Knowledge Foundation for Future Challenges:** This research serves as a reference, for the authors, for lessons learned for future related works. Additionally, it can be considered as a relevant technical starting point for the authors and other researchers to investigate deeper and more complicated challenges related to AUVs and underwater robotics.

1.8 Text Structure

This document is structured as follows:

This Chapter presented a brief introduction to the study covering the motivation for performing the auto-docking task using machine learning, RL, and Visual Servoing techniques; the state of the art of AUVs; a brief introduction to docking systems; and the main goals behind this project.

In Chapter 2, we present the background that was relevant to our challenge, which encompasses computational resources, agreed-upon conventions, intricacies of the simulation environment, supporting theories, and state of the art of the main subjects with a focus on underwater applications.

In Chapter 3, we present the methodology adopted to carry out the experiments and to evaluate the methods. Additionally, it should cover the theoretic formulation, implemented algorithms, and evaluation metrics.

In Chapter 4, the procedure required to enable the experiments' execution is presented.

In Chapter 5, the obtained results are presented, focusing on key performance metrics. A subsequent section provides a comparative analysis to evaluate the relative advantages and disadvantages of each proposed method.

Finally, in Chapter 6 we present our conclusion, based on the results presented in Chapter 5, and suggest future works, that might add relevant progress to this project.

2
Background

In the present Chapter, the background concepts required for carrying out the auto-docking task, considering a simulated environment, are detailed. These consist of the simulation environment, the method to detect the cage, and the control algorithm.

2.1
Simulation Environments

A 3D virtual world is an unreal environment represented considering the same dimensions of the application working area. The concept is to link digital technology and computer vision and become a tool to carry out engineering studies, design analysis, and architectural projects [36]. It has been applied to various applications such as the simulation of manufacturing plants, the planning of robotic work cells, and robot operating systems.

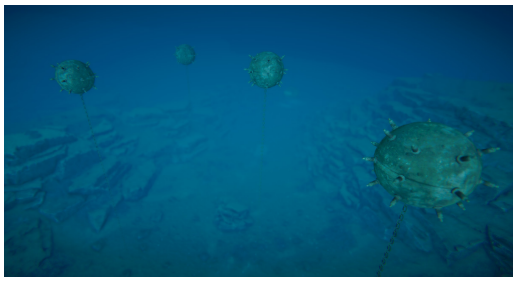
The 3D underwater environment simulation imposes some challenges intrinsically related to the difficulty of realistically modeling hydrodynamic forces acting on the robot itself and the complex environments in which the operations take place. According to [37], the main requirements to meet the main challenges faced by the underwater environment simulation are:

- Physical fidelity for the simulation of rigid-body dynamics and collisions;
- Interface with robotics middlewares, which are commonly used by the vehicles and systems to structure the software scope (e.g., ROS, ROCK);
- Low complexity for the setup of new world scenarios and robot models;
- Extensibility for integration of additional modules;
- Adequate documentation;
- Regularity of updates and maintenance;
- Capability of multi-robot simulation;
- Open-source application with permissive licenses (e.g., MIT, BSD, Apache).

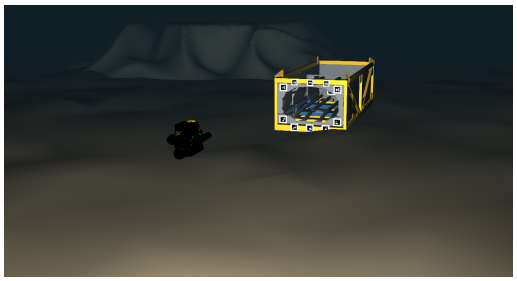
At the first stage, three simulation tools were considered as potential choices. Unity3D [38], UUV Simulator [37] and UWSim [39] (Figure 2.1)

meet all of the requirements mentioned above. However, they present different functionalities that figure as a trade-off. Thus, a trade study was performed in order to define the most appropriate tool, considering the scope of this work. The following Sections detail the evaluated aspects of the trade study.

(a) UWSim.



(b) Unity3D.



(c) UUV Simulator.

Figure 2.1: Potential simulators considered for this work, primarily: (a) UWSimulator, (b) Unity3D, and (c) UUV Simulator.

2.1.1 Simulator Tools Overview

UWSim and UUV Simulator are ROS-based tools that provide third-party rendering and physical engines. Meanwhile, Unity3D is a general-purpose simulation tool commonly used as a gaming development platform. It uses PhysX by NVIDIA [40] as a physics engine to simulate the behavior of objects regarding their physical environment and forces acting upon them. For rendering, it considers third-party assets that can be purchased or shared by other users.

According to [38], UWSim is a tool that has been implemented in *C++*, makes use of the Open Scene Graph (OSG) and osg Ocean libraries for rendering, and a built-in package aimed to provide the required underwater dynamics. Moreover, it provides an XML-based interface to describe the elements present in the simulation environment. In addition, [37] defines the UWSim as a tool that offers a wide range of sensor models, provides realistic renderings of underwater environments due to its graphics engine OpenSceneGraph library

and has been used in several academic publications. The physics engine is used only for handling contact forces and the implementation of the vehicle dynamics, including the simulation of thruster forces, it is located in one monolithic ROS node, but it could be modified to adhere to a more modular structure or interfaced with an external platform such as Matlab, if necessary. Setting up a new simulation, however, requires the configuration of the scenario, vehicles, and other objects in a single XML description file, which can make this task laborious.

According to [37], UUV Simulator is an extension for Gazebo, which supports multiple underwater vehicles (ROVs and AUVs) and robotic manipulators with the high-fidelity representation of hydrostatic and hydrodynamic forces. Several commonly used sensors are included, e.g., underwater camera, pressure sensor, Inertial Measurement Unit (IMU), Magnetometer, DVL, etc. Models for fins and thrusters are also included for actuation. UUV Simulator allows researchers to create complex underwater environments with models already included for seabeds, lakes, shipwrecks, etc. It uses the Object-Oriented Graphics Rendering Engine (OGRE) as a rendering engine and counts on the Gazebo hydrodynamics built plugin to simulate the underwater dynamic.

2.1.2 Criteria Definition

According to [41], in order to evaluate the performance of a mobile robot based on a simulation, the simulation must be of sufficient precision and accuracy.

From the [42] perspective, similar to other machine learning applications, for the Reinforcement Learning methods, the acquisition of training data is one of the more challenging tasks. The quality of training data closely correlates to the quality of the simulation environment generating this data. Thus, the quantity of the training data depends on the speed of the simulation environment. Therefore, developers need a reliable simulation environment with a physical model representing their process environment well enough to produce meaningful data. Based on that, this Section aims to detail the criteria adopted to define the simulation tool for emulating the required environment to accomplish the goals of this work. [43] defines metrics to compare Gazebo and Unity simulators and supports the criteria definition for this work. The criteria aspects definition are described below, according to the scope of this work:

– **Graphical fidelity**: This metric is not commonly considered in the benchmarks works that compare the performance simulators

[44][41][45][46]. However, as this metric impacts directly the RL algorithms performance [44], it was considered. Visual Technologies availability such as High Dynamic Range (HDR) [41], Reentrancy Computing [44], GPU processing [45], and shader runtime compilation [46] were considered.

- **Physics engine**: Underwater hydrostatic and hydrodynamics physics representation support.
- **Community support**: Maintenance frequency, discussion forums, number of projects already supported, and documentation.
- **AUV basic motion structure (sensor, joints, links) description**: The complexity of representing the AUV sensors and actuators elements and their respective physics along with ROS.
- **Underwater rendering plugins (or assets) availability**: Number of software components capable of representing the visual dynamic of the underwater environment (e.g., fog, suspended particles, and vortex).
- **Computational performance**: Real-time computing and resources allocation.

Table 2.1 concentrates on the main aspects evaluated during the trade study. All of the simulators were tested and the technical information was validated, aiming to choose the best option for the proposed work.

From the first aspect, graphical fidelity, Unity has a considerable advantage over the others, since it has the High Definition Render Pipeline (HDRP)[1], which provides a remarkable graphical performance when compared to the other alternatives.

For the physics engine, UWsim and UUV Simulator meet the requirements for underwater simulation. However, UUV Simulator has a slight advantage since it also simulates the plume caused by the interaction between the AUV thruster forces and the water. Unity does not have a specific asset for simulating both hydrodynamics and hydrostatics.

Considering the community support aspects, only the UWSim does not provide a consistent level of maturity since there is no well-structured documentation, dedicated forums, or updated repositories.

[1]According to [40], HDRP is a high-fidelity scriptable render pipeline built by Unity to target modern (Compute Shader compatible) platforms. HDRP utilizes physically based Lighting techniques, linear lighting, HDR lighting, and a configurable hybrid Tile/Cluster deferred/Forward lighting architecture. It gives the tools needed to create applications such as games, technical demos, and animations to a high graphical standard.

Regarding the AUV description capabilities, UUV Simulator provides all interfaces compatible with ROS. Meanwhile, the other alternatives only provide one ROS-compatible interface.

Evaluating the underwater rendering plugins, UWSim has an advantage since the representation of visual effects and phenomenons exceeds the expectations for this type of application.

In terms of computational performance, according to the documentation of each alternative, they are capable of delivering real-time-based simulations. Unity3D has an advantage since it can use GPU programming to leverage the simulation performance.

Table 2.1: Simulators trade study assessment.

Criteria	**UWSim**	**Unity3D**	**UUV Simulator**	**Winner**
Reality fidelity	OpenGL	HRP Up to 8k HDR NVidia-based	OpenGL	Unity3D
Physics engine	Hydrostatic Hydrodynamic		Hydrostatic Hydrodynamic Plume	UUV Simulator
Community support	Poor Documentation Outdated Repository ROS Support	Massive Documentation Updated Repository	Massive Documentation Updated Repository ROS Support	UUV Simulator
AUV Description	XML	URDF	URDF SDF XACRO	UUV Simulator
Rendering Plugin	OSG Fog and Debris	Crest Ocean	Gazebo Plugin	UWSim
Performance	Real Time	Real Time GPU Programming	Real Time	Unity3D

Therefore, the chosen simulator was the UUV Simulator since, beyond the discussed features, it also provides a set of AUVs that already has the control algorithms stack implemented and tested, in some cases even in relevant fields.

2.1.3 UUV Simulator

UUV Simulator consists of a Gazebo-based package that provides a well-structured environment to simulate unmanned underwater vehicles. In this work, UUV Simulator is responsible for delivering the AUV simulation (including motion, sensing, and control), the physics simulation (hydrodynamic and hydrostatic), the water rendering pipeline, and the ROS stack, including the interaction with the sensor's data, frames transformation and all required data that the simulation should provide. Figure 2.2 depicts the UUV Simulation software structure.

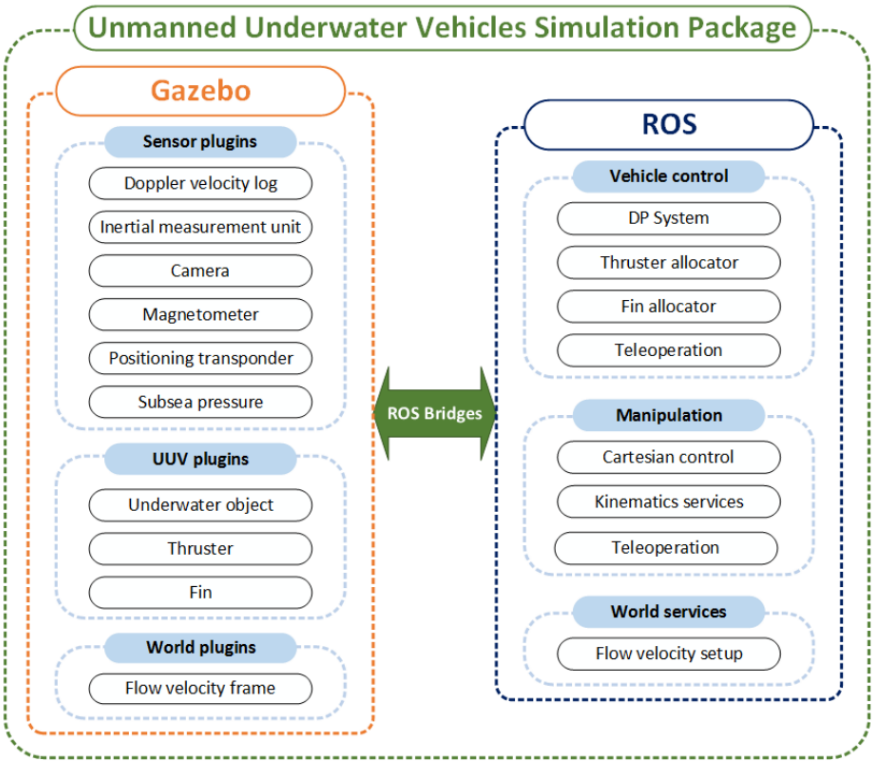


Figure 2.2: UUV Simulator software stack [37].

UUV Simulator uses Gazebo 9 as a backbone. Gazebo [47] is the primary simulation tool of the ROS community. As ROS is used for an increasing amount of robots [41], Gazebo has extensivly been proven a useful tool for simulating robots specifically mobile robots.

As the physics simulation engine, UUV Simulator uses the Open Dynamics Engine (ODE). ODE is a free, industrial-quality library for simulating

articulated rigid body dynamics that uses XWindows and OpenGL to render the scene being simulated.

As the visualization framework, OGRE is used. The OGRE library is a scene-oriented, flexible 3D engine written in *C++* designed to make it easier and more intuitive for developers to produce applications utilizing hardware-accelerated 3D graphics. The class library abstracts all the details of using the underlying system libraries like Direct3D and OpenGL and provides an interface based on world objects and other intuitive classes.

2.2 Auto-Docking

As introduced in the previous Chapter, the auto-docking task can be categorized as a "localization" challenge, which figures as one of the main aspects of underwater autonomous navigation. According to [48], Underwater navigation is a challenging work from the aspect of control, construction, and design, one must face constraints that are not encountered in other environments. Naturally, there is a great research opportunity in the implementation and designing of new algorithms and technology for the navigation and localization of AUVs.

According to [49], due to the lack of underwater GPS, localization in the ocean is much more challenging than on land. Common approaches for obtaining a submerged device's position are based on acoustic pings, such as Ultra-Short-Baseline Localization (USBL) or LBL [10]. From [12], Inertial Navigation System [13] and Dead-Reckoning [13] are methods for navigation and localization of AUVs. These systems depend on acceleration, water speed, and vehicle velocity to determine the AUVs' position. They don't need to send/receive signals externally, ideal for the long-range mission [10], the main issue is position error, which grows over time, termed accuracy drift [10].

None of the techniques mentioned is a perfect solution to the challenges, in practice it is common for a vehicle to employ a combination of these procedures. However, in this work, the visual servoing approach is considered since one of the premises of this work, as illustrated in Figure 1.4, is to start the auto-docking task at a certain distance from the cage, making possible the visualization of it and hence the implementation of this technique.

Therefore, the next Section provides an overview of the Visual Servoing paradigm, followed by the methods considered to be implemented to estimate the cage and control the AUV. For each of them, it is provided a "Work Scope" subsection describing the method's role within the scope of this work.

2.2.1 Visual Servoing

According to [50], movement and control-related tasks for autonomous vehicles generally consist of a complex challenge, requiring integration of different areas of engineering, including computer vision, dynamic systems modeling, and robot control. The vehicles are developed mainly using two approaches: path planning and visual servoing, both of which allow guiding a robot through a series of positions in the joint or task space, from an initial to a desired position.

Visual servoing is a well-known approach to guide robots using visual information. Image processing, robotics, and control theory are combined in order to control the motion of a robot depending on the visual information extracted from the images captured by one or several cameras [51], in which case the motion of the robot induces camera motion, or the camera can be fixed in the workspace so that it can observe the robot motion from a stationary configuration [52]. Usually, it consists of two intertwined processes: tracking and control. Tracking provides a continuous estimation and update of features during the robot/object motion. Based on this sensory input, a control sequence is generated. In addition, the system may also require an automatic initialization part which may include figure–ground segmentation and object recognition [50]. Figure 2.3 depicts this process.

A typical example is to extract image features, recognize the desired object by matching image features to a geometrical model, and compute its position and orientation (pose) relative to the camera (robot) coordinate system. This Cartesian–space information is used to move the robot to the desired pose. Considering the conventional methods, to estimate the pose of the object, the model of the object must be available. To move the robot based on the visual information extracted in the camera frame, the visual sensor has to be calibrated with respect to the robot [41]. The previous example demonstrates a typical challenge covered by this field, also known as the position-based visual servoing system. In this work, this type of system is considered, since the auto-docking challenge can be applied to it.

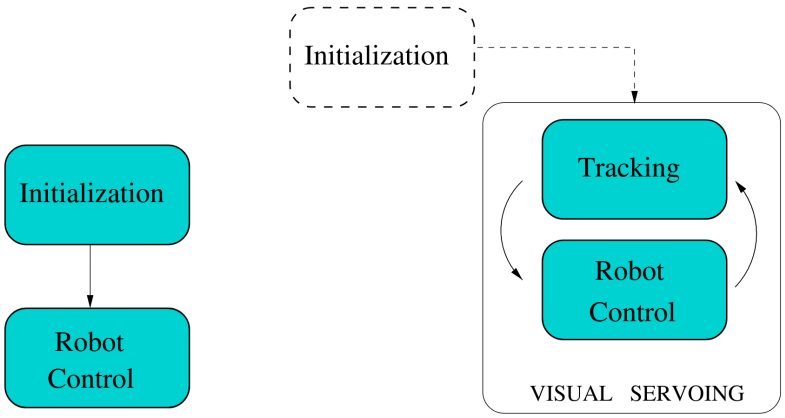


Figure 2.3: Major components of a visual servoing system: Initialization (visual servoing sequence is initialized), Tracking (the position of features used for robot control are continuously updated during the robot/object motion), Robot Control (based on the sensory input, a control sequence is generated)[52].

The Visual Servoing approach arose in 1979 when Hill and Park ([53]) introduced the term visual servoing to distinguish their approach from earlier work. In 1980, the following taxonomy of visual servo systems was introduced in [53]:

- **Dynamic look-and-move systems**: These systems control the robot in two stages: the vision system provides input to the robot controller that then uses joint feedback to stabilize the robot internally. As pointed out by [53] nearly all of the reported systems adopt this approach.
- **Direct visual servo systems setup**: Here, the visual controller directly computes the input to the robot joints, and the robot controller is eliminated.

Considering the first taxonomy, according to [54], currently, three types of visual servoing exist depending on how this information is used: image-based visual servoing (IBVS), position-based visual servoing (PBVS) and hybrid visual servoing (HVS). IBVS control [53], uses coordinates on an image plane corresponding to observation points S that vary over time and are used to calculate an error associated with the reference, which is then used to calculate a compensation signal to guarantee its convergence to zero. PBVS control [53], takes the object's orientation and positioning as parameters that are compared to the reference to calculate its error, while the HVS control method [53], is based on an integration of the above methods to formulate the control law. Visual control can also be classified according to the camera position. If the camera is static in the robot's workspace, it is called an eye-to-hand

configuration. If the camera is positioned on the robot, i.e., if the robot's movement moves the camera, the configuration is called eye-in-hand.

As mentioned before, this work is based on the position-based control system. Therefore, this system should be covered in this Section and throughout this document.

2.2.2 Position-Based Control

According to [41], PBVS is usually referred to as a 3D servoing control since image measurements are used to determine the pose of the target with respect to the camera or some common world frame. The error between the current and the desired pose of the target is defined in the task (Cartesian) space of the robot. According to [55], the advantage of the PBVS is that the pose of the end-effector can be controlled relative to the target directly and naturally, while the drawbacks are that the pose and motion estimation is prone to camera calibration errors, target model accuracy, and image measurement noise. These challenges have been successfully addressed by many researchers to eliminate image errors caused by an uncalibrated camera and suppress the image noise due to the vibration of the camera resulting from flexible manipulators [54]. Figure 2.4 illustrates the frames involved in a PBVS application.

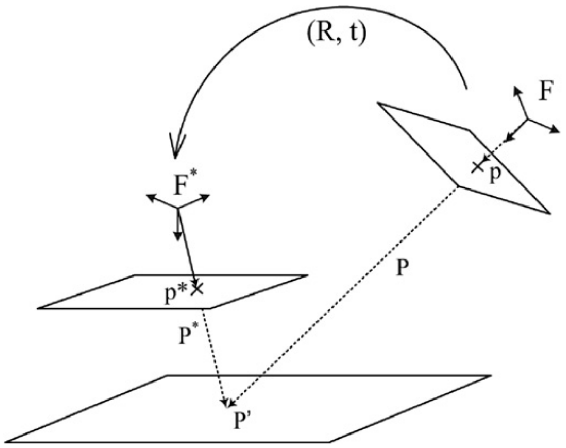


Figure 2.4: Camera projection diagram showing the desired ($F*$) and the current (F) frame, where R and t denote the rotation and translation from the robot to the target object, respectively [56].

Using a conventional PBVS method, as a premise, a 3D camera calibration is required in order to map the 2D data of the image features to the Cartesian space data. This is to say that the intrinsic and extrinsic parameters of the camera must be evaluated. Intrinsic parameters depend exclusively on the

optical characteristics, namely, lens and Charge-Coupled Device (CCD) sensor properties. The calibration of intrinsic parameters can be operated offline in the case that the optical setup is fixed during the operative tasks of the robot. Extrinsic parameters indicate the relative pose of the camera reference system with respect to a generic world reference system. It is assumed that the world reference system is exactly the object frame so that the extrinsic parameters directly give the pose of the camera with respect to the target. Obviously, the extrinsic parameters are variable with a robot or target motion, and an online estimation is needed in order to perform a dynamic look-and-move tracking task [52]. Section 3.1.1 depicts the mathematical background to compute the camera's intrinsic parameters.

Moreover, PBVS includes methods based on analysis of 2D features or direct pose determination using 3D sensors. In order to recognize those reference features in the scene, plenty of techniques can be considered, such as photogrammetric, stereo vision, and depth from motion. Another common technique used in this field is based on the usage of fiducial markers. Figure 2.5 shows the block diagram that represents the PBVS control chain, which highlights the feature's identification of a crucial role in the system.

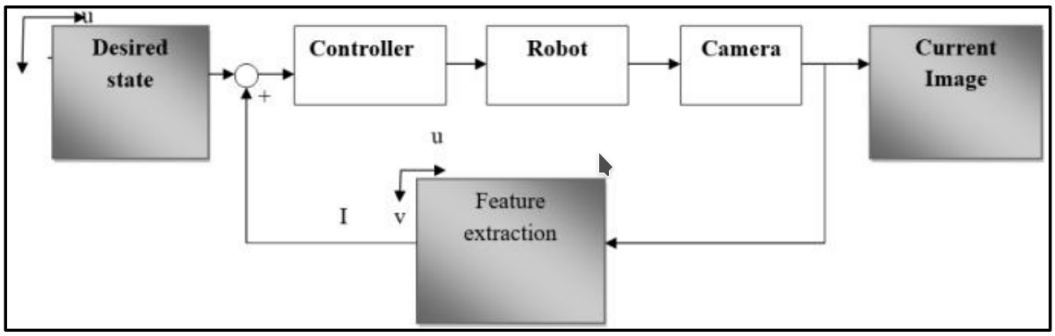


Figure 2.5: Typical Block Diagram of a Position Based Control [54].

The present study investigates the utilization of fiducial markers as a means of facilitating feature extraction and recognition, as previously referenced. Accordingly, a comprehensive examination of this methodology is presented in the ensuing section.

2.3 Fiducial Markers

Visual fiducials are artificial landmarks designed to be easy to recognize and distinguish from one another. Although related to other 2D barcode systems such as QR codes [57], they have significant goals and applications. With a QR code, a human is typically involved in aligning the camera with

the tag and photographs it at a fairly high resolution, obtaining hundreds of bytes, such as a web address. In contrast, a visual fiducial has a small information payload (perhaps 12 bits). Still, it is designed to be automatically detected and localized even when it is at very low resolution, unevenly lit, oddly rotated, or tucked away in the corner of an otherwise cluttered image. Aiding their detection at long ranges, visual fiducials are comprised of many fewer data cells: the alignment markers of a QR tag comprise about 268 pixels (not including required headers or the payload), whereas the visual fiducials commonly range from about 49 to 100 pixels, including the payload [58].

According to [58], in the pursuit of retrieving the 6-DOF configuration of the servoing target, artificial features (fiducials markers) could largely increase the tracking precision and robustness and reduce the computational cost. More importantly, extra information could be computed to assist better visual servoing, such as scale, depth, and rotations.

In underwater robotics, the most commonly used fiducial markers are passive and active markers that can be detected by imaging devices such as sonar, LiDAR, or cameras. Passive fiducial markers such as retro-reflective spheres or reflective tape are often used for underwater positioning and tracking systems. These markers reflect sound waves or light emitted by the imaging device back to the sensor, making them easily detectable. Retro-reflective spheres, for example, are widely used in underwater robotics applications as they are highly visible in sonar and LiDAR scans and can be easily attached to underwater structures or vehicles. Active fiducial markers, such as acoustic beacons or LED lights, are also used in underwater robotics. These markers emit signals that can be detected by sensors, allowing for accurate tracking and positioning in underwater environments where passive markers may not be visible or easily detectable [12].

The AprilTags and ArUco markers are the most common types of fiducial diffused in underwater applications. In this work, the usage of AprilTags is considered.

2.3.1 ArUco Markers

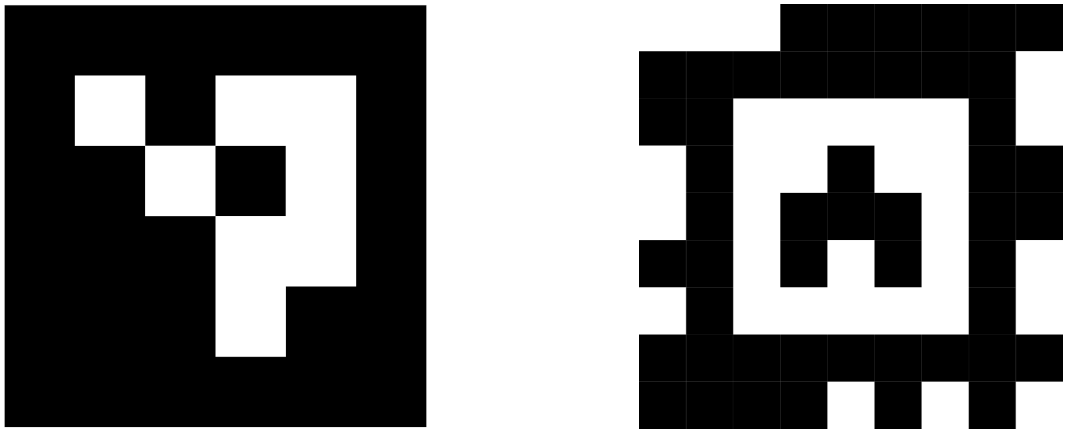
According to [59], an ArUco marker is a synthetic square marker composed of a wide black border and an inner binary matrix that makes up its identifier. From the [60] experiences, the ArUco tags, made available by the OpenCV library, was used to help estimate the pose of objects in images taken by a single camera from a single perspective. The OpenCV Aruco module provides the users with a simple interface to plug in the tag characteristics

they desire to detect the pose. It takes as input a 2D image containing the tag along with the camera parameters to estimate the pose of the tag in a 3D coordinate space relative to the camera. An alternative to ArUco provided by OpenCV is Charuco, which embeds ArUco-like tags in a chessboard. Charuco is more accurate than the ArUco grid, thanks to the included chessboard, and takes advantage of tag-based calibration, which supports occluded or partially visible calibration patterns [60].

2.3.2 AprilTag Markers

According to [61], the AprilTag algorithm [58] is an improvement of the ARToolkit pose estimation algorithm. Using the threshold value given by the user when acquiring the marker, the ARToolkit algorithm can only get a simple binary image and consequently can obtain information quickly, but showing poor stability in the complex environment with variable light intensity. In contrast, the AprilTag algorithm can get tag according to the gradient of the image, implying that it is more practical. Generally speaking, the AprilTag algorithm can take advantage of its detection mechanism to detect partially occluded markers, detect markers more easily, implement real-time error correction faster, complete simultaneous detection of multiple markers and carry out relative pose analysis better [61].

Figure 2.6 illustrates ArUco and Apriltag markers. As mentioned previously, the Apriltags are considered for this work, since their advantages include their high detection accuracy, low computational cost, and robustness to occlusion and lighting changes [61]. Furthermore, the availability of ROS packages for Apriltag detection and pose estimation makes their integration into robotic systems straightforward. These packages provide a simple interface for detecting Apriltags in real time, estimating their poses, and publishing the results as ROS messages. This integration can be used for a variety of robotic applications, such as object manipulation, autonomous navigation, and collaborative robotics.



(a) ArUco marker 4x4 ID 0. (b) AprilTag tag41h12 ID 15.

Figure 2.6: Samples of ArUco and Apriltag markers.

2.3.3 Work Scope

The fiducial markers are considered in the scope of this work as one of the methods to estimate the pose of the cage in relation to the robot's. Section 3.1.1 provides more details regarding the application of this method and how it should interact with the other components of the proposed system.

2.4 Supervised Learning

One of the techniques considered to estimate the pose of the AUV in relation to the cage is the supervised learning method, detailed in this Section.

Machine learning represents a large field in information technology, statistics, probability, artificial intelligence, psychology, neurobiology, and many other disciplines. With machine learning the problems can be solved simply by building a model that is a good representation of a selected dataset. Machine learning has become an advanced area from teaching computers to mimic the human brain. It has brought the field of statistics to a broad discipline that produces fundamental statistical, and computational theories of the learning processes [62].

Supervised learning is the most common machine learning technique in use today [62]. In the supervised learning paradigm, the goal is to infer a function $f : X \rightarrow Y$, the classifier, from a sample data or training set A_n composed of pairs of (input, output) points, x_i belonging to some feature set X, and $y_i \in Y$:

$$A_n = \{(x_1, y_1), \ldots, (x_n, y_n)\} \in (X \times Y)^n. \tag{2-1}$$

Typically $X \subset \mathbb{R}^d$, and $y_i \in \mathbb{R}$ for regression problems, and y_i is discrete for classification problems.

The second fundamental concept is the notion of error or loss to measure the agreement between the prediction $f(x)$ and the desired output y. A loss (or cost) function $L : Y \times Y \to \mathbb{R}^+$ is introduced to evaluate this error. The choice of the loss function $L(f(x), y)$ depends on the learning problem being solved. Loss functions are classified according to their regularity or singularity properties and according to their ability to produce convex or non-convex criteria for optimization.

In the case of pattern recognition, where $Y = \{-1, +1\}$, a common choice for L is the misclassification error:

$$L(f(x), y) = \frac{1}{2} |f(x) - y| . \tag{2-2}$$

This cost is singular and symmetric. Practical algorithmic considerations may bias the choice of L. For instance, singular functions may be selected for their ability to provide sparse solutions [63].

2.4.1 Convolutional Neural Networks

Artificial Neural Networks (ANNs) are a common choice for representing the learned mapping f. In [64], it was proposed the first mathematical model of neurons — the Multilayer Perceptron model. In [65] was proposed a single-layer perception model by adding learning ability to the MP model. However, single-layer perceptron networks cannot handle linear inseparable problems (such as XOR problems). In [66], a multilayer feedforward network was trained by the error backpropagation algorithm—backpropagation network, which addressed some problems that single-layer perceptron could not solve.

In the particular area of computer vision, the specific type of neural network used is called a Convolutional Neural Network. Nowadays, CNNs are used to construct the majority of computer vision algorithms.

In [67], the concept of CNN is defined. It consists of a feedforward neural network that can extract features from data with convolution structures. Different from the traditional feature extraction methods [68], CNN does not need to extract features manually. The architecture of CNN is inspired by visual perception [10]. A biological neuron corresponds to an artificial neuron; CNN kernels represent different receptors that can respond to various features; activation functions simulate the function that only neural electric signals exceeding a certain threshold can be transmitted to the next neuron. Loss functions and optimizers are something people invented to teach the whole

CNN system to learn what we expect. Figure 2.7 illustrates the main differences between CNN and Fully Connected (FC) layers.

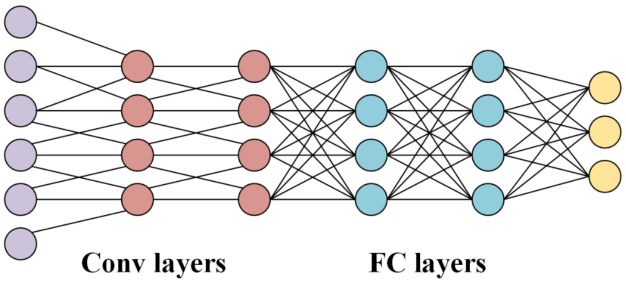


Figure 2.7: Diagram of CNN and Fully connected layers [67].

According to [63], compared with fully connected networks, CNN possesses many advantages:

1. **Local connections**: Each neuron is no longer connected to all neurons of the previous layer but only to a small number of neurons, effectively reducing parameters and speeding up convergence.

2. **Weight sharing**: a group of connections can share the same weights, which reduces parameters further.

3. **Downsampling dimension reduction**: a pooling layer harnesses the principle of image local correlation to downsample an image, which can reduce the amount of data while retaining useful information. It can also reduce the number of parameters by removing trivial features.

These three appealing characteristics make CNN one of the most representative algorithms in the deep learning field. To be specific, to build a CNN model, four components are typically needed. Convolution is a pivotal step for feature extraction. The outputs of convolution can be called feature maps [63]. When setting a convolution kernel with a specific size, it loses information on the border. Hence, padding is introduced to enlarge the input with zero value, which can adjust the size indirectly. Besides, to control the density of convolving, the stride is deployed. The larger the stride is, the lower the density is. After convolution, feature maps consist of many features prone to causing overfitting problems [69]. The procedure of 2-D (two-dimensional) a CNN is shown in Fig. 2.8.

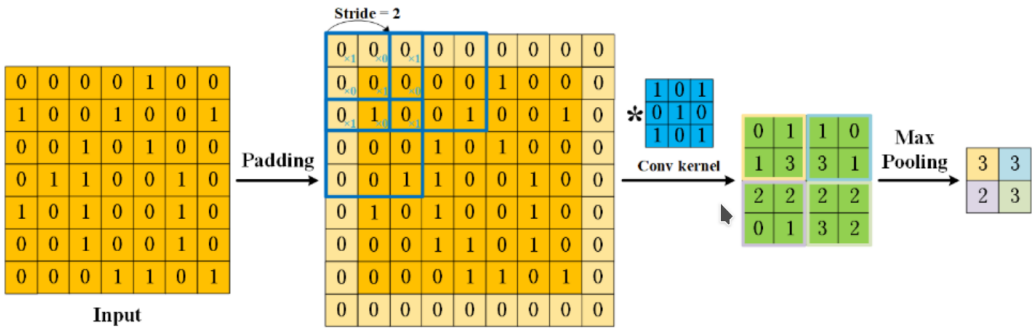
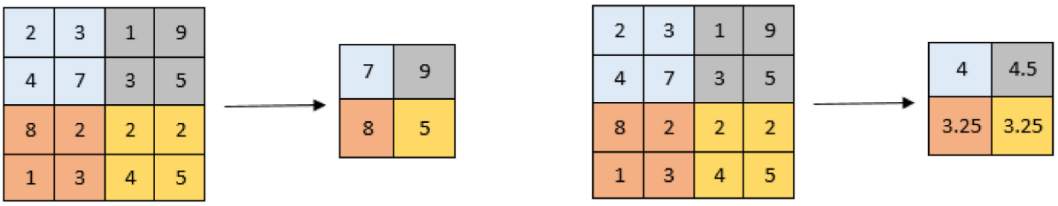


Figure 2.8: Procedure of a 2-D CNN [67].

According to [63], after obtaining the feature maps, it is necessary to add a pooling (sub-sampling) layer in CNN, next to a convolution layer. The job of the pooling layer is to shrink the convolved feature's spatial size. As a result of the dimensionality reduction, the computer power required to process the data is reduced. This also aids in the extraction of leading characteristics that are positional and rotational invariant, which preserves the model's practical training. Pooling reduces the training time while also preventing over-fitting.

The pooling layer calculates a summary statistic of the surrounding outputs to replace the network output at certain points. As a result, it aids in reducing the representation's spatial dimension, which reduces the amount of computation and weights required. The pooling procedure is carried out independently on each slice of the representation. The average of the rectangle neighborhood, the L2 norm of the rectangle neighborhood, and a weighted average depending on the distance from the central pixel are all pooling functions as shown in Figure 2.9. The most frequent method, however, is maximum pooling, which reports the neighborhood's most significant output.



(a) Max filter with 2x2 filter and stride 2.

(b) Average pool with 2x2 filter and stride 2.

Figure 2.9: Examples of maximum and average pooling layers [63].

[63] details a fundamental layer for all CNN networks, the activation function. It plays a vital role in CNN layers. The filter output is provided to another mathematical function called an activation function. ReLu, which stands for rectified linear unit, is the most common activation function used in

CNN feature extraction. The main motive behind using the activation function is to conclude the output of neural networks, such as “yes” or “no”. The activation function maps the output values between 1 to 1 or 0 to 1, etc. (it depends on the activation function). Non-linear activation functions are required to build complicated mappings between the network’s inputs and outputs, which are critical for learning and modeling complex data, including images, video, audio, and non-linear or high-dimensional data sets.

The FC is the last layer commonly considered for CNN architectures. A fully connected layer is nothing more than a feed-forward neural network as shown in Figure 2.10. Fully connected layers are found at the network’s very bottom layers. A fully connected layer receives input from the final pooling or convolutional layer’s output layer, which is flattened before being delivered as input. Flattening the output entails unrolling all values from the output that were obtained after the last pooling or convolutional layer into a vector (3D matrix).

Adding an FC layer is a simple technique to learn nonlinear combinations of high-level features represented by the convolutional layer’s output. In that space, the FC layer is learning a possibly nonlinear function.

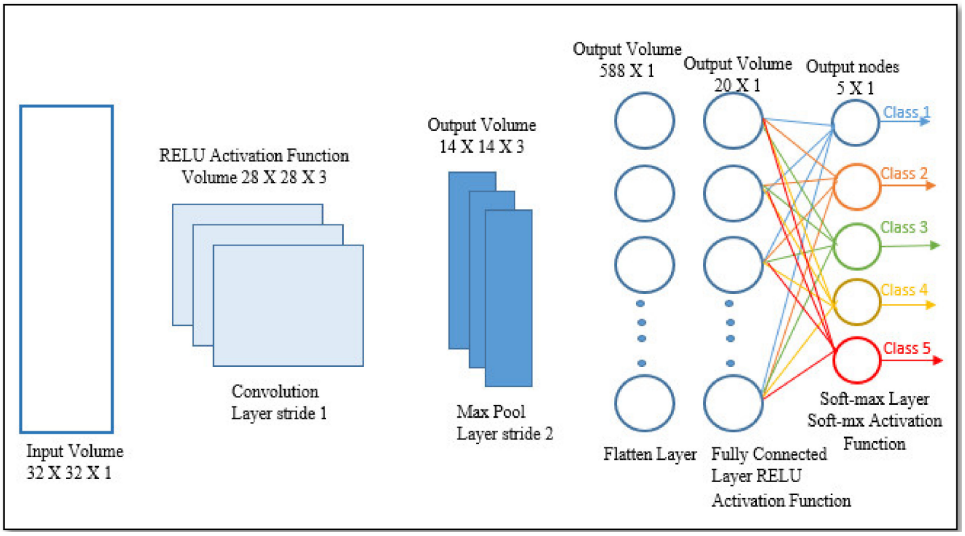


Figure 2.10: FC example presented in [67].

2.4.2 Work Scope

This study investigates the usage of CNNs for building different application scenarios, which were taken into account for the auto-docking task.

Therefore, a CNN model should be employed to estimate the cage pose relative to the AUV’s. The VGG-16 [70] architecture was selected due to its flexibility in taking an image as input and outputting the seven values

$(x, y, z, q_x, q_y, q_z, q_w)$ corresponding to the cage's pose (translation and rotation).

Further details regarding the necessary resources for implementing this method and its proper usage are presented in subsequent chapters.

2.5 Reinforcement Learning

Reinforcement Learning (RL) is a Machine Learning paradigm capable of optimizing sequential decisions [71]. RL is interesting because it mimics how humans learn. Humans are instinctively capable of learning strategies that help people master complex tasks like riding a bike or taking a mathematics exam. RL attempts to copy this process by interacting with the environment to learn strategies.

The concept of *learning by reinforcement* combines two tasks. The first is exploring new situations. The second is using that experience to make better decisions. Given time, this results in a plan to achieve a task. For example, a child learns to walk, after standing up, by leaning forward and falling into the arms of a loving parent. But this is only after many hours of hand-holding, wobbles, and falls. Eventually, the leg muscles of the baby operate in unison using a multistep strategy that tells them what to move and when. You can not teach every plan that the child will ever need, so instead, life provides a framework that allows them a way to learn.

From the perspective presented in [72], in a nutshell, RL is based on rewards and punishments, with the following relevant aspects:

- It differs from normal Machine Learning as it does not look at training datasets.
- Interaction happens not with data but with environments that depict real-world scenarios.
- As RL is based on environments, many parameters come into play. It takes lots of information to learn and act accordingly.
- Environments in RL are real-world scenarios that might be 2D or 3D simulated worlds or game-based scenarios.
- RL is broader in a sense because the environments can be large in scale, and there might be a lot of factors associated with them.
- The optimization criterion (rewards) in RL are obtained from the environment.

RL consists of an agent that learns by itself via interaction with the environment. Figure 2.11 introduces the four elementary components of RL.

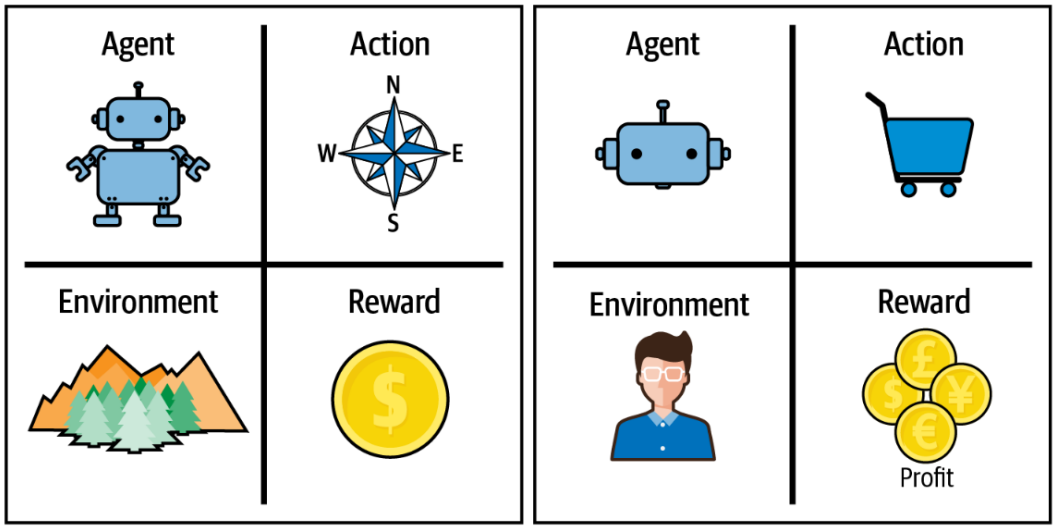


Figure 2.11: A plot of the four components required for RL: an agent to present actions to an environment for the greatest reward. The example on the left shows a robot that intends to move through a maze to collect a coin. Example on the right shows an e-commerce application that automatically adds products to users of baskets to maximize profit [71].

The trial and error learning behavior of RL is dangerous in the real world, where accidents are to be avoided at all costs. For this reason, RL is most often carried out in simulators. Learning is based on reward instead of labels, where the goal is to maximize reward accumulated over time. The reward can be sparsely distributed, for example, when achieving a goal such as successfully completing a turn-right maneuver. A reward can also be negative, i.e., a penalty, for instance, when the car departs from the lane. Hence, the reward is different from labels and not tied to the local action but to the overall achievement [72].

Beyond the agent and the environment, one can identify four main subelements of a reinforcement learning system: a policy, a reward signal, a value function, and, optionally, an environment model [71].

As defined by [73], a policy defines the learning agent's behavior at a given time. Roughly speaking, a policy is a mapping from perceived states of the environment to actions to be taken when in those states. It corresponds to what in psychology would be called a set of stimulus–response rules or associations. In some cases, the policy may be a simple function or lookup table, whereas in others, it may involve extensive computation, such as a search process. The policy is the core of a reinforcement learning agent because it alone is sufficient

to determine the behavior. In general, policies may be stochastic, specifying probabilities for each action.

Moreover, a reward signal defines the goal of a reinforcement learning problem. On each time step, the environment sends a single number called the reward to the reinforcement learning agent. The agent's sole objective is to maximize the total reward it receives over the long run. The reward signal thus defines what the good and bad events for the agent are. In a biological system, we might think of rewards as analogous to the experiences of pleasure or pain. They are the immediate and defining features of the problem faced by the agent. The reward signal is the primary basis for altering the policy; if an action selected by the policy is followed by a low reward, then the policy may be changed to select some other action in that situation. In general, reward signals may be stochastic functions of the state of the environment and the actions taken.

Whereas the reward signal indicates what is good in an immediate sense, a value function specifies what is good in the long run. Roughly speaking, the value of a state is the total amount of reward an agent can expect to accumulate over the future, starting from that state. Whereas rewards determine the immediate, intrinsic desirability of environmental states, values indicate the long-term desirability of states after taking into account the states that are likely to follow and the rewards available in those states. For example, a state might always yield a low immediate reward but still have a high value because it is regularly followed by other states that yield high rewards. Or the reverse could be true. To make a human analogy, rewards are somewhat like pleasure (if high) and pain (if low), whereas values correspond to a more refined and farsighted judgment of how pleased or displeased we are that our environment is in a particular state.

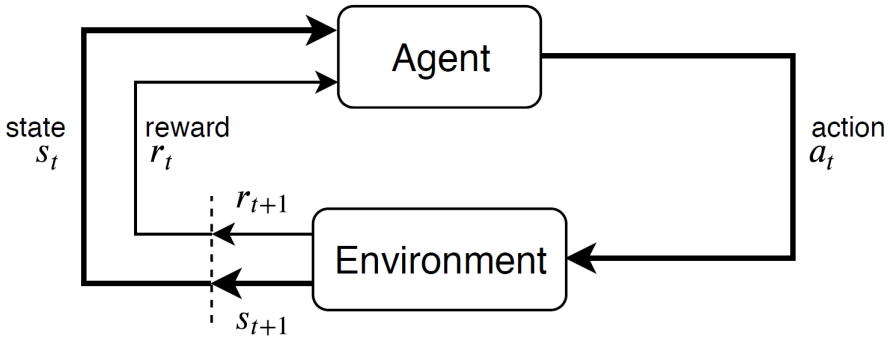


Figure 2.12: Reinforcement learning setup where an agent learns by interaction with an environment. The agent computes an action based on the current state S_t. The environment executes the action and moves to a new state S_{t+1} and yields a reward r_{t+1}. The agent optimizes its policy, i.e., how to choose actions, by seeking to maximize reward. Adapted from [74].

Following the background provided by [72], in the common RL setup, depicted in Figure 2.12, an agent interacts with an environment at discrete time steps t by receiving the state S_t, on which basis it selects an action a_t as a function of policy π with probability $\pi(a_t|S_t)$ and sends it to the environment where a_t is executed and the next state S_{t+1} is reached with associated reward r_{t+1}. Both, state S_{t+1} and reward r_{t+1}, are returned to the agent which allows the process to start over. The discounted return corresponds to the sum of rewards that is gained taking infinite time steps and is to be maximized by the agent. It writes:

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}. \tag{2-3}$$

Where $\gamma \in [0, 1[$ is the discount factor. Hypothetically, we could set $\gamma = 1$, so that there would be no decay and future rewards are as important as the current reward. However, this creates a mathematical problem, because the sum could go to infinity for positive rewards. Setting $\gamma < 1$ makes the sum of rewards finite and helps the convergence of reinforcement algorithms. A typical value is $\gamma = 0.99$, which decays the rewards over time, making an agent slightly short-sighted in desiring to collect rewards earlier, rather than later.

2.5.1 Main RL Techniques

Model-free RL algorithms can be divided into two different families:

– **Policy-based RL** methods directly try to estimate the policy $\pi(a|s)$. Thus, for each state s, the policy $\pi(a|s)$ tells us which action we should

take. In practice, policy-based RL methods are often coupled with an actor-critic approach to stabilize training.

– **Value-based RL** algorithms or **Q-learning**, seek to approximate the optimal action-value function

$$Q^*(s,a) \doteq max_\pi \mathbb{E}_\pi[R_t|s_0 = s, a_0 = a]. \tag{2-4}$$

which gives the expected return for taking action a in current state s and acting according to the optimal policy thereafter. Value-based methods do not estimate the policy $\pi(a|s)$. Instead, they choose the optimal action $a^*(s)$ that maximizes the action-value function.

$$a^*(s) = argmax_a Q^*(s,a). \tag{2-5}$$

The actions in RL can be continuous or discrete. In the discrete case, action prediction amounts to a classification problem. The set of available actions is referred to as action space. Pure value-based RL techniques are limited to discrete actions, while policy-based approaches can handle continuous actions.

One of the most efficient and stable policy-based RL algorithms is TD3. It builds upon two other approaches, the value-based Deep Q-Network (DQN), and Deep Deterministic Policy Gradient (DDPG). However, it is essential to note that there is a wide range of RL methods available [72], and the best solution for a particular challenge will depend on its unique requirements and concerns.

2.5.2 DQN

DQN, which was introduced for the first time in [75], is the first scalable reinforcement learning algorithm that combines Q-learning with deep neural networks. To overcome stability issues, DQN adopts two novel techniques that turned out to be essential for the balance of the algorithm: a replay buffer to get over the data correlation drawback and a separate target network to get over the non-stationarity problem [73]. DQN uses the Q-learning target

$$Q(s_i, a_i; \theta) \leftarrow r_i + \gamma \max_{a'} Q(s'_i, a'; \theta^{\mathrm{t}}), \tag{2-6}$$

where s_i and s_i are the current state and action, r_i is the reward, and s'_i is the next state, all sampled from the replay buffer, and θ and θ^{t} are the online and target network weights, respectively.

The replay buffer ideally contains all the transitions that have taken place during the agent's lifetime. When doing Stochastic Gradient Descent,

a random mini-batch is gathered from the experienced replay and used in the optimization procedure. Since the replay memory buffer holds varied experiences, the mini-batch that is sampled from it tends to be diverse enough to provide independent samples.

The moving target problem is due to continuously updating the network during training and modifying the target values. Nevertheless, the neural network has to update itself in order to provide the best possible state-action values. The solution that's employed in DQNs is to use two neural networks. One is called the online network, which is constantly updated. In contrast, the other is called the target network, which is updated only every N iteration (with N usually being between 1,000 and 10,000). The online network is used to interact with the environment while the target network is used to predict the target values. In this way, for N iterations, the target values that are produced by the target network remain fixed, preventing the propagation of instabilities and decreasing the risk of divergence. A potential disadvantage is that the target network is an old version of the online network. Nonetheless, in practice, the advantages greatly outweigh the disadvantages, and the algorithm's stability improves significantly [72]. Figure 2.13 shows the pseudo-code for the DQN algorithm.

function DQL(γ, β)
  $\theta \leftarrow 0$, $\theta^t \leftarrow 0$
  while true **do**
    $s \leftarrow$ Start
    Sample a using policy derived from $\hat{Q}(s, a; \theta)$
    repeat
      Take action a, observe new state s', reward r
      Add $(s, a) \rightarrow (r, s')$ to $\mathcal{D}$
      Sample a' using policy derived from $\hat{Q}(s, a; \theta)$
      Choose random mini-batch $\mathcal{B} \subset \mathcal{D}$
      for each sample $b_i : (s_i, a_i) \rightarrow (r_i, s_i') \in \mathcal{B}$ **do**
        $q_i = r_i + \gamma \max_{a'} \hat{Q}(s_i', a'; \theta^t)$
      Train $\hat{Q}(\theta)$ on all samples $(s_i, a_i) \in \mathcal{B} \rightarrow q_i$
      $\theta^t \leftarrow \theta^t + \beta(\theta - \theta^t)$
      $s \leftarrow s'$, $a \leftarrow a'$
    until episode ends

Figure 2.13: DQN pseudo algorithm.

2.5.3 DDPG

DDPG was the first policy-based RL algorithm to employ deep neural networks. It inherits characteristics of a policy gradient method [72] and an Actor-Critic (AC) method [72], which has its structure depicted in Figure 2.14. It uses two Deep-Q Networks to represent the policies: one for the actor part and one for the critic part. Thus, the policies can represent problems with high complexity. One of the big advantages of DDPG is dealing with high dimensional continuous state and action space, which is necessary to apply to control problems such as the AUV auto-docking challenge. To have this feature, the policy of actor part, a CNN, receives a state from an environment and generates a continuous action.

As mentioned previously, the DDPG algorithm is based on the Actor-Critic paradigm. According to [74], the network structure of the AC framework includes a policy network and an evaluation network. The policy network is called an Actor-network, and the evaluation network is called a Critic network. The Actor network is used to select actions corresponding to the DDPG, and the Critic network evaluates the merits and demerits of the selected actions by calculating the value function. The Actor-network and the Critic network are two separate networks that share the state information. The Actor network uses state information to generate actions, while the environment feeds back the resulting actions and outputs reward. The Critic network uses state and reward to estimate the value of the current action and constantly adjusts its own value function.

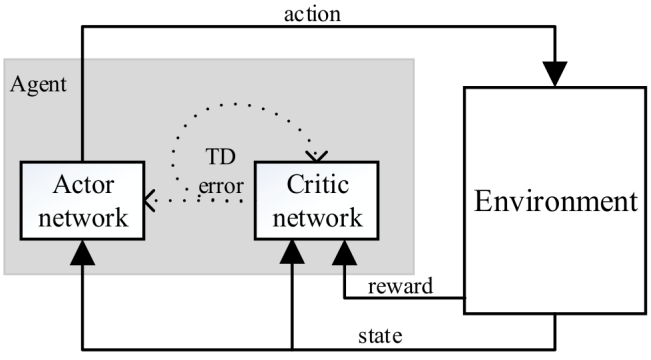


Figure 2.14: Actor-Critic algorithm structure [75].

In DDPG, The critic Q is learned with standard DQN, while the actor π is updated according to the deterministic policy gradient

$$\nabla_a Q(s, a; \theta_{\mathrm{Q}})|_{s=s_i, a=\pi(s_i;\theta_\pi)} \nabla_{\theta\pi} \pi(s; \theta_\pi)|_{s=s_i}, \tag{2-7}$$

averaged over a minibatch with samples s_i drawn from the experience replay buffer. The DPG moves the policy parameters $\theta_p i$ towards actions that have higher Q values according to the critic.

Figure 2.15 gathers the pseudo algorithm steps to implement the DDPG paradigm.

DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    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 sampled policy gradient:

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

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

Figure 2.15: DDPG pseudo algorithm [72].

2.5.4 TD3

According to [76], the Twin-Delayed Deep Deterministic Policy Gradient is an off-policy model that recently achieved breakthroughs in Artificial Intelligence state-of-the-art models with continuous high-dimensional spaces. TD3 was built on the DDPG to increase stability and performance with consideration of function approximation error [71]. The uniqueness of the TD3 algorithm is in the combination of three powerful Deep Reinforcement Learning techniques: continuous Double Deep Q-Learning [72], Policy Gradient [71], and Actor-Critic [72].

The main advantages of TD3 over the pure DDPG are detailed in [77]:

1. A reduced overestimation of a dual Critic network. Since noise (θ) appears in the sample value (y), the target value E_θ of Q network in the real case of the learning process is:

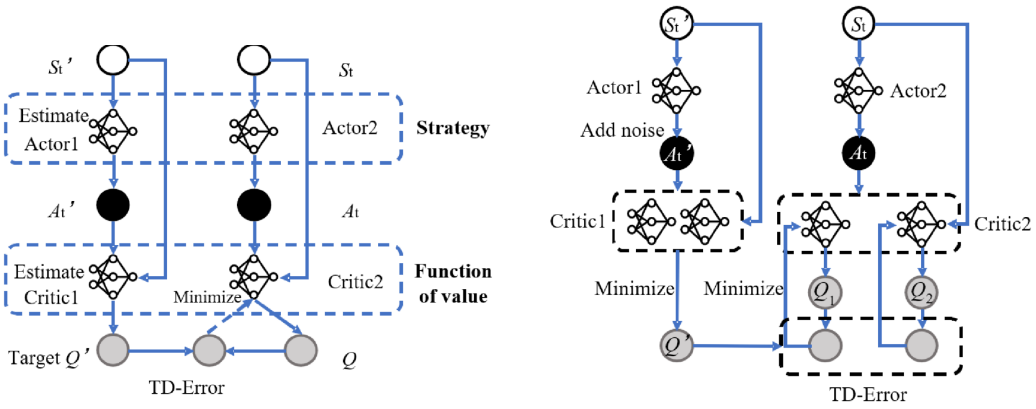
$$y = r + \gamma \max_{a'} Q(s', a'). \tag{2-8}$$

$$\mathbb{E}_\theta \left[\max_{a'} Q(s', a') + \theta \right] \geq \max_{a'} Q(s', a'). \tag{2-9}$$

After updating the Q function many times, errors will accumulate, leading to a number of bad states being assigned with higher values, and to a large deviation. A double Q-value network reduces estimation errors because it decouples the action and updates operations. Using two independent Critics (sharing experience pool) in the TD3 algorithm, we can take the minimum value between the two Critics to eliminate the phenomenon of overestimation and to update the target value. However, although this may lead to an underestimated value for the strategy, this is preferable to overestimating it, as cumulative overestimation will make the strategy ineffective.

2. Smooth regularization of the target strategy. Each step of the TD update produces a small error, which is more noticeable for approximate estimates. After many updates, a large number of errors will accumulate, eventually leading to an inaccurate Q value. When the Actor and Critic are trained simultaneously, there may be a situation where training is unstable or divergent. The method includes two aspects. First, a regularization of the parameters (add noise). Secondly, it updates the target network every d times after updating Critic. The target network calculates the update objectives of Critic to improve the value function convergence, wherein the value function is updated at a higher frequency, and the strategy is updated at a lower frequency.

Figure 2.16 illustrates the high-level architecture of both methods.



(a) Structure of DDPG algorithm [77]. (b) Structure of TD3 algorithm [77].

Figure 2.16: High-level structures of TD3 and DDPG algorithms.

According to [76], the steps to implement TD3 are similar to those in DDPG. Figure 2.17 depicts the algorithm considered to implement TD3.

TD3

Initialize critic networks Q_{θ_1} , Q_{θ_2} , and actor (policy) 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 replay buffer $\mathcal{B}$
for $t = 1$ **to** T **do**
  Select an action with exploration noise $a \sim \pi(s) + \epsilon$, $\epsilon \sim \mathcal{N}(0,\sigma)$ and observe reward r and new state s'

  Store transition tuple (s,a,r,s') in $\mathcal{B}$
  Sample mini-bach of N_T transitions (s,a,r,s') from $\mathcal{B}$
  $\tilde{a} \leftarrow \pi_{\phi'}(s) + \epsilon$, $\epsilon \sim \text{clip}(\mathcal{N}(0,\sigma), -c, c)$
  $y \leftarrow r + \gamma \min_{i=1,2} Q_{\theta_i'}(s', \tilde{a})$
  Update critics $\theta_i \leftarrow \min_{\theta_i} N_T^{-1} \sum (y - Q_{\theta_i}(s,a))^2$
  if t **mod** d **then**
    Update ϕ by the deterministic policy gradient: $\nabla_\phi J(\phi) = N_T^{-1} \sum \nabla_a Q_{\theta_1}(s,a)\Big|_{a=\pi_\phi(s)} \nabla_\phi \pi_\phi(s)$
    Update target networks: $\theta_i' \leftarrow \tau\theta_i + (1-\tau)\theta_i'$, $\phi_i' \leftarrow \tau\phi + (1-\tau)\phi'$
  end if
end for

Figure 2.17: TD3 pseudo algorithm [76].

Initially, two critic networks have to be passed, denoted by the parameters θ_1 and θ_2. Most implementations expect you to pass in two independent DL networks and typically these have exactly the same architecture. The original algorithm clips the action noise to keep the result close to the original action. The intent is to prevent the agent from choosing invalid actions. Most implementations set this value to match the environment's action space. It chooses a noise function that matches the action space and smoothing goals, then reuses the standard deterministic action function from DDPG. The predicted value is the minimum value from both critics from the target networks. In other words, it picks the lowest action-value estimation; this is more likely to be correct due to the overestimation bias. Afterward, both critics are updated using this prediction of the discounted reward. This helps to restrain the network that is doing the overestimation.

The inner loop occurs once every d iteration; this delays the policy update. All other lines are the same as DDPG.

2.5.5 Work Scope

The RL agent proposed for this work should be applied to the controlling task.

The definition of the method that should be implemented is detailed in Chapter 3. Based on the context presented in the previous Section, it is essential to carefully design the RL algorithm and the agent's architecture to ensure robustness and efficiency because the RL will be used in an underwater scenario, where environmental challenges like low visibility and high

pressure present major obstacles. Furthermore, non-stationary dynamics and sparse reward signals are common in underwater environments, necessitating quick adaptation and effective learning from limited feedback. Choose a RL algorithm that can handle non-stationary environments and has a sample-efficient learning mechanism.

3
Methodology

In this chapter, the methodology adopted for the progression and implementation of this work is detailed. The approach is experimental, based on the application of exploratory and deductive techniques for the construction and interpretation of results.

This work aims to propose a solution for performing the auto-docking task involving machine learning and RL paradigms. To achieve this objective, it is explored various scenarios to evaluate their performance. By doing so, it is expected to provide a comprehensive understanding of the best approaches to implementing auto-docking, under the scope detail so far, thereby contributing to the advancement of this field.

In summary, this Chapter follows the workflow considered to detail the previous Chapter. However, only the block that concerns the experiments, named as *Visual Servoing*, is detailed, since the methods proposed to handle the cage pose estimation and controlling are presented. Figure 3.1 illustrates the aimed workflow.

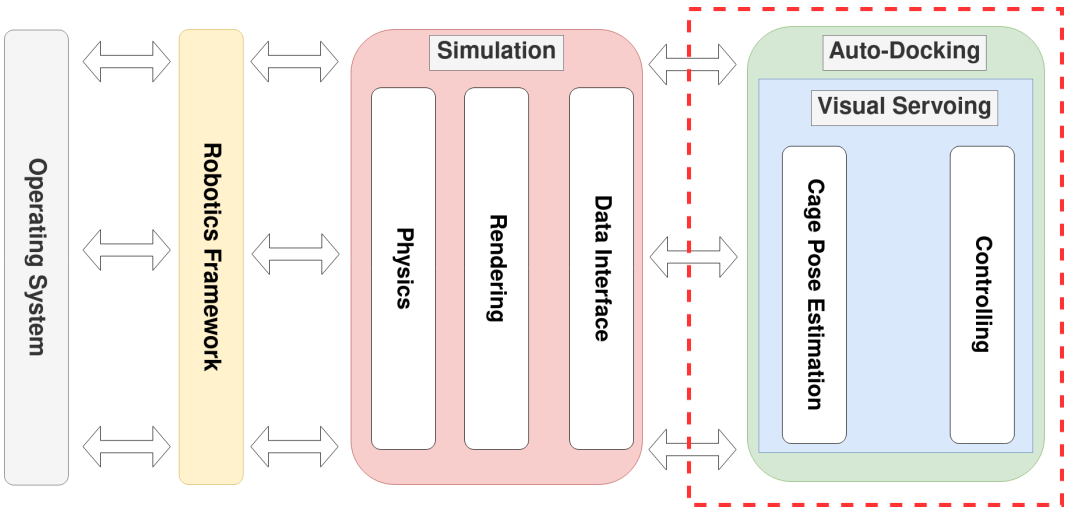


Figure 3.1: Workflow adopted for this work. The Visual Servoing scope is highlighted since the methods chosen to carry out this task are covered in this chapter, following the depicted order.

3.1 Proposed Methods

The methods proposed in this Section for performing the auto-docking task are based on the steps required to accomplish the task: from receiving an image to delivering the linear and angular velocities. From a high-level perspective, it comprises the AUV's pose estimation in relation to the cage and controlling it to achieve the corresponding goal.

In order to estimate the AUV's pose in relation to the cage, two approaches are considered in this work: fiducial-based (detailed in Section 2.3) and supervised learning-based (detailed in Section 2.4).

Regarding controlling, two approaches are considered in this work: PID-based and Reinforcement Learning-based (detailed in Section 2.5).

The following Sections detail each of the proposed methods.

3.1.1 Pose Estimation: Fiducial-Based

As defined in [61], visual fiducials are artificial landmarks designed to be easy to recognize and distinguish from one another, which can be detected and retrieved from images using a processing algorithm (in a similar manner with bar-codes or QR-codes. As detailed in Section 2.3, different types of fiducial markers are available providing their own features to be recognized by specific algorithms, but all of them are based on the projective geometry concepts required to estimate the tag's pose.

In this work, AprilTags [58] is considered as the target family of tags, since advantages include their high detection accuracy, low computational cost, and robustness to occlusion and lighting changes (Section 2.3 provides more details towards the reasons of this definition). For implementing the alignment method, the *apriltag_ros*[1] ROS open-source package was chosen, given that a complete software library with tools for generating and detecting the markers is available [61]. In Figure 3.2 is shown a fiducial marker and the frames and transformation matrix applied to obtain its pose in the space.

[1]https://github.com/AprilRobotics/apriltag_ros

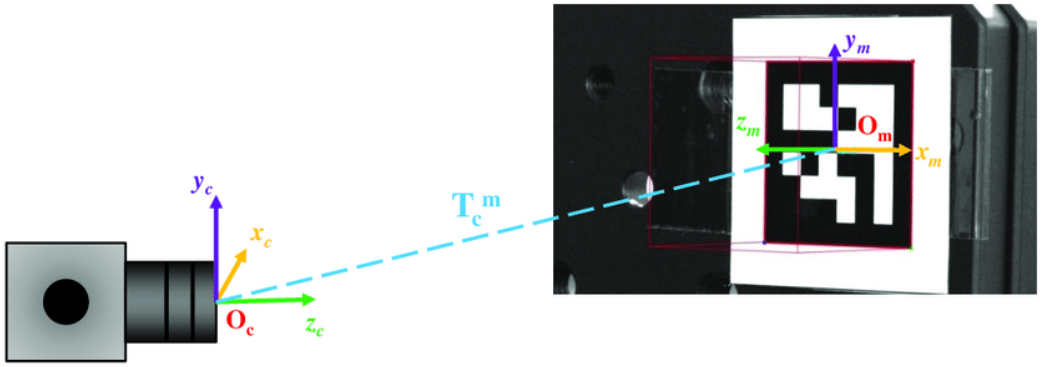


Figure 3.2: The estimation of the position and orientation of a fiducial marker. This is achieved using the pinhole camera model in which the extrinsic parameters are determined knowing the intrinsic ones [78].

In this work scope, the center of the cage has known orientations and translations relative to each tag. Then the docking station target can be automatically pre-aligned with the AUV without requiring additional steps as illustrated in Figure 3.13. In order to clarify the interactions, Figure 3.3 illustrates the AprilTag pose estimation and the transformation interactions between the cage, the tags and the robot in the proposed simulated environment.

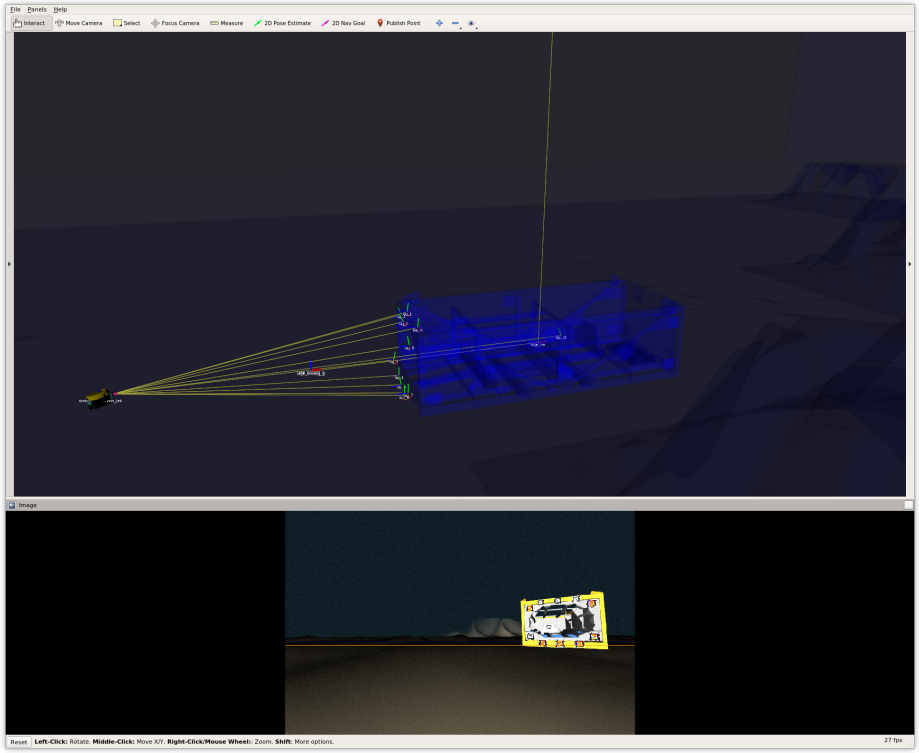


Figure 3.3: Transformations represented in RViz tool. The interaction between the frames illustrates how the pose estimation is applied using fiducials.

The tags detection and pose estimation workflows are based on the capa-

bility of the AUV's camera to capture the tags having a clear view of the tag's corners and their respective middle content. Once captured, the *apriltag_ros* algorithm is triggered, and processing starts. As illustrated in Figure 3.4, the output of the ROS package is the transformations between the AUV's camera and the tags (/tf), the detect tags numbers (/tag_detections) and the image containing the detected tags with bounding boxes (/tag_detections_image).

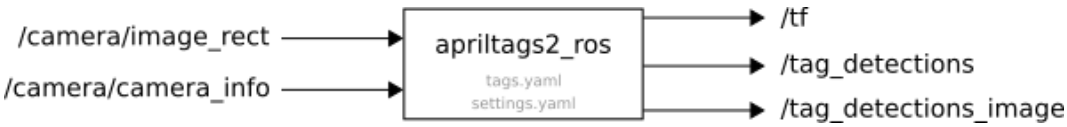


Figure 3.4: *apriltag_ros* ROS software components.

3.1.2 Pose Estimation: Supervised Learning-Based

Performing robotic learning in a simulator could accelerate the impact of machine learning on robotics by allowing faster, more scalable, and lower-cost data collection than is possible with physical robots. This is achieved because of the knowledge of the environment provided by the simulation. This information can be used to acquire respective ground-truth data required to train the supervised learning algorithms properly. In that context, the usage of Convolutional Neural Networks, detailed in Section 2.4.1, can be leveraged since the simulators provide one more crucial feature, the acquisition of a considerable amount of data under different environment configurations.

In this work, the usage of CNN is adopted for estimating the AUV's pose in relation to the docking cage. As detailed in Section 2.4.1, there are different types of architectures composing the CNN taxonomy. For this work, the Spatial Exploitation based CNNs architecture was chosen. According to [67], this type of architecture is designed to be invariant to translation, meaning they can recognize features regardless of their location in the image, a crucial aspect in pose estimation tasks where the position of the subject can vary.

As a starting point, the dataset was considered one of the primary resources for analyzing the feasibility of this challenge. Initially, a software tool was implemented to save a pair of data corresponding to the image acquired, and the associated cage pose at the same image's timestamp. Each pair of image was treated as a sample of the dataset. Figure 3.5 illustrates the workflow to acquire the dataset.

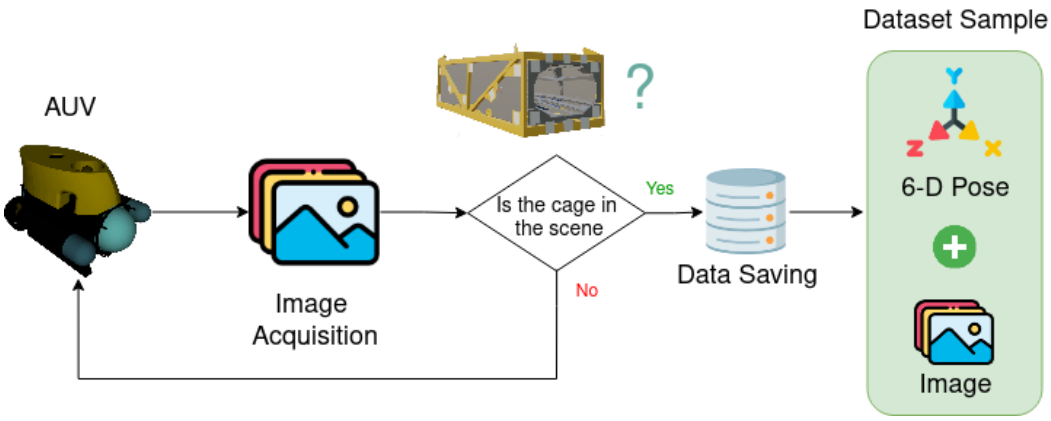


Figure 3.5: Dataset acquisition workflow.

As illustrated in Figure 3.5, the only concern was related to the presence of the cage in the acquired scene. It was done by respawning the robot in random positions in a certain range. The range was set in order to guarantee that the cage would be at the scene. In total, 5 thousand of data samples were acquired. The proportion of 80%, 10%, 10% was considered for training, validation, and testing, respectively. As noticed, manual labeling was not required.

An additional point to consider in this work, commonly associated with the usage of CNNs is the concept of transfer learning. As reported in [69], this approach allows for the utilization of pre-trained models that have been trained on large datasets to extract useful features from new data. This can significantly reduce the amount of computational resources and time needed to train a model from the beginning. Pre-trained models are usually trained on varied datasets and can therefore extract more generalized features, improving the model's performance on unseen data.

Therefore, Figure 3.6 illustrates the solution adopted in this work to estimate the AUV's pose in relation to the cage. This model is an adaptation of the Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World, proposed by [79]. It is based on the original VGG-16 backbone architecture, and the weights are initialized with the ImageNet dataset [69]. The head of the network has been replaced with a 3D position predictor that outputs (x, y, z) coordinates and an orientation predictor that outputs a quaternion (q_x, q_y, q_z, q_w).

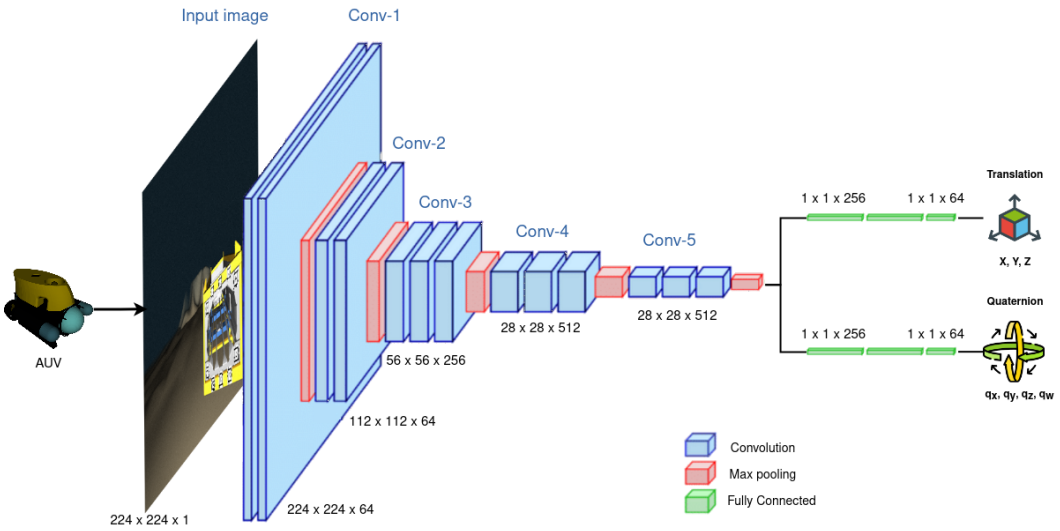


Figure 3.6: Adapted VGG-16 architecture considered for this work. The last layers were removed in order to adapt it to perform data regression instead of data classification, as originally proposed in [79].

The hyperparameters used in the experiments involving CNN applications were kept the same as those proposed in [79]. As the primary focus of this study was not to modify the CNN hyperparameters and the original work provided satisfactory results, no changes were made. The hyperparameters considered are detailed in Table 3.1.

Table 3.1: CNN hyperparameters considered for this work.

Parameter	**Value**
Number of GPUs used	4
Training dataset size	4500
Batch training size	20
Accumulation steps	10
Epochs	120
Beta loss	10
Sample size (Train)	0
Validation dataset size	500
Batch validation Size	30
Evaluation frequency	4
Sample size (Validation)	0
Test dataset size	200
Batch test size	30
Sample size (Test)	0
Image Scale	224
Adam optimizer (lr)	0.0001
Adam optimizer ($beta_1$)	0.9
Adam optimizer ($beta_2$)	0.999
Translation accuracy threshold	0.5

As detailed in 2.4.1, the main purpose of CNNs is to minimize the loss function, which is used to measure the difference between the predicted values and the true targets. This concept was applied to both the translation and orientation components of the network, using the Mean Squared Error (MSE) as loss function. The MSE calculates the average of the squared discrepancies between the predicted outputs ($\hat{y}_i$) and the actual targets (y_i).

The accuracy associated with the estimated translation and orientation was individually computed with the objective of providing an intuitive metric to evaluate global performance during the training and validation steps.

3.1.3 Controller: PID-Based

A PID controller, encompassing Proportional, Integral, and Derivative terms, serves as the foundation for managing robotic movements in visual servoing, primarily driven by visual feedback [53].

As detailed in Section 2.2.1, the essence of visual servoing revolves around visual error, which is discerned from the disparity between the current and desired camera perspectives. This visual error seamlessly transitions into the

role of the principal input for the PID controller. In response, the controller orchestrates corrective maneuvers to minimize the prevalent error. Herein, the proportional segment modulates the robot's trajectory in proportion to the visual error, the integral component grapples with the accumulation of historical errors, while the derivative facet is forward-looking, making adjustments based on anticipated errors. Together, they lead to a robust alignment of the camera's perspective with the aspired position or the adept tracking of a dynamic entity. Figure 3.7 illustrates the block diagram adopted as proposed solution for the PID-based pose controller.

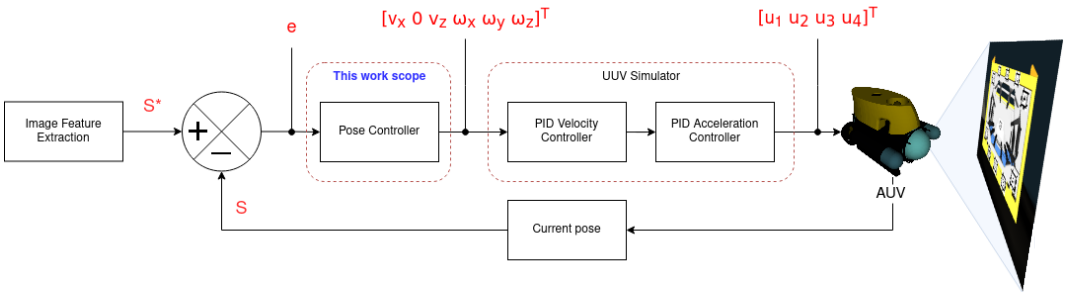


Figure 3.7: Block diagram representation of the AUV's control system utilizing a PID-based pose controller.

As illustrated in Figure 3.7, the PID-based controller designed for this work receives the pose estimation (calculated by its respective components) as a setpoint (S^*), which consists of the visual error of the current robot pose for the desired one (both poses are based on visual feedback). This estimation is compared to the current pose computed by the robot (S), using on-board sensors, and serves as input error (e) for the pose controller implemented within this scope of work. Then, the computed velocities outputs ($\begin{bmatrix} v_x & v_y & v_z & \omega_x & \omega_y & \omega_z \end{bmatrix}^\top$) are forwarded to the cascade PID controllers implemented by the *uuv_simulator* ROS package (detailed in Section 2.1.3), which outputs the control signals ($\begin{bmatrix} u_1 & u_2 & u_3 & u_4 \end{bmatrix}^\top$) to the thrusters, adhering to the architecture depicted in Figure 4.10.

The velocities, linear and angular, considering the time domain[2], are calculated as per:

– **Linear Velocity Control (V)**: Given $e(t)$ as the error in pose estimation at time t, the linear velocity control is formulated as:

$$V(t) = K_p \cdot e(t) + K_i \cdot \int_0^t e(\tau)d\tau + K_d \cdot \frac{de(t)}{dt}. \tag{3-1}$$

[2]The PID controller equations presented are primarily formulated in the continuous time domain. This is a common practice for analytical clarity and initial design considerations. However, the implemented controllers operate in discrete time. The conversion from continuous to discrete PID control is well-established and not particularly intricate.

where:

- K_p is the proportional gain,
- K_i is the integral gain, and
- K_d is the derivative gain.

– **Angular Velocity Control (ω)**: Let $\theta(t)$ be the angular error at time t. The angular velocity control is:

$$\omega(t) = K_{p\theta} \cdot \theta(t) + K_{i\theta} \cdot \int_0^t \theta(\tau)d\tau + K_{d\theta} \cdot \frac{d\theta(t)}{dt}, \tag{3-2}$$

where:

- $K_{p\theta}$ is the proportional gain for angular error,
- $K_{i\theta}$ is the integral gain for angular error, and
- $K_{d\theta}$ is the derivative gain for angular error.

It ensures that the pose controller computes both linear and angular velocities based on the difference between the desired pose and the current pose. Adjusting the PID gains will influence how quickly and accurately the system responds to pose errors. Table 3.2 gathers the gains considered for this controller.

Table 3.2: Empirically Determined PID Gains.

Component	**Proportional (P)**	**Integral (I)**	**Derivative (D)**
Linear	0.1	0.05	0.1
Angular	3.5	0.02	3.5

The PID gains provided in Table 3.2 were determined empirically due to the unavailability of specific knowledge about the robot's dynamics. An empirical tuning method was utilized, enabling performance adjustments, particularly when the robot's analytical model isn't available or is too intricate to formulate.

3.1.4 Controller: RL-Based

In the rapidly evolving landscape of RL, algorithms designed for continuous action spaces have emerged as essential tools for solving a wide array of complex tasks [71]. Among these algorithms, Proximal Policy Optimization (PPO) [72], Soft Actor-Critic (SAC) [72], and TD3 (detailed in Section 2.5.4) have gained significant attention for their robustness, efficiency, and scalability [71]. These algorithms are particularly well-suited for applications in 3D

environments, where the state and action spaces are naturally continuous and high-dimensional [72].

Consequently, for the execution of the auto-docking experiments within the RL paradigm, PPO, SAC, and TD3 will be evaluated as candidate algorithms to this work. The selected algorithm will operate within the control layer, aligning with the specifications detailed in Chapter 2, establishing the control domain as the appropriate context for the RL application since the intricacies associated with image processing present their own set of challenges and are considered beyond of the scope of this work.

In this section, we detail the methodology employed for evaluating the candidate algorithms - PPO, SAC, and TD3 — and the architecture for the reinforcement learning environment tailored for training. The RL training environment's main aspects that shall be covered are:

- **Reward Formulation**: The criteria for evaluating actions and steering the learning process.
- **Action Space Definition**: The range and types of actions the agent can take within the environment.
- **State Update Mechanics**: The rules governing how the environment and agent states evolve as actions are taken.
- **Observation Collection Protocols**: Guidelines for how sensory data and other observations are collected and processed.

Figure 3.8 illustrates the software components used in this work to build the target RL training environment.

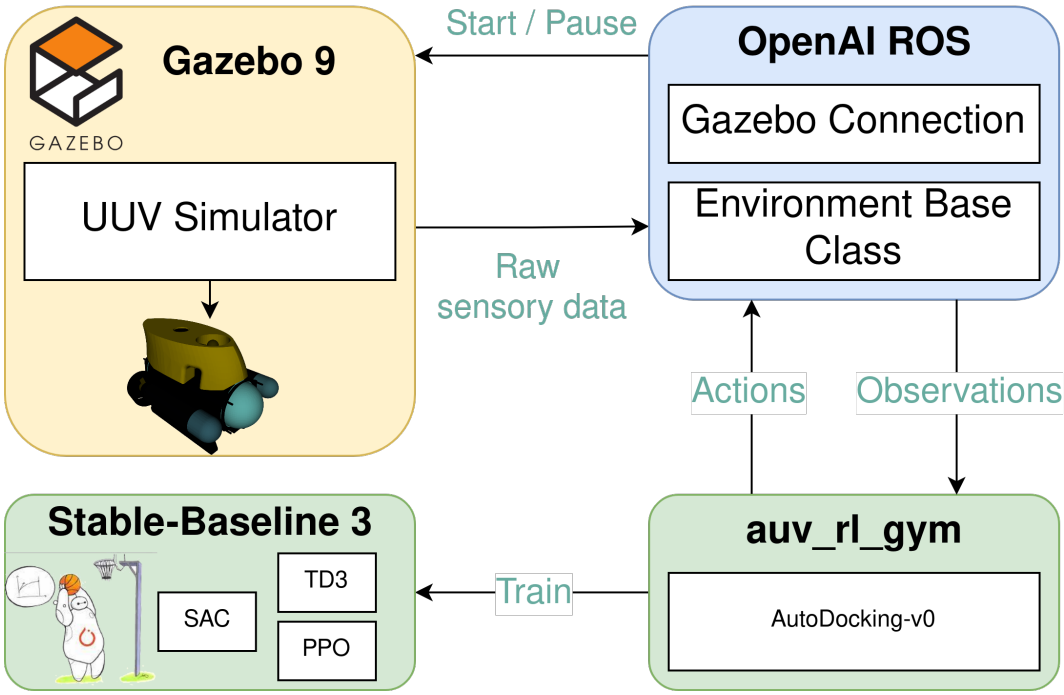


Figure 3.8: High-Level architecture of the software components required to build the RL training environment.

As illustrated in Figure 3.8, the environment designated as *AutoDocking-v0* serves as the foundational RL training class for our experiments. This environment interfaces with Gazebo 9 through the OpenAI ROS[3], which manages the simulation to update and extract the information from the operational environment.

The main environment aspects defined within the *AutoDocking-v0* context are:

- **State-space**: The state space was defined based on the current pose error and the linear and angular velocities of the AUV.
- **Initial states**: The AUV's initial state is set randomly at the start of every episode. This helps to avoid any bias arising from starting in a single fixed position. To ensure that the initial position of the AUV is not directly within the docking station or too close to it, it inhibited the starting positions along the x and y axes to be within 2 meters of the maximum workspace limit.
- **Terminal states**: The terminal states are cases where the agent has either driven the AUV outside the stipulated workspace bounds or driven it expertly to the goal position located at the docking station. Moreover, a timeout of five minutes was set to avoid long episodes without any representative ending.

[3]https://bitbucket.org/theconstructcore/openai_ros.git

- **Action space**: The action space is defined as a 3-dimensional vector [v_x, v_z, ω_z], representing x and z linear and the z angular velocities, respectively.
- **Reward function**: The reward is based on the pose error, as represented by equation 3-3.

$$R_t = -\left(\text{Heading Error} + \text{Euclidean Distance}\right). \tag{3-3}$$

The agent training is based on the models provided by *Stable-Baselines3* [80]. It provides open-source implementations of RL algorithms in Python, using as main machine learning libraries PyTorch [81]. Its interface allows the training of an agent in customized or predefined environments, aiming to provide a solution to validate different techniques from mature implementations of the main RL methods. The algorithms follow a consistent interface and are accompanied by extensive documentation, making it simple to train and compare different RL algorithms towards different applications and scenarios.

Stable-Baselines3 has a well-defined interface that allows different setups for training the models. To maintain methodological consistency, the hyperparameters for the candidate algorithms — PPO, SAC, and TD3 — were retained at their default settings. The sole modification pertained to the number of steps executed during the training phase. Table 3.3 gathers the number of steps considered for each training session.

Table 3.3: TD3, SAC, PPO number of training steps.

Method	**Number of training steps**
TD3	200,000
PPO	200,000
SAC	200,000

The training was conducted in real-time, and a total of two hundred thousand steps were considered for each training run. Each step is 1s, for a total of 55h training time. This number was chosen based on community-accepted benchmarks for similar environments, such as the inverse pendulum problem. The aim was to minimize the training duration in order to maintain a feasible work schedule and ensure the project's timely completion. Table 3.4 gathers the training duration for each proposed method. Section 4.1 details the computation resources considered to train the models.

The system presented is designed to facilitate the training of the methods under evaluation. During this phase, the key metrics we focus on are the average reward earned per step and the mean duration of each episode.

These metrics offer a straightforward and quantitative basis for assessing the effectiveness and computational efficiency of any RL algorithm. Figures 3.9 and 3.10 illustrate the results obtained for each method for the reward and episode duration mean, respectively.

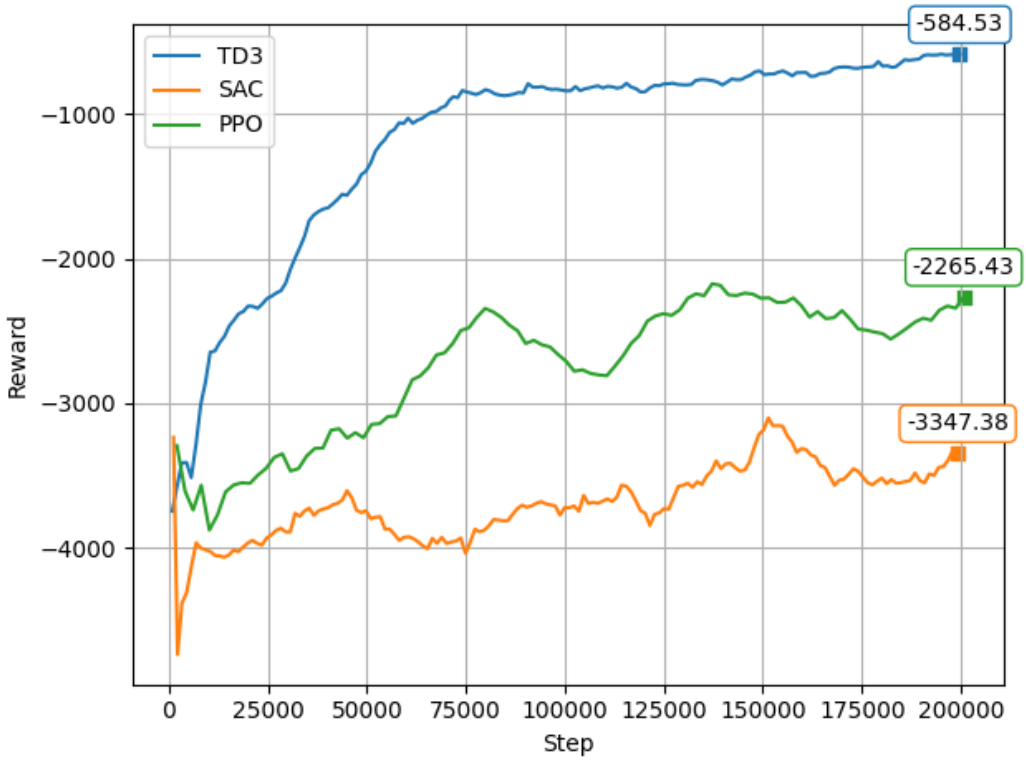


Figure 3.9: Step reward mean curve presented for each considered method.

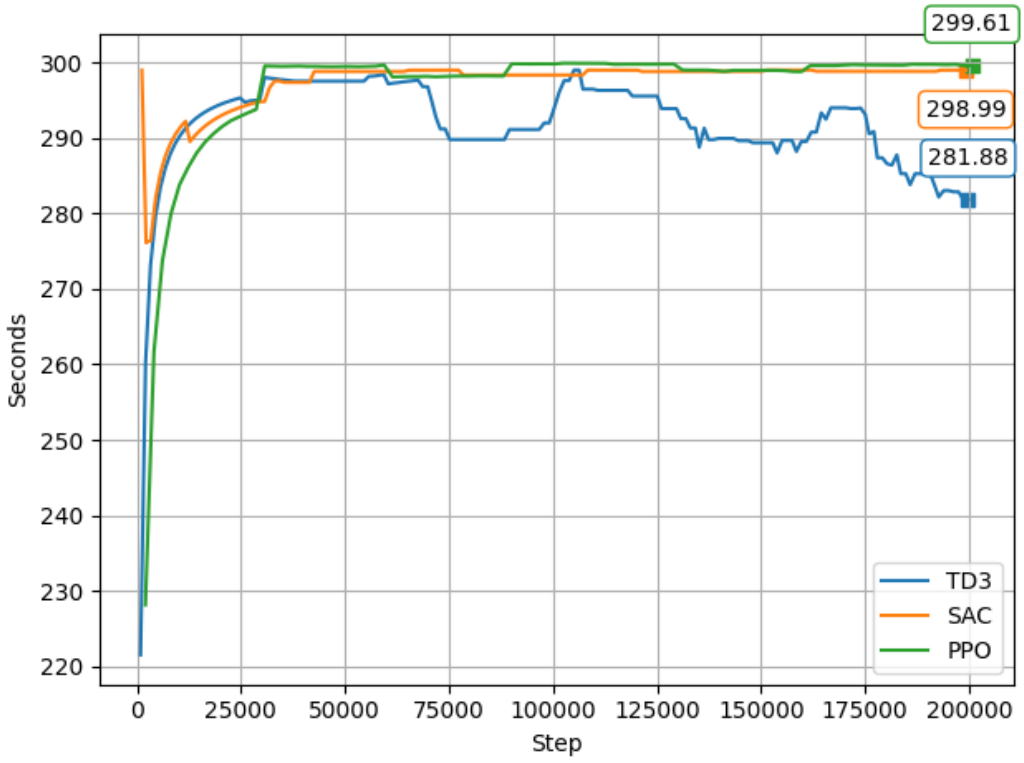


Figure 3.10: Episode duration mean curve presented for each considered method.

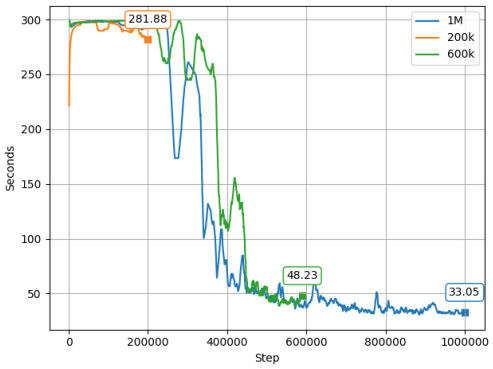
As observed, among the methods evaluated, TD3 exhibited superior performance by showing a faster potential convergence in both key metrics compared to the other algorithms. Note that episode duration first increases, because the robot learns to stay within the workspace, and then decreases, because it learns to reach the goal.

Consequently, based on the results presented, TD3 was selected as the foundational algorithm for the proposed agent in this study. To generate a model capable of effectively managing the control aspects of the auto-docking task, additional training steps are needed. Specifically, two extended training scenarios were considered: one with 600 thousand steps and another with 1 million steps. Table 3.4 outlines the time required to train each model under these conditions.

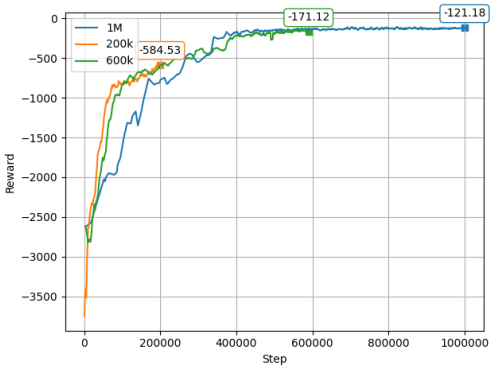
Naturally, the performance of the models for the different number of steps was evaluated. In line with the evaluation metrics presented previously, Figure 3.11 delineates the performance of TD3 models trained with 200 thousand, 600 thousand, and 1 million steps.

Table 3.4: Time required to train the models evaluated.

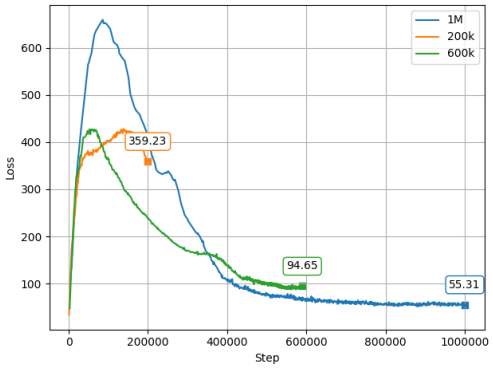
Method	**Steps**	**Time (hours)**
PPO	200,000	56
SAC	200,000	56
TD3	200,000	56
TD3	600,000	166
TD3	1,000,000	282



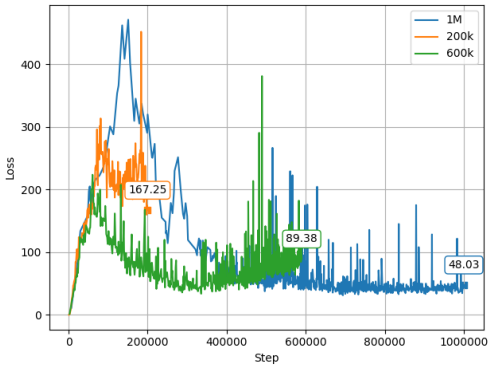
(a) Episode duration mean curve presented for the evaluated TD3 models.



(b) Step reward mean curve presented for the evaluated TD3 models.



(c) Actor loss curve for the evaluated TD3 models.



(d) Critic loss curve for the evaluated TD3 models.

Figure 3.11: TD3 performance comparison for 200k, 600k and 1M training steps.

Therefore, as observed from the curves illustrated in Figure 3.11, the TD3 method trained with 1 million action steps is considered as the RL method model to run the proposed experiments. The subsequent Section delineates the scenarios that will be executed, taking into account the methods previously discussed and illustrated.

3.2 Scenarios

This section outlines the proposed scenarios for evaluating the performance of RL and machine learning paradigms in executing the auto-docking task in various contexts. The first two scenarios are illustrated in Figure 3.12. The goal is to gain insight into the strengths and weaknesses of different approaches and determine the most effective strategies for achieving the desired result.

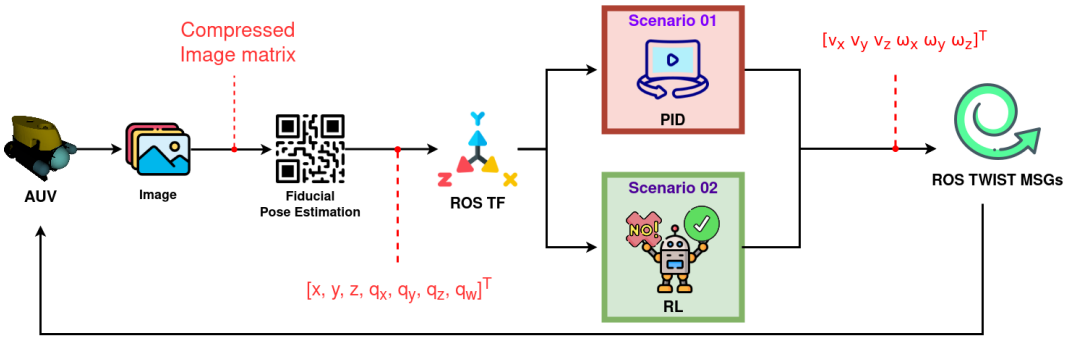


Figure 3.12: First two proposed scenarios involve utilizing fiducial pose estimation as the basis for them, with the main variations being RL and PID controllers.

As depicted in Figure 3.12, the first two scenarios are based on the transformation between the robot and the cage frames. The transformation is computed by the fiducial pose estimation algorithm, which receives images from the front camera of the AUV. The fiducial markers, introduced in Section 2.3, are used as a reference for the *apriltag-ros* package[4] to calculate the desired transformation. Figure 3.13 shows the frame arrangement in this application considered to compute the transformation between the AUV and the cage.

[4]https://github.com/AprilRobotics/apriltag_ros

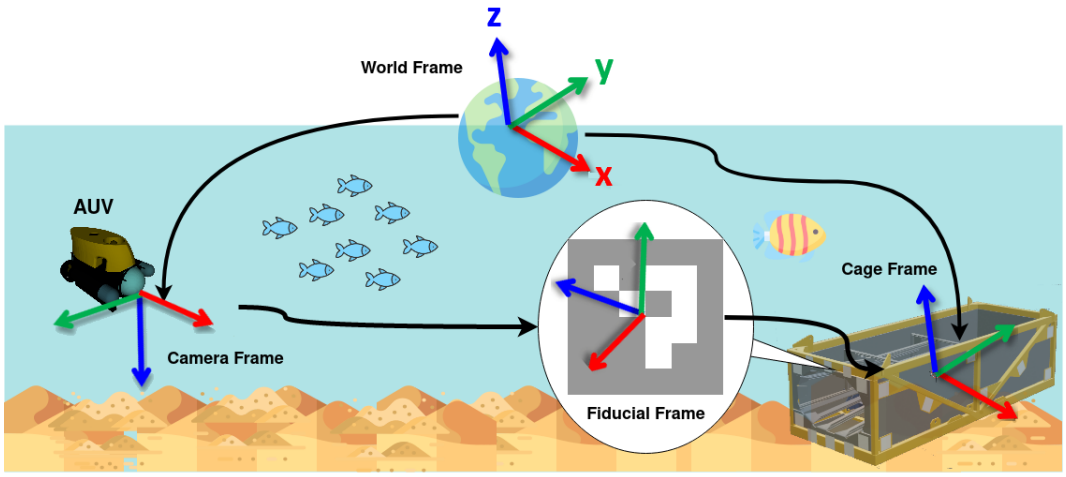


Figure 3.13: Frames arrangement considered for this work.

As illustrated in Figure 3.13, the *World* frame is defined in the ENU system (detailed in Appendix A.3.2) and is used as a reference frame for all other frames. The AUV frame represents the position and orientation of the AUV for the world frame. The *cage* frame represents the position and orientation of the docking station cage. There is no transformation between the world and cage frames, since the cage is stationary with respect to the world frame.

However, there is a transformation between the world and the camera frames. The camera is mounted on the AUV and is oriented so that its X-axis points towards the front of the robot and its Z-axis points downwards. This transformation is necessary to relate the camera's measurements to the world frame.

Finally, there is a transformation between the cage frame and a fiducial marker mounted on it. The fiducial marker is a reference point for the *apriltag-ros* package algorithms to estimate the cage's pose. This transformation is defined so that the Y-axis of the fiducial points upwards, the X-axis points towards the front of the fiducial, and the Z-axis points to the right of the fiducial. Figure 3.14 depicts the transformation tree that is computed by ROS in real-time in order to reflect the relative positions between the coordinate systems involved.

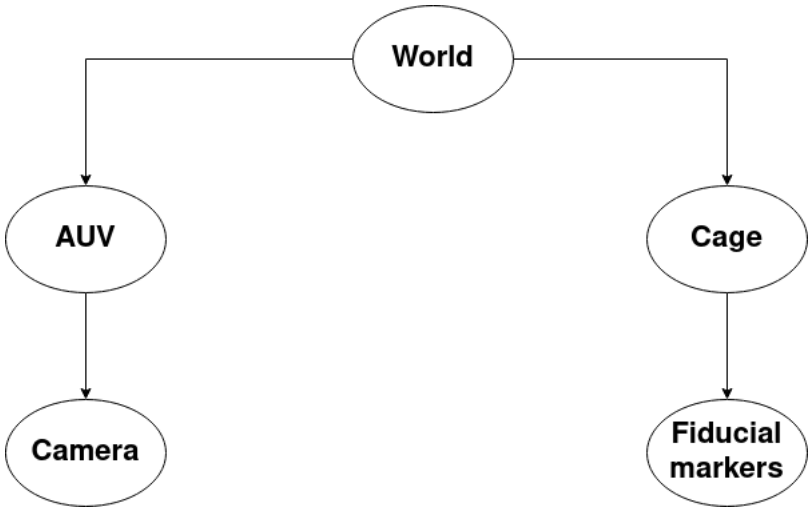


Figure 3.14: ROS TF tree that represents the frame transformations for this work.

By precisely defining the frames and their inter-relationships, a more concise representation of the scenarios can be provided:

- **Scenario 01:** in Scenario 01, the PID approach serves as the primary controller. This approach receives the target transformation from the *apriltag-ros* package through ROS TF messages. After processing the received frame offset, it sends to the AUV the desired velocities, through ROS Twist messages, which consist of linear and angular velocities, as defined in Equation 3-4.
- **Scenario 02:** in Scenario 02, the RL agent is considered the primary controller. As in the previous scenario, it receives the target transformation from the *apriltag-ros* package through ROS TF messages. After processing the received frame offset, it sends to the AUV the desired velocities, through ROS Twist messages, which consist of linear and angular velocities, as defined in the equation 3-4. The RL agent should be trained before performing those tasks. Once trained, it should be able to estimate a model of the environment.

As mentioned above, the ROS Twist messages are based on angular and linear velocities and can be represented as:

$$\begin{bmatrix} v_x & v_y & v_z & \omega_x & \omega_y & \omega_z \end{bmatrix}^\top. \tag{3-4}$$

The linear velocities and angular velocities in the ROS Twist message are correlated with the position and orientation of an object. The variables v_x, v_y, and v_z represent the linear velocities of an object along the x, y, and z axes, respectively, as represented by the ROS Twist message, and the variables ω_x,

ω_y, and ω_z represent the angular velocities of an object around the x, y, and z axes, respectively, as represented by the ROS Twist message.

The first two experimental scenarios are based on the pose estimation capabilities of the *apriltag-ros* package. However, the third and fourth scenarios incorporate a different approach to pose estimation, instead using a CNN-based solution. Despite this change, the RL agent and PID methods used in the first two experiments remain the same. By comparing the results of these scenarios to those of the first two, we can assess the impact of CNN's performance on the overall system. Figure 3.15 illustrates the setup for the third and fourth scenarios.

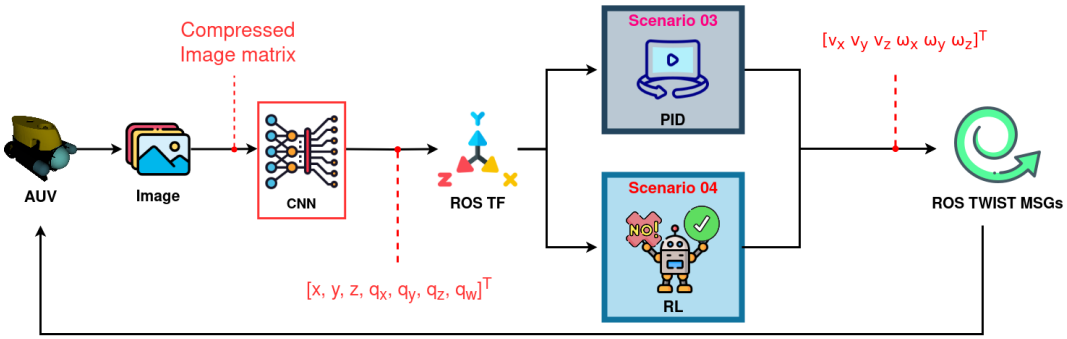


Figure 3.15: Third and fourth scenarios involve utilizing a CNN for pose estimation, with the RL and PID methods implementing the main variations.

As an overview, the third and fourth scenarios can be detailed:

- **Scenario 03:** in Scenario 03, the PID approach serves as the primary controller. This approach receives the target transformation from the CNN-based pose estimation algorithm, through ROS TF messages. After processing the received frame offset, it sends to the AUV the desired velocities, through ROS Twist messages, which consist of linear and angular velocities, as defined in equation 3-4.
- **Scenario 04:** this scenario follows the same workflow as the previous, with one notable difference: the primary controller is the RL agent.

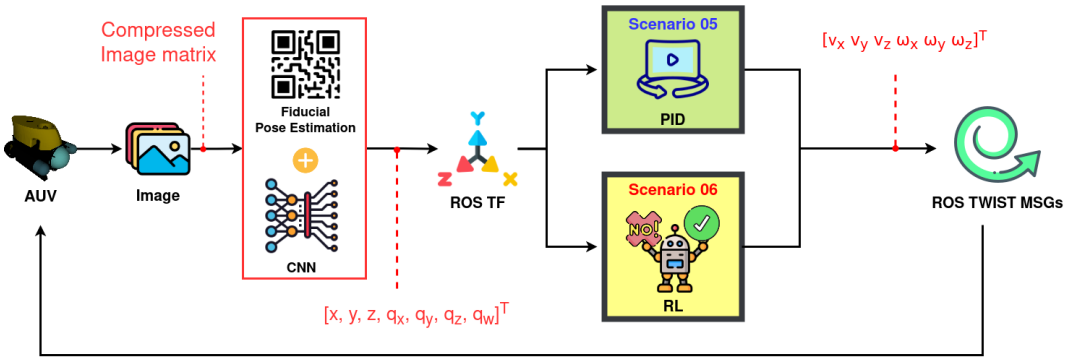


Figure 3.16: Last two proposed scenarios for this work, combining fiducial and CNN pose estimation approaches as the basis, with the main variations being the RL and PID controllers.

Figure 3.16 illustrates the last arrangement considered for this work. In this case, the pose estimation is a combination of fiducials-based and CNN-based approaches, using the fiducials if their position can be properly estimated, and the CNN otherwise. As an overview, they can be detailed:

- **Scenario 05:** in Scenario 05, the pose estimation task is performed by the combination of fiducials-based and CNN-based approaches. The main goal is to evaluate the sum of the contribution of those two methods when working together. For Scenario 05, the PID controller is considered.
- **Scenario 06:** this scenario follows the same workflow as the previous one, with one notable difference: the primary controller is the RL agent.

Once the experimental scenarios are defined, the workflow expected for conducting the experiments is fully designed. Consequently, the final phase before contrasting the proposed arrangements is related to the evaluation metrics specification. The subsequent section delves into the methodologies adopted for the auto-docking challenge.

3.3 Evaluation Metrics

These metrics collectively provide a comprehensive view of the system's performance, ensuring that any improvements or modifications are evaluated against a robust set of criteria. For the auto-docking task, a distinct set of evaluation metrics was defined to accurately assess the performance of the proposed methodologies. Building upon the insights from [82], the following metrics were selected:

- **Success Rates**: The ratio of the number of successful docking operations to the total number of trials. This is calculated as:

$$\text{Success Rate} = \frac{\text{Number of successful dockings}}{\text{Total number of trials}}. \tag{3-5}$$

- **Convergence Time**: Represents the time taken during one episode to complete a docking operation, from initiation to successful docking.
- **Final Positioning Error (Diff)**: This metric gauges the discrepancy in position from the desired docking location. It can be calculated as:

$$\text{Diff} = \|\text{Desired Position} - \text{Final Robot Position}\|. \tag{3-6}$$

- **Cos of Angles for Orientation**: Evaluates the orientation error based on the cosine of the angle difference. It's given by:

$$\cos(\theta) = \frac{\text{Desired Orientation Vector} \cdot \text{Final Robot Orientation Vector}}{\|\text{Desired Orientation Vector}\| \times \|\text{Final Robot Orientation Vector}\|}. \tag{3-7}$$

- **Accumulative Positioning Error**: This metric records the cumulative error at every time step, providing insights into how the error evolves over time:

$$\text{Accumulated Error} = \sum_{t=0}^{T} |e(t)|. \tag{3-8}$$

3.4 Overview

In this chapter, it was outlined the methodology used to conduct the experiments that were aimed at evaluating the effectiveness of the auto-docking task presented in this work. To accomplish this, it was identified a series of steps, which are illustrated in Figure 3.17. These steps were carefully designed to ensure that the results obtained would be reliable and meaningful.

Overall, the proposed methodology involved a number of important considerations, such as the selection of appropriate docking algorithms, the establishment of realistic simulation conditions, and the identification of suitable performance metrics.

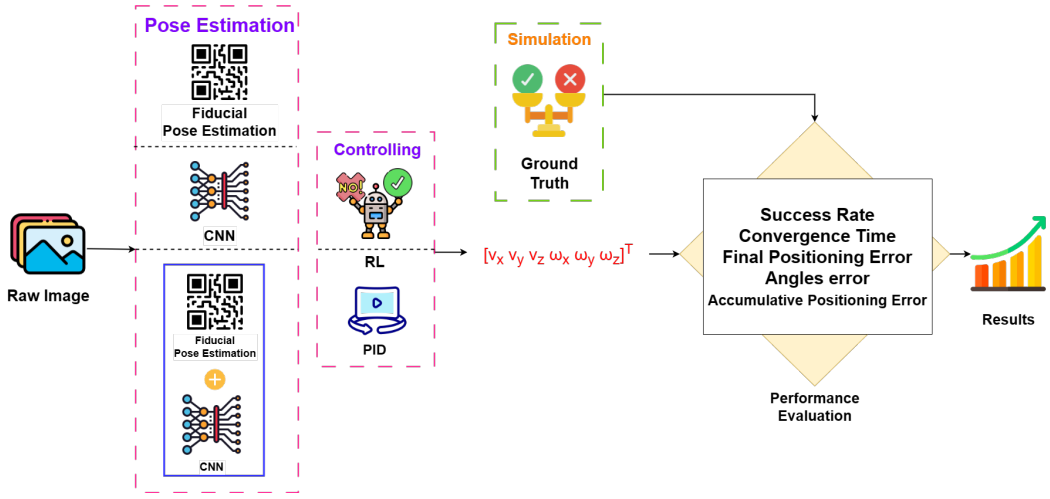


Figure 3.17: Auto-docking methodology adopted for this work main workflow.

As Figure 3.17 illustrates, the auto-docking pipeline should receive a processed image as a primary input, with the chosen method of image enhancement appropriately applied. As a variation in the proposed arrangements, pose estimation is performed using fiducial markers, CNN, and a combination of them. The estimated pose is forwarded to the second step, the controlling task. The controlling task is performed by the RL and PID techniques.

At this point, the resources required to evaluate all scenarios are available, since the ground truth data is provided by the 3D simulated environment. Finally, the evaluation metrics can be applied and their results can be assessed in order to provide the conclusions and main assumptions as the output of this work.

The next Chapter details the workflow considered to build the experimental setup aimed to apply the proposed methodology.

4
Experimental Setup

In this Chapter, it is outlined the experimental setup procedure designed to implement the methodology described in the preceding Sections. The aim is to provide a detailed account of the steps involved in the experimental setup to enable reproducibility and facilitate the understanding of the methodology and the target results.

Implementing the auto-docking task methodology proposed in the previous Chapter requires two main types of resources: computational and simulation. The following Sections describe the experimental setup implemented to achieve the aimed goals.

4.1
Computational Resources

The proposed Auto-Docking methodology involves comparing different machine learning methods, which require training steps to optimize their performance. Therefore, it is crucial to use a computational environment that has hardware capable of executing both simulation and training tasks efficiently. In this regard, external servers were deemed necessary for this project.

By using external servers, the computational burden can be distributed among multiple machines, which can significantly reduce the time required for training and testing. Moreover, these servers can provide higher processing power and memory capacity than typical local computers, enabling the implementation of more complex and demanding algorithms. Table 4.1 represents the servers' hardware configuration for this work. As Table 4.1 represents, two servers are responsible for executing the RL and CNN methods experiments. A Secure Shell (SSH) remote access was considered to log in on them.

Table 4.1: Hardware details of the machine learning servers.

Server ID	RAM Memory	GPU	Hard Disk Capacity	CPU
01	512 GB	4 x NVidia A100-SXM4 80 GB	8 TB	x86-64 AMD EPYC 7742 2.25 GHz
02	512 GB	4 x NVidia A100-SXM4 80 GB	8 TB	x86-64 AMD EPYC 7742 2.25 GHz
03	512 GB	4 x NVidia A100-SXM4 80 GB	8 TB	x86-64 AMD EPYC 7742 2.25 GHz

4.2 Simulation

The simulation tool considered for the auto-docking task is Gazebo 9, with the UUV simulator running on top of it. Section 2.1 details the criteria considered for choosing the UUV Simulator as the solution for providing the simulated underwater environment. The following Sections detail the steps to set up the experiments proposed throughout this document.

4.2.1 Ocean and Underwater Environment

To create a realistic underwater environment, it is crucial to include the ocean cube, which serves as a framework for applying physics, dynamics, interactions, and rendering effects using Gazebo. The ocean cube determines the boundaries of the simulated environment, ensuring that the simulation accurately reflects real-world conditions. This work implemented an ocean cube with specific dimensions, as shown in Figure 4.1. A comprehensive and immersive underwater simulation is created by incorporating the ocean cube.

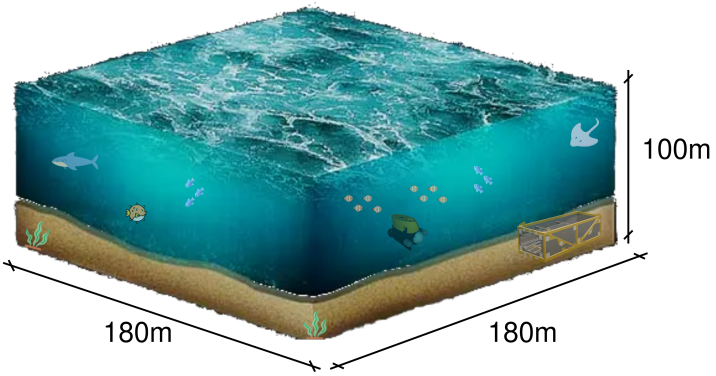


Figure 4.1: Ocean cube dimensions adopted for the simulated environment.

In order to simulate the underwater hydrodynamics interaction, the UUV Simulator plugin proposed by [57] was considered in this work. The hydrodynamic plugin used in the UUV Simulator is a component that simulates the hydrodynamic forces and moments acting on an underwater vehicle. The plugin is based on the added mass and damping coefficients of the vehicle, which are obtained through experiments or numerical simulations. It takes into account the vehicle's motion and computes the corresponding hydrodynamic forces and moments, which are then applied to the vehicle's dynamics model to update its position and orientation. The hydrodynamic forces include drag, lift, and added mass forces, while the moments include those resulting from roll, pitch, and yaw. Moreover, the plugin also includes features such as the

capability to simulate waves and currents, as well as support for different vehicle geometries and configurations. The hydrodynamic coefficients can be set for each vehicle individually, enabling users to simulate a wide range of underwater vehicles with varying hydrodynamic properties.

As the main configuration parameters for the underwater rendering, the fog values considered for the simulation are represented in Table 4.2.

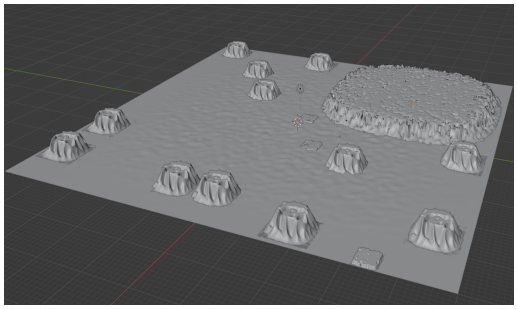
Table 4.2: Fog Technical Specifications.

Parameter	**Value**
Color	17, 31, 39, 1.0 (RGBA)
Linear density	linear
Starting range	15 meters
Ending range	21 meters

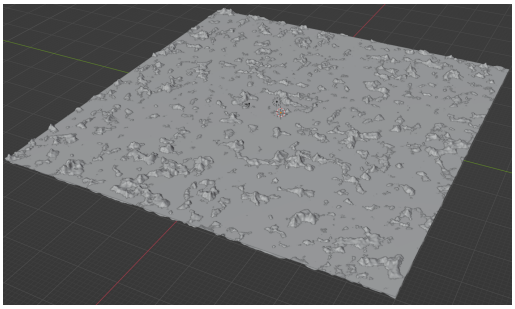
4.2.2 Seabed

The seabed has been identified as a crucial element not only for serving as a landmark in navigation algorithms but also for feature extraction for various other purposes. For this project, the goal was to create a seabed that followed a random pattern, emulating the real-world distribution encountered during AUV operations. Two techniques were considered for this purpose. The first technique involved using a specialized Blender plugin specifically designed for generating realistic surfaces, named Another Noise Tool Landscape. The second one is based on 2D height map surface generation, where a bitmap image containing different grayscale values for each pixel is converted into a surface that corresponds to the grayscale range, considering a scaling factor. Figure 4.2 illustrates the surfaces generated by the two techniques.

In an effort to create a more authentic surface, the seabed depicted in Figure 4.2(a) was chosen for this project. To enhance the fidelity of the underwater environment simulation, a texture that closely resembles the characteristics of a real-world seabed was applied. This approach contributes to a more accurate reproduction of the underwater environment.



(a) Seabed surface generated with ANT Landscape.



(b) Seabed surface generated by applying the height map technique.

Figure 4.2: Seabed surfaces considered for this work.

Figure 4.3 displays the selected seabed component, featuring the applied texture that mimics a realistic seabed.

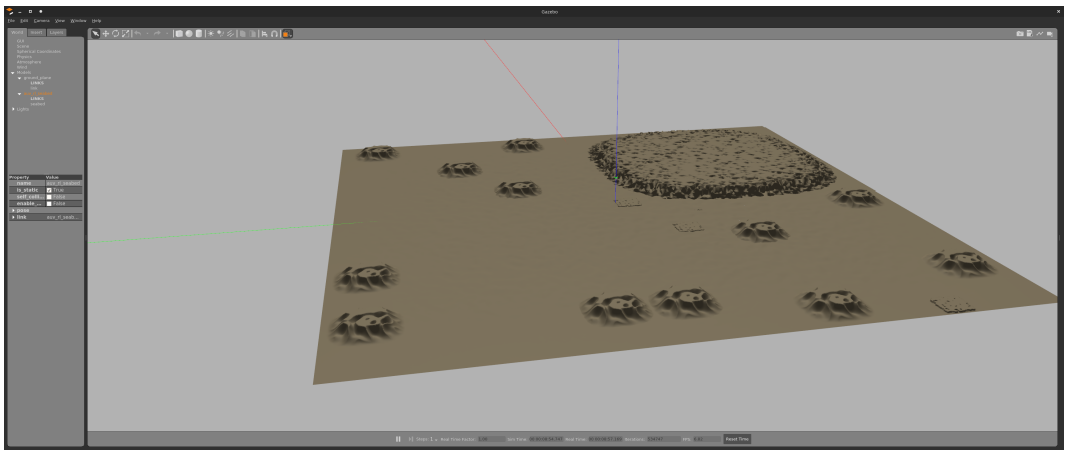
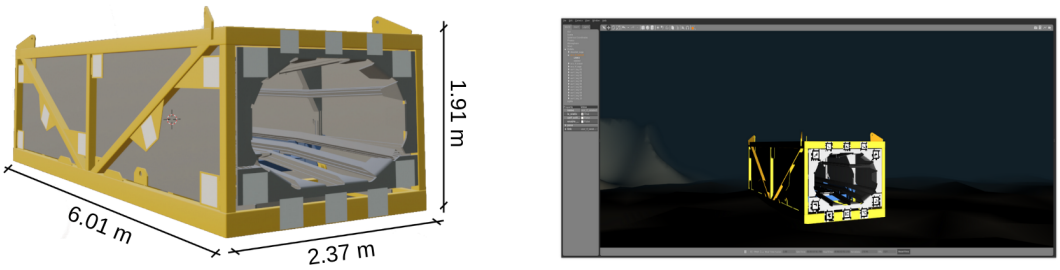


Figure 4.3: Seabed surfaces considered for this work with the textured surface applied on the simulation environment.

4.2.3 Docking Station (Cage)

In this project, the docking station also referred to as a cage or garage, is designed as a modular rectangular prism-shaped frame. The cage is vital for auto-docking tasks, as it functions as a base for recharging, and mission data transferring and serves as a georeferenced point. Further details about the cage's role in auto-docking tasks can be found in Section 1.3.1. Figure 4.4 showcases the cage employed for this work.

As Figure 4.4 details, the docking station has 6.01 meters of length, 2.37 meters of width, and 1.91 meters of height. For the proposed experiments, twenty-three fiducial markers were placed at the frontal side of the cage.



(a) Docking station dimensions.

(b) Docking station deployed in the simulation environment.

Figure 4.4: Docking station considered for this work.

The docking station is equipped with fiducial markers known as ApriL-Tags, which are further explained in Section 2.3.2. These markers are crucial for the performance of the proposed pose estimation algorithm, as they serve as visual references. Furthermore, the proposed ApriLTag solution utilizes a bundle concept, where the pose estimation of the entire bundle is determined through filtered estimations of individual markers rather than computing separate estimations for each marker. Figure 4.5 illustrates the proposed ApriLTag bundle for this work.

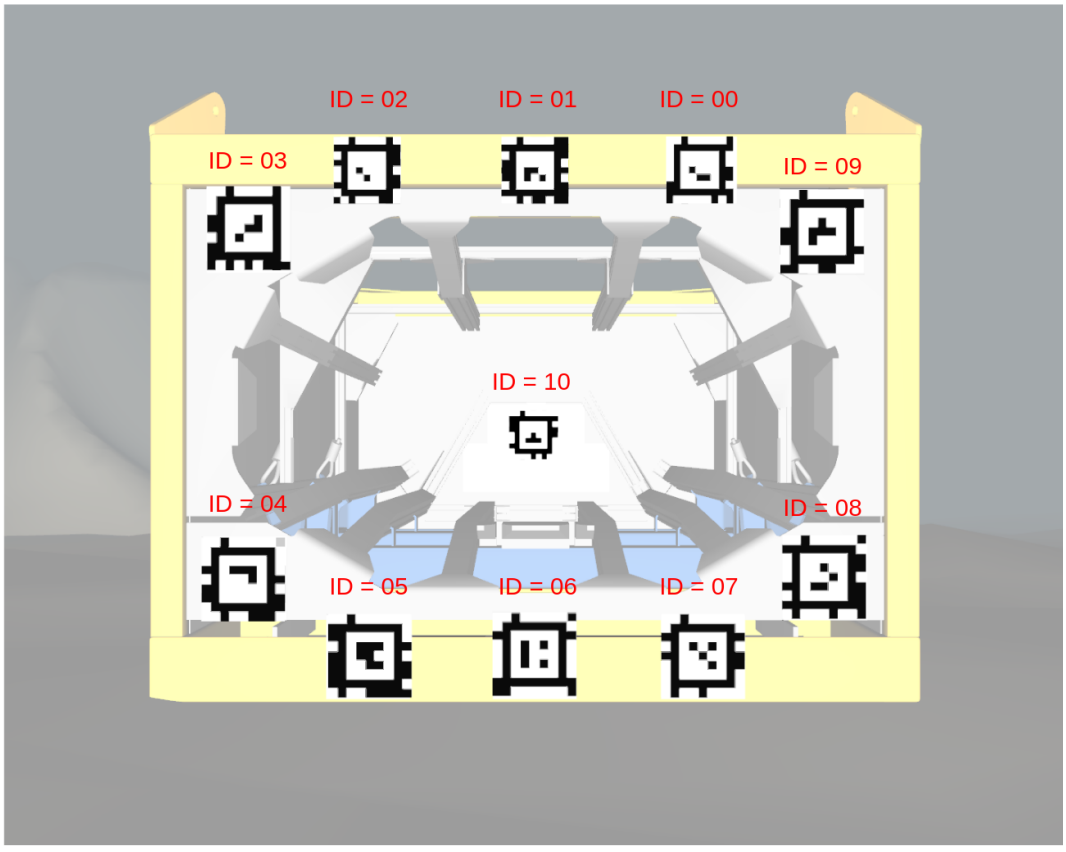
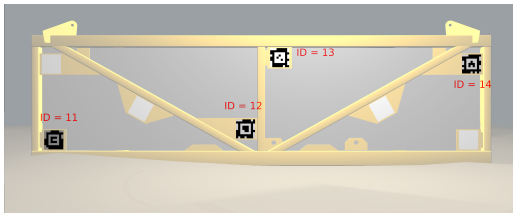
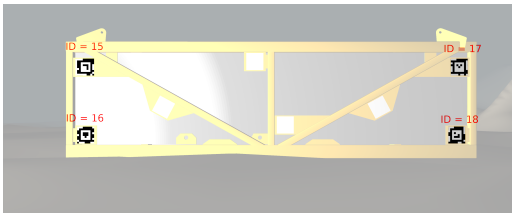


Figure 4.5: ApriLTag markers frontal arrangement on the docking station.

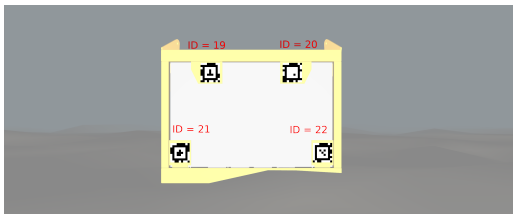
Figure 4.6 illustrates the bundles for the right, left, and rear sides of the cage.



(a) Cage fiducials bundle of the left side. Tags from ID 11 to 14.



(b) Cage fiducials bundle of the right side. Tags from ID 15 to 18.



(c) Cage fiducials bundle of the rear side. Tags from ID 19 to 22.

Figure 4.6: ApriLTag markers right, left, and rear arrangements on the docking station.

Figure 4.5 presents the ApriLTags from the *tagStandard41h12* family [58] installed on the cage. A total of twenty-three fiducials are attached to the cage, with tags *ID 00 to ID 22* forming the bundle utilized for pose estimation before the AUV reaches the docking homing position. Once the homing position is attained, the tag with *ID 10* facilitates a smooth approach.

4.2.4 Desistek Saga AUV

The AUV considered for the experiments is the Desistek SAGA ROV, an inspection class ROV manufactured by Desistek, illustrated in Figure 4.7. The dimensions of this vehicle are 420 mm × 330 mm × 270 mm. The mass of the vehicle is 10 kg, and the maximum depth is 250 m. Maximum surge speed is 3 knots and maximum heave speed is 1 knot. The vehicle has three thrusters, which supply a maximum force of 10 N. One of these thrusters is directed vertically, while the other two are located at the right and left sides and are directed horizontally.

Although the Desistek SAGA typically functions as a ROV in real-world scenarios, it is treated as an AUV for this project. This is because the UUV Simulator offers all the necessary features to operate it autonomously, eliminating the need for remote control.

(a) Desistek Saga ROV.

(b) Desistek Saga ROV URDF representation.

(c) Desistek Saga ROV deployed in the simulation environment.

Figure 4.7: Desistek Saga robot, the AUV considered for this work scope.

4.2.4.1 3D modeling

The AUV 3D model was based on the description file implemented by the UUV Simulator package and considers all of the real dimensions of the robot, sensors, mechanical joints, actuators, and the color pallet specified by the Desistek Saga engineers.

4.2.4.2 Sensors

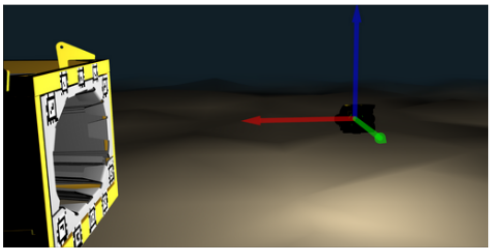
The Desistek Saga vehicle is equipped with an array of sensors designed to enhance its performance, efficiency, and data collection capabilities in underwater environments. Considering the sensors implemented in UUV Simulator, the AUV has:

- **Mono camera:** a Red, Green, Blue (RGB) camera positioned in the front part of the vehicle that are supported by one pair of LEDs.
- **IMU:** models the data collection of acceleration, angular velocity, and magnetic field to assist in navigation and stability within the simulated environment.
- **Forward-Looking Sonar (FLS):** the FLS provides real-time visualization of obstacles in front of the AUV, ensuring safe and efficient operation.
- **CTD Probe:** the CTD probe gathers data on salinity, temperature, and depth, enabling researchers to study the physical and chemical properties of the water column.
- **GPS:** the Desistek SAGA is also outfitted with a GPS system, which can help locate the robot in cases of loss or emergencies. Its positive buoyancy is designed to enable the vehicle to surface, allowing the GPS to effectively pinpoint its location.

Among the sensors mentioned, this work utilizes a simulated monocular camera with a resolution of 768 x 492 pixels. It should be noted that the UUV Simulator does not replicate the behavior of LED lights. However, as part of the proposed solution for this work challenge, a Gazebo plugin comprising the LED physics behavior was implemented.

The LED plugin implemented adheres to the design outlined in [57]. This plugin enables the flashing and dimming of lights and visual objects within a 3D model. By providing specific parameters to the plugin, the application can precisely control which lights and visuals to blink. Additionally, the application can customize the duration and interval time for each light's flashing, allowing for a tailored and dynamic experience.

Figure 4.8 illustrates the comparison between two scenarios in the simulation environment, showcasing the LED functionality in one and its absence in the other. This visual representation highlights the difference between active and non-active LED states, enabling a clear understanding of the LED system's impact within the simulated context.



(a) LEDs turned on in the simulated environment.

(b) LEDs turned off in the simulated environment.

Figure 4.8: The impact of LEDs usage in the simulation environment.

4.2.4.3 Controlling

The controlling stack is founded on the cascade PID approach, specifically designed and implemented for this vehicle by UUV Simulator. Figure 4.10 demonstrates the primary components integral to this task, providing a clear visualization of the elements and their interconnections within the control system.

As Figure 4.10 depicts, the controlling stack implemented for the AUV has two main ROS packages, *desistek_saga_control* and *uuv_control_cascade_pid*. The *desistek_saga_control* is responsible for bringing up all the application entities through the *start_cascade_pid_with_teleop* launch file, which has a version named as *start_cascade_pid* that covers the same scope as the first, however without the manual input controller. This

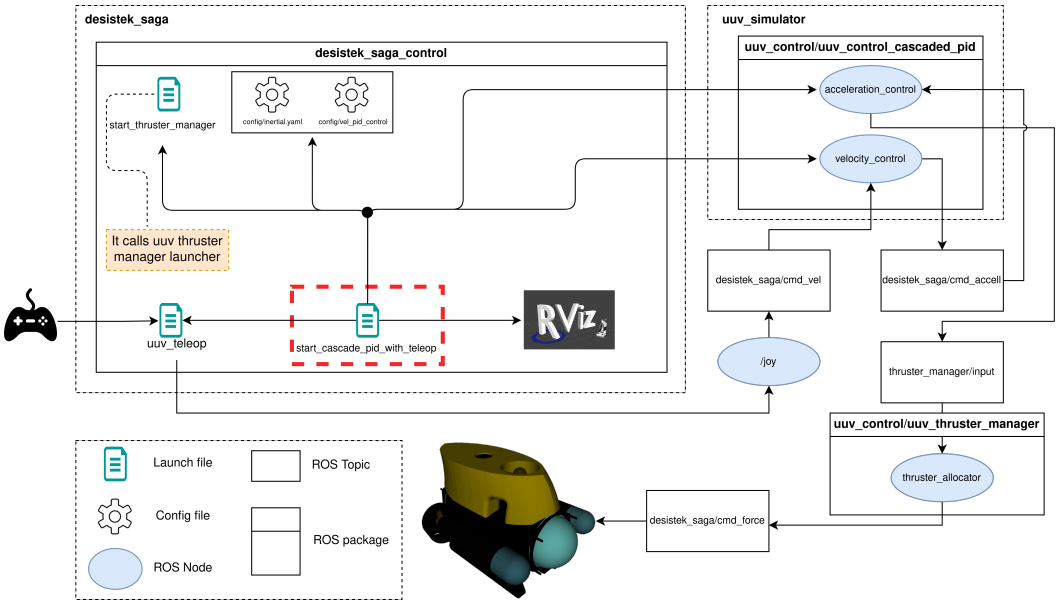


Figure 4.9: The software stack implemented to control the AUV considering the cascade PID solution proposed by UUV Simulator.

launch file is input for the *uuv_teleop*, RViz, and *start_thruster_manager*. The *uuv_teleop* entity brings up the stack responsible for managing the manual inputs through a compatible joystick[1]. The interaction with *RViz* is accomplished through a config file, which specifies the starting arrangements for the widgets to visualize the relevant data coming from the simulation. The *start_thruster_manager*, detailed in Figure 4.10, is responsible for the ratio allocation between forces and percentage output, considering a Thruster Allocation Matrix (TAM) as the config file. The *thruster_allocator* node manages the computation of allocation according to the controller setpoints and publishes the results in the *thruster_manager/input topic*.

The second package elucidated in Figure 4.10 is the *uuv_control_cascade_pid*. This package manages the cascade controllers; the inner controller deals with acceleration, while the outer controller manages velocity setpoints and outputs target accelerations. The acceleration controller subsequently outputs these target accelerations, which are processed by the *thruster_allocator* to output the necessary forces to achieve the intended goal. Ultimately, these forces are processed by the simulation and applied to the AUV.

In summary, the control stack presented, implemented by UUV Simulator and adapted to meet these work requirements, serves as an abstraction layer for the control applications. It is configured and designed to align with the real-world physics models of the AUV, as the proposed work requires pro-

[1]The precedence of the joystick manual input is higher than the AUV autonomy inputs.

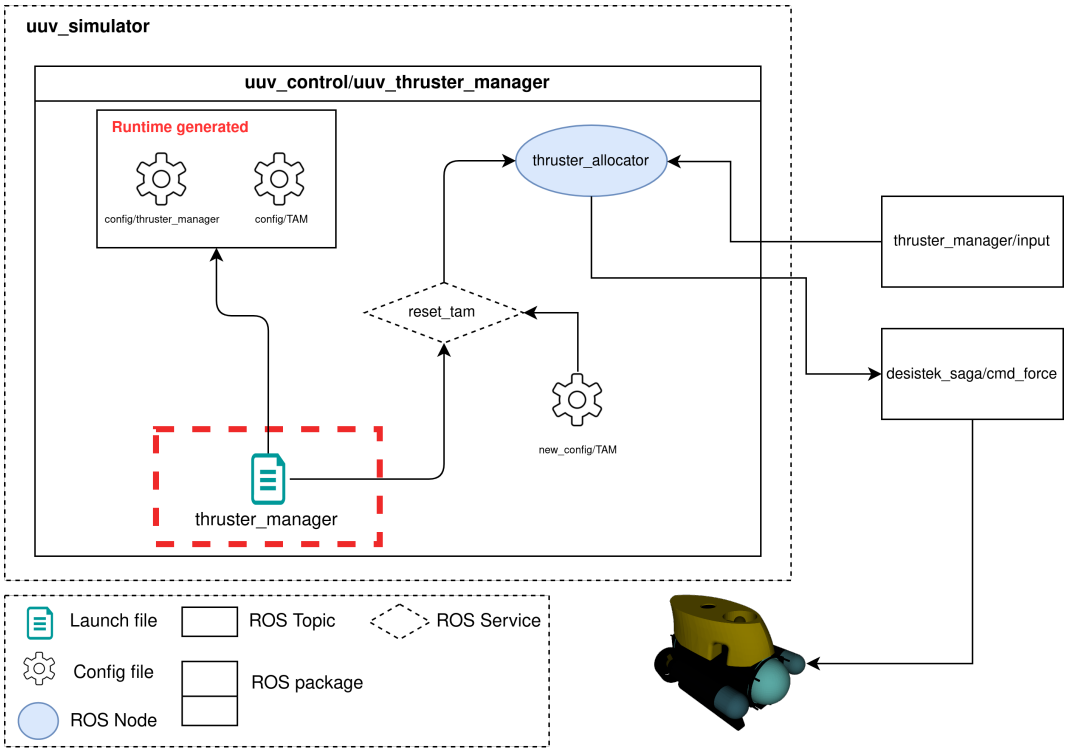


Figure 4.10: The software stack implemented to control the AUV considering the cascade PID solution proposed by UUV Simulator.

viding velocity setpoints for processing by the outer controller. This approach ensures that the AUV is equipped with appropriately designed controllers to effectively execute the proposed experiments. Figure 4.11 illustrates the detailed abstraction.

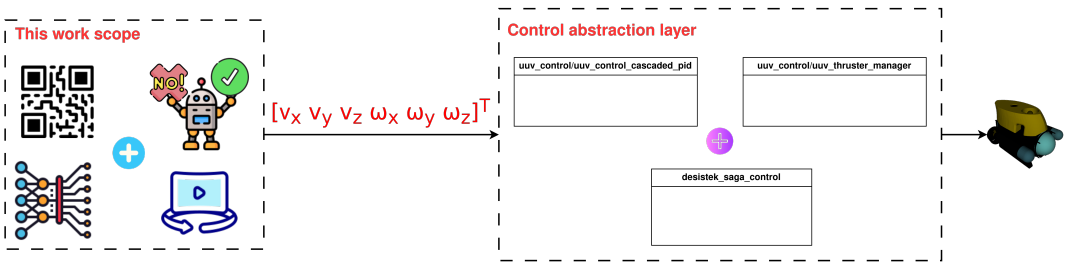


Figure 4.11: Control abstraction considered for this work.

4.3 Experiment's Workflow

The experiment's workflow detailed in this Section covers the steps to consolidate the resources to enable the execution of the six proposed scenarios. This also includes the mechanisms for capturing and monitoring the key metrics essential for the understanding and evaluation of each scenario.

To facilitate the management, loading, and configuration of each proposed algorithm, as well as the automated recording of metric values for each episode, a dedicated software stack is necessitated. This stack aims to execute these tasks in an automated fashion. Consequently, the architecture (represented by a simplified UML [83] class diagram) illustrated in Figure 4.12 has been implemented to offer a modular interface, enabling effortless switching between different solutions.

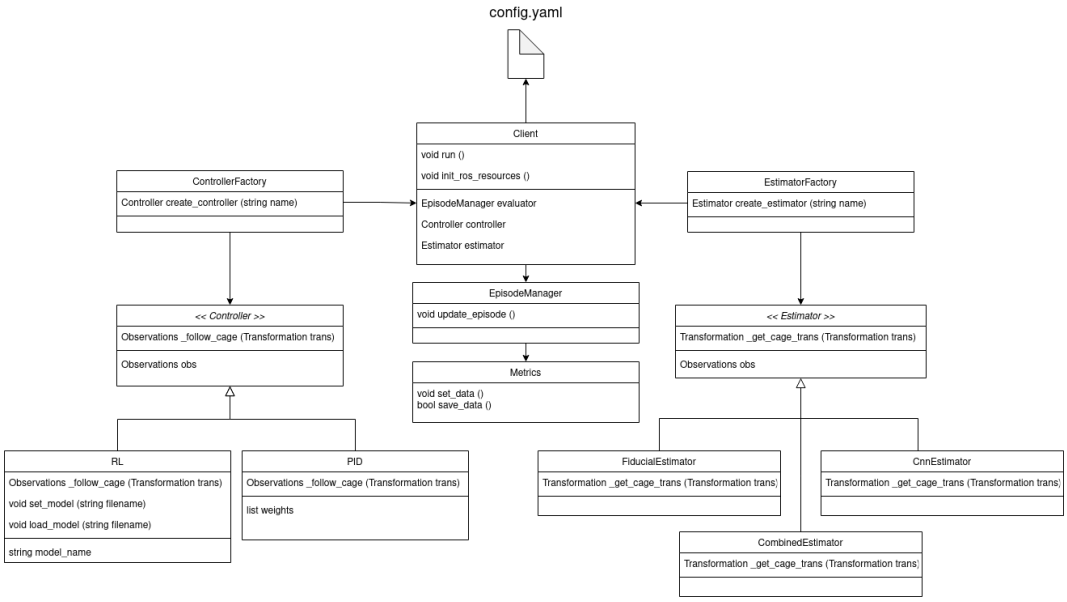


Figure 4.12: Simplified UML class diagram implemented to manage the experiments.

As illustrated in Figure 4.12, the software stack engineered for the seamless execution of the six predefined scenarios comprises five core classes:

- ***Client***: This class orchestrates the overall behavior of the target system. It dynamically switches between controller and estimator methodologies as specified in the *config.yaml* file and executes the main loop as detailed in Figure 4.13 to carry out the assigned scenario.
- ***Controller***: This abstract class serves as an interface, laying down the essential methods and attributes that all inheriting controller classes must implement.
- ***Estimator***: Similar to the Controller class, this is an interface class that prescribes the mandatory methods and attributes for all descendant estimator classes.
- ***EpisodeManager***: This class oversees the life cycle of each episode within a given scenario. It handles tasks such as robot respawning and metric tracking for each episode.

– ***Metrics***: Managed by the *EpisodeManager* class, this class is responsible for saving, loading, recording, and formatting all the relevant metrics.

The software stack was implemented in accordance with the algorithm outlined in Figure 4.13, ensuring each scenario is initiated, executed, and concluded in alignment with the methodology presented in Chapter 3.

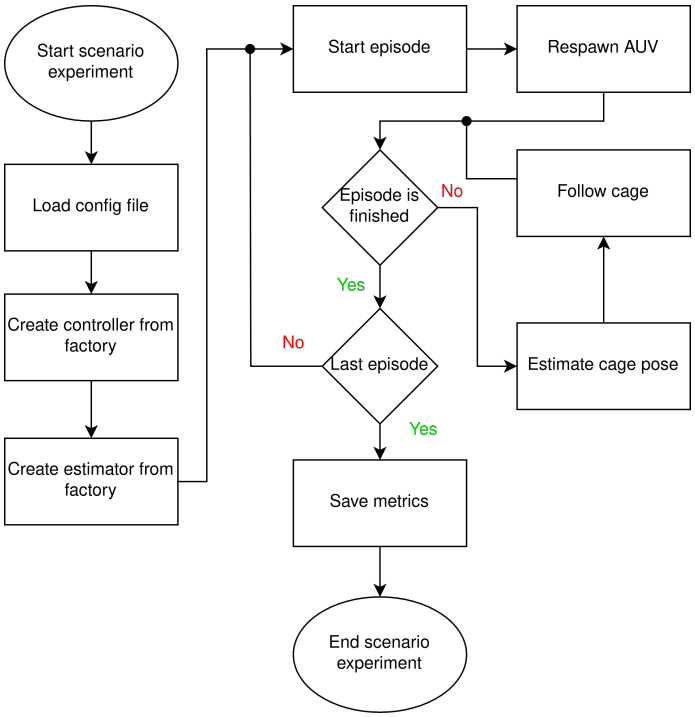


Figure 4.13: Algorithm considered to execute a scenario.

As illustrated in Figure 4.13, three steps rely on defining specific values, such as the number of episodes, the episode starting pose, and the conditions to consider an episode as done. The starting points were chosen randomly, in six different radius distances, as illustrated in Figure 4.14.

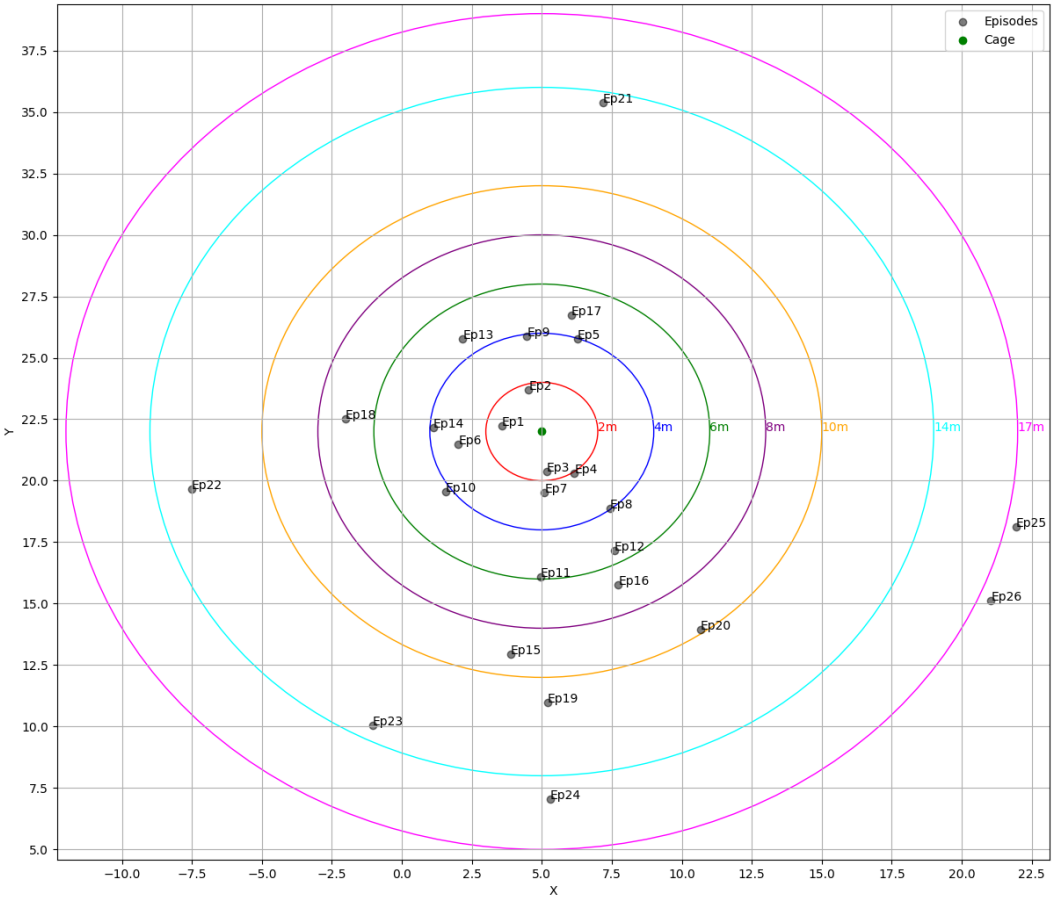


Figure 4.14: Initial points considered for the experiments.

As the techniques approached in this work must be evaluated considering the same setup, the points were kept the same for all scenarios. The yaw angles of these starting points were calibrated to ensure that the AUV was always oriented toward the cage at the outset of an episode. Figure 4.15 showcases a collection of images captured by the AUV camera at the commencement of each episode.

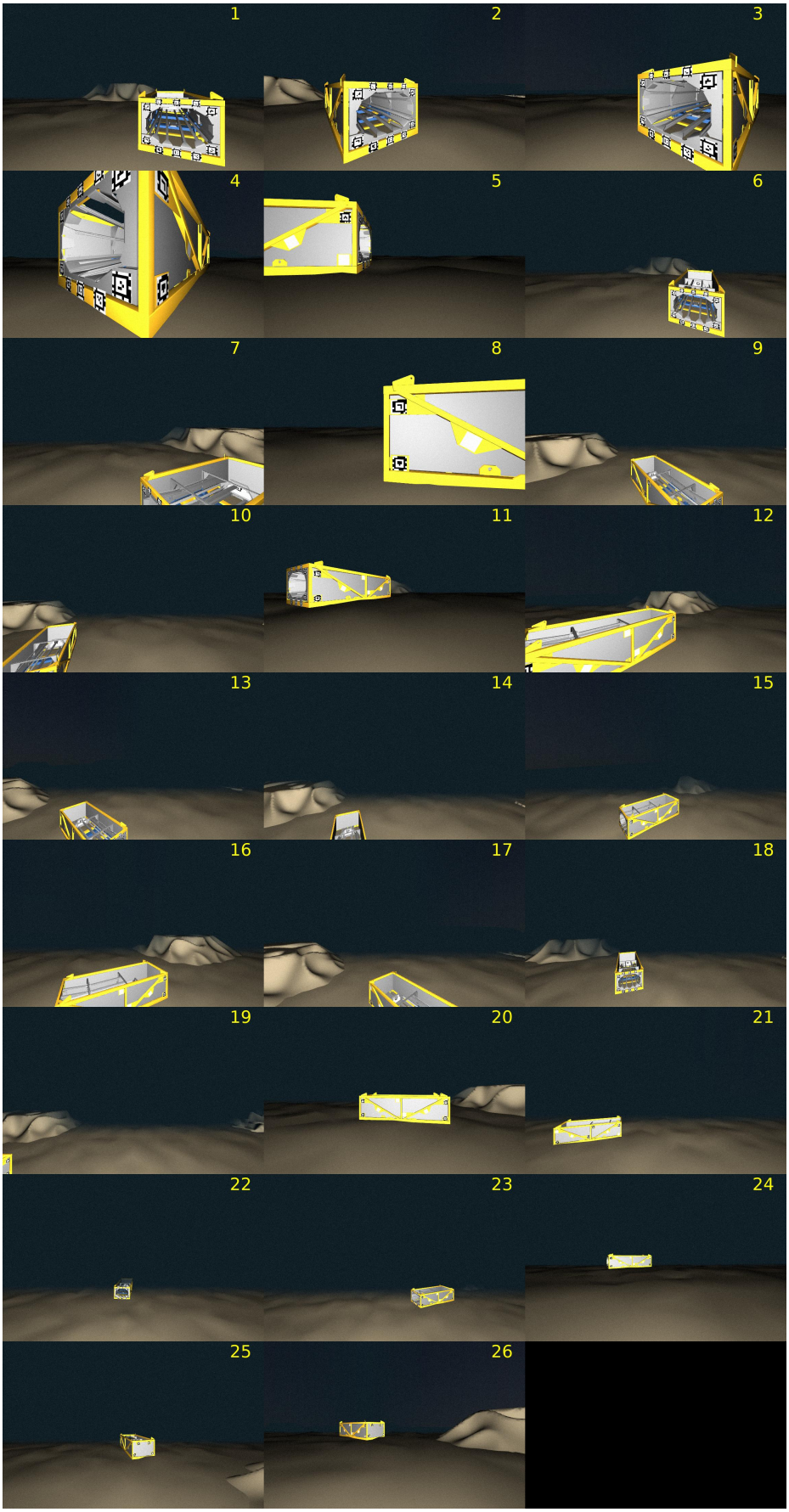


Figure 4.15: Initial points images acquired from the AUV's camera before starting each episode.

In conclusion, the stopping conditions for each episode hinge on either the AUV being driven outside the predefined workspace boundaries or successfully navigating to the target location at the docked pose. Additionally, a five-minute timeout has been instituted to prevent episodes from dragging on without reaching a meaningful conclusion. Table 4.3 collates the various stopping conditions proposed for each experimental scenario.

Table 4.3: Episode termination conditions.

Condition	**Value**
Timeout	> 5 minutes
Success offset	< 0.5 meters
Failure offset	> 22 meters

The next Chapter summarizes the results obtained for each proposed experimental scenario.

5
Results and Discussion

This chapter aims to provide a comprehensive description of the results obtained through the execution of the experiments proposed and detailed in the preceding chapters.

The auto-docking experiments are divided into six distinct scenarios, each representing a unique configuration and set of challenges. These scenarios are carefully designed to assess various aspects of the auto-docking system, particularly focused on the AUV's pose estimation and control challenges.

The primary goal of these experiments is to achieve successful and efficient docking of the AUV under different conditions. To gauge this, the first six subsections of this chapter showcase the successful cases along with the associated metrics for each corresponding scenario, detailing the AUV's behavior throughout all episodes.

Subsequently, an evaluation of the commonly successful episodes is conducted to compare the scenarios and determine if one outperforms the others, while discussing potential underlying reasons for such performance.

The concluding section consolidates an evaluation focused on the performance of the RL agent as compared to the conventional PID control chain.

5.1
Scenario 01

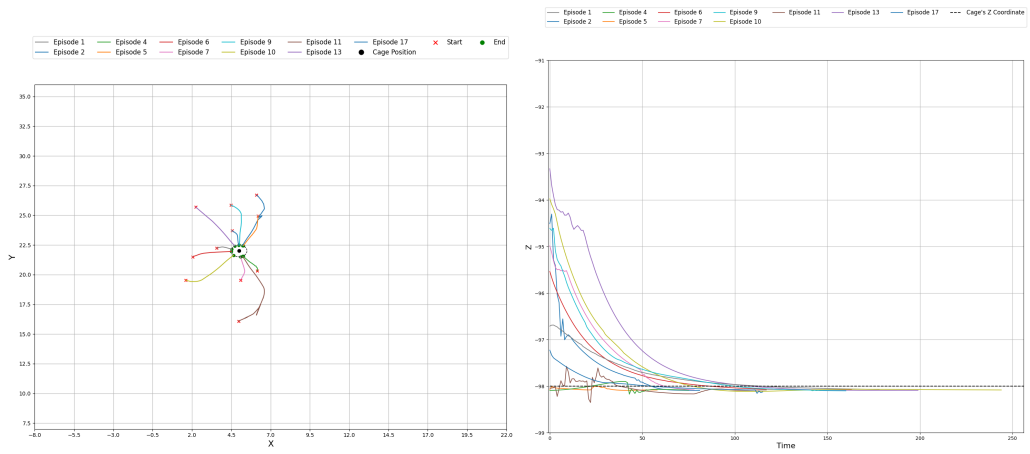
The first scenario focuses on the pose estimation carried out via the fiducial-based approach, coupled with control logic executed by the proposed conventional PID Controller, both elaborated in Section 3.1. As an initial data point, Table 5.1 aggregates the counts of failure and success cases, along with the variety of failure modes observed across all episodes. This provides insight into the robustness and reliability of the combination of fiducial-based pose estimation and PID control in tackling the auto-docking task.

Table 5.1: Episode success and failure metrics for Scenario 01.

Success Count	**Failure Count**	**Success Rate**	**Not Converged**	**Timeout**
11	15	42.31%	5	10

Based on the information provided in Table 5.1, it's evident that in most failure cases, the AUV actively sought to reach the target, specifically in 10 instances. These episodes mostly correspond to situations where the initial pose of the AUV was at a significant distance from the cage, rendering fiducial detection unfeasible. In 5 of these episodes, the AUV did manage to detect the cage but failed to achieve convergence to the target position. Overall, the system achieved successful convergence in fewer than 50% of the cases. This performance metric indicates that the solution is neither robust nor well-generalized for the problem space.

Figure 5.1(a) illustrates the 2D traveled path presented by the AUV for the 11 successful episodes.



(a) XY projection. (b) Z coordinate variation over the time.

Figure 5.1: Traveled paths performed during Scenario 01 experiments considering only succeeded cases.

As depicted in the Figures above, using Fiducial algorithms in the auto-docking system resulted in successful episodes when the distance to the cage was within 3 meters. However, it was observed that this solution faced challenges when dealing with tags that did not offer a clear view of their landmarks. Consequently, the lack of capability of Fiducial algorithms to process such obscured tags contributed to the system's limitations in certain scenarios.

In terms of the behavior presented related to the position variation along the Z axis, Figure 5.1(b) illustrates the obtained values.

For the Z-axis position control, the trajectory curve suggests that the controller maintained a smooth control profile, effectively keeping the AUV near the target altitude without significant perturbations. This implies a stable and precise Z-axis control mechanism, potentially signifying that the PID controller successfully mitigated altitude-based errors in this scenario.

5.2 Scenario 02

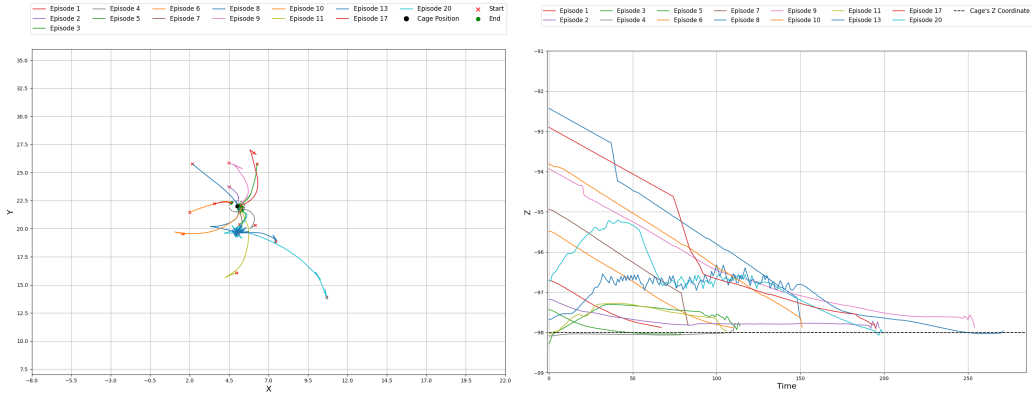
The second scenario focuses on the pose estimation performed by the fiducial-based approach coupled with the RL controller agent. Table 5.2 aggregates the counts of failure and success cases, along with the variety of failure modes observed across all episodes.

Table 5.2: Episode success and failure metrics for Scenario 02.

Success Count	Failure Count	Success Rate	Not Converged	Timeout
14	12	53.85%	4	8

As noticed in Table 5.2, once again, most of the failures were due to timeout, specifically in 8 instances. These episodes mostly correspond to situations where the initial pose of the AUV was at a significant distance from the cage, where fiducial detection is not feasible. In 4 of these episodes, the AUV did manage to detect the cage, since it moved, but failed to achieve convergence to the target position. Overall, the system achieved successful convergence higher than 50% of the cases. This performance metric indicates a better performance when compared to Scenario 01's. However, it can not be considered a robust solution since representative episodes have failed.

Figure 5.2(a) illustrates the 2D traveled path presented by the AUV for the 14 successful episodes.



(a) XY projection. (b) Z coordinate variation over the time.

Figure 5.2: Traveled paths performed during Scenario 02 experiments considering only succeeded cases.

As in the previous scenario, the fiducial-based pose estimation method performs better when the initial pose is in front of the cage, within a range where the fiducial markers can be identified.

However, a unique behavior emerges for episodes where the AUV begins on the cage's right-hand side. This pattern can be attributed to a limitation in the pose estimation algorithm, which struggles to generate accurate bundle pose estimates when relying solely on a single fiducial marker. Compounding the issue is the fact that the RL agent's control loop exhibits a faster response rate, which leads to frequent changes in the fiducial bundle reference. Consequently, this results in erroneous pose estimates.

To rectify this specific issue, one viable approach is to increase the density of fiducial markers on the cage's right side. By doing so, we can enrich the sensor data available for pose estimation, thereby enhancing both the robustness and accuracy of the system's spatial awareness.

In terms of the behavior presented related to the position variation along the Z axis, Figure 5.2(b) illustrates the obtained values.

As depicted in Figure 5.2(b), the RL agent exhibits a more aggressive control strategy. Notably, it reaches the target setpoint in a shorter duration by demanding higher thrust output from the AUV's propulsion system. This suggests that the RL agent optimizes the control actions to minimize time-to-target while leveraging the available thruster capacity more effectively.

5.3 Scenario 03

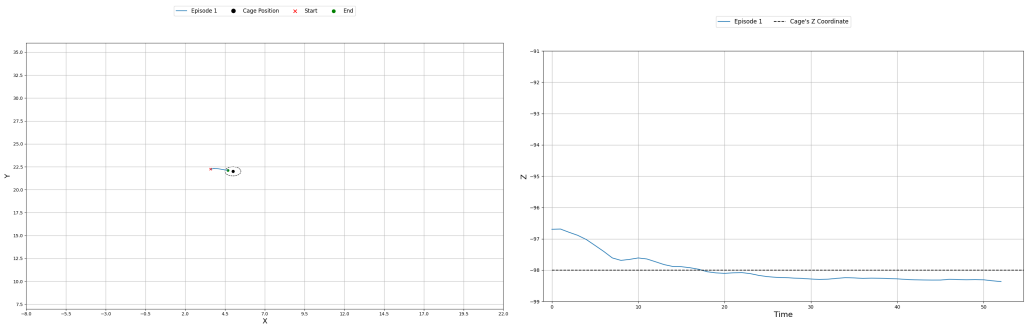
The third scenario arranges the CNN-based pose estimator and the PID controller to handle the auto-docking. Table 5.3 collects the counts of failure and success cases, along with the variety of failure modes observed across all episodes.

Table 5.3: Episode success and failure metrics for Scenario 03.

Success Count	**Failure Count**	**Success Rate**	**Not Converged**	**Timeout**
1	25	3.85%	0	26

As indicated in Table 5.3, the cage pose estimator struggled to converge to an optimal solution. Interestingly, the only episode that succeeded was the first one, which was also the closest to the target pose. It's crucial to note, however, that all instances of failure were attributed to timeouts rather than boundary violations. This implies that, despite the lack of convergence, the AUV managed to maintain its pose within the predefined spatial boundaries throughout the test scenarios.

Figure 5.3(a) describes this experiment's performance throughout the execution of the successful cases.



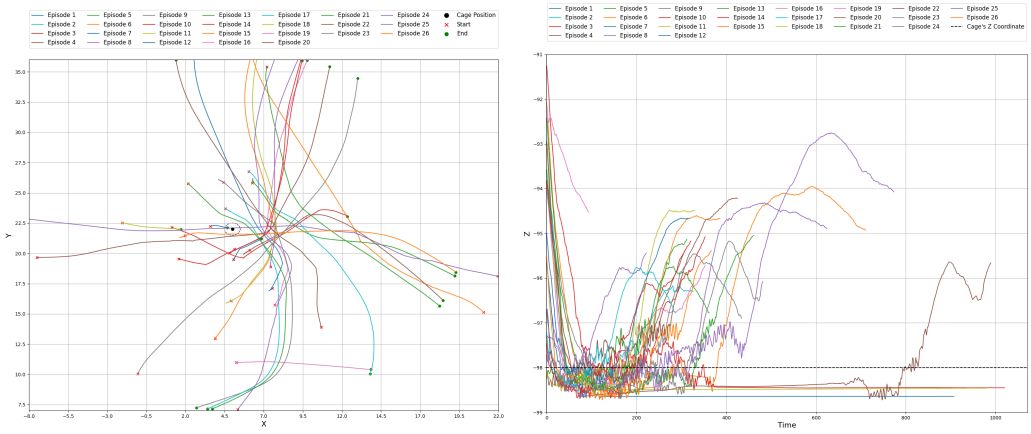
(a) XY projection. (b) Z coordinate variation over the time.

Figure 5.3: Traveled paths performed during Scenario 03 experiments considering only succeeded cases.

In terms of the behavior presented related to the position variation along the Z axis, Figure 5.3(b) illustrates the obtained values.

As observed, the auto-docking system exhibited a limited convergence rate, with only the first episode successfully reaching the target point. This behavior is primarily attributed to the lack of accuracy in the estimation model. The CNN algorithms utilized in the system necessitate a substantial amount of training images to develop a robust model capable of accurately estimating the object pose, even in challenging and hazardous environments.

Notwithstanding the convergence limitations, the auto-docking system displayed a consistent pattern in its behavior, demonstrating a tendency to approach the target point, albeit without sufficient precision to finalize the approximation adequately. Figure 5.4 illustrate this pattern, which indicates that the system possesses some capability to guide the AUV towards the cage. However, the lack of precision in pose estimation prevents the system from achieving successful docking with the desired level of accuracy. Further refinement and extensive training of the CNN algorithms may be necessary to enhance the system's performance and achieve reliable and precise auto-docking capabilities.



(a) XY projection. (b) Z coordinate variation over the time.

Figure 5.4: Traveled paths performed during Scenario 03 experiments considering all episodes.

As observed, the AUV consistently employed a heuristic to navigate toward the cage in all episodes. Despite its efforts, the movements executed were insufficient to achieve precise convergence with the target point. Additionally, the AUV demonstrated a distinct behavior when controlling the Z axis, exhibiting a notable amount of noise throughout most episodes.

The heuristic-based approach used by the AUV to approach the cage indicates its capability to initiate the docking process. However, the lack of accurate pose estimation and control algorithms hindered the system from achieving a successful and precise docking outcome. Furthermore, the noise observed in the Z-axis control indicates a potential issue in the control system's stability. The inconsistency in controlling the vertical position could be attributed to lack of variation during the model training; a more robust dataset would be required.

5.4 Scenario 04

Scenario 04 connects the CNN-based pose estimator and the RL-based controller. Table 5.4 collects the counts of failure and success cases, along with the variety of failure modes observed across all episodes.

Table 5.4: Episode success and failure metrics for Scenario 04.

Success Count	Failure Count	Success Rate	Not Converged	Timeout
1	25	3.85%	0	26

As Table 5.4 illustrates, the behavior presented in the previous scenario is similar. The CNN pose estimator remains the same, and the final results

as well. In order to ensure that, Figure 5.5 illustrates the 2D traveled path obtained for all episodes.

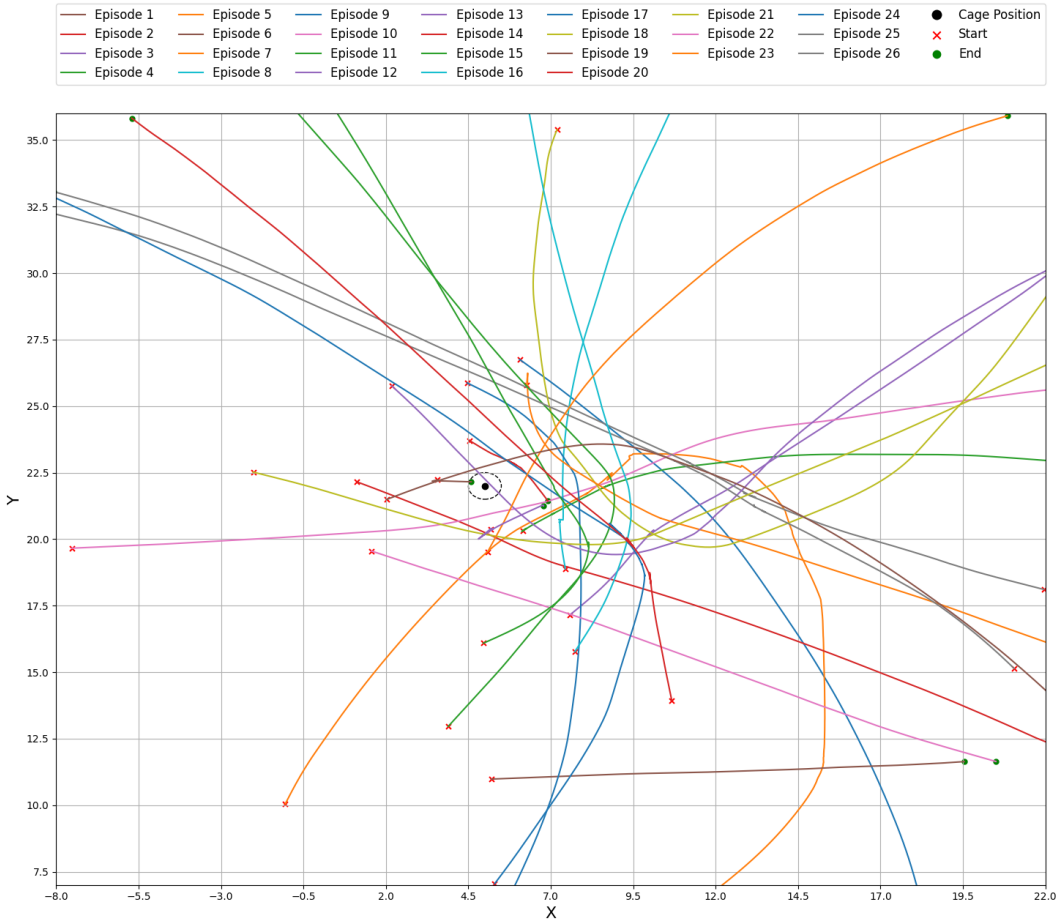


Figure 5.5: Z coordinate variation over the time for Scenario 04, considering all cases.

Evidently, the behavior closely mirrors what is presented in Figure 5.4. The main difference relies on the trajectory policy adopted by the controllers. The RL agent adopted a smoother policy when compared to the PID's. Consequently, the CNN-based estimator, when operating in isolation, failed to achieve convergence. Given that these results are consistent with those observed in the previous scenario, we have chosen to consider the performance metrics from the earlier scenario as applicable to this one as well.

5.5 Scenario 05

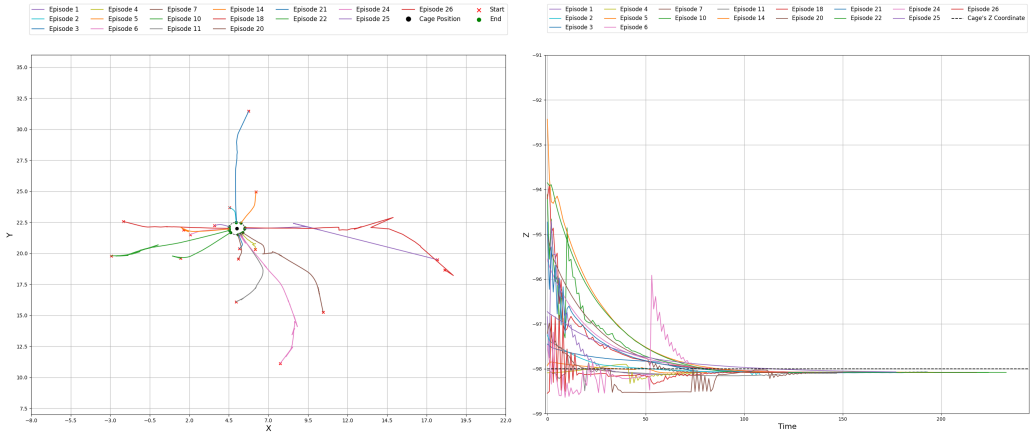
In Scenario 05, the combined version of the CNN and fiducial-based estimator is considered. For this instance, the PID controller is the evaluated solution to drive the AUV. Table 5.5 collects the counts of failure and success cases, along with the variety of failure modes observed across all episodes.

Table 5.5: Episode success and failure metrics for Scenario 05.

Success Count	**Failure Count**	**Success Rate**	**Not Converged**	**Timeout**
17	9	65.38%	0	9

As highlighted in Table 5.5, the AUV's performance in this scenario is notably superior, registering a success rate of 65.38%. Remarkably, the AUV maintained its position within the designated boundaries throughout each episode, including those where it began at a considerable distance from the cage. This demonstrates a marked improvement over the behaviors observed in Scenarios 01 and 02.

Figure 5.6 illustrates the 2D traveled path presented by the AUV for the 17 successful episodes.



(a) XY projection. (b) Z coordinate variation over the time.

Figure 5.6: Traveled paths performed during Scenario 05 experiments considering only succeeded cases.

As depicted in Figure 5.6(a), the AUV successfully navigated to the target pose even in challenging conditions, specifically during episodes 25 and 26, where the initial poses were 17 meters far from the target. The trajectories closely mirror those observed in Scenario 01. This suggests that the CNN-based estimation effectively took over the navigation tasks when fiducial markers were not within a detectable range, seamlessly complementing the fiducial-based pose estimation.

In terms of the behavior presented related to the position variation along the Z axis, Figure 5.6(b) illustrates the obtained values.

For the Z-axis control outputs, the trajectory exhibited noticeable disturbances compared to the smoother curve trend observed in Scenario 01,

where the fiducial-based estimator was employed. The behavior seems to incorporate characteristics from both Scenario 01 and Scenario 02, manifesting as short-term variations attributed to the CNN-based estimator. Importantly, the magnitude of these disturbances is less pronounced than in previous scenarios and falls within acceptable limits for this particular experiment.

5.6 Scenario 06

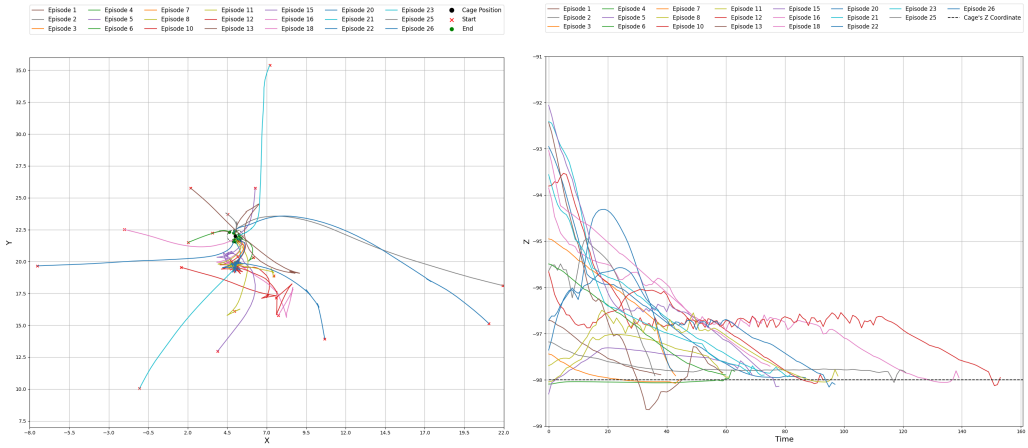
Scenario 06 connects the fiducial-based and CNN-based pose estimators along with the RL agent controller. Table 5.5 collects the counts of failure and success cases, along with the variety of failure modes observed across all episodes.

Table 5.6: Episode success and failure metrics for Scenario 06.

Success Count	**Failure Count**	**Success Rate**	**Not Converged**	**Timeout**
17	9	65.38%	0	9

As indicated in Table 5.6, this experiment yielded similar statistics for both the number of successful episodes and the types of failure modes compared to prior experiments. This suggests that the robustness of the solution is not solely dependent on the controller component but is tied to the estimator's performance. Nonetheless, the trajectory by which the AUV reaches its final pose serves as a distinct differentiating factor between this and previous experiments.

Figure 5.7(a) illustrates the 2D traveled path presented by the AUV for the 17 successful episodes.



(a) XY projection. (b) Z coordinate variation over the time.

Figure 5.7: Traveled paths performed during Scenario 06 experiments considering only succeeded cases.

As shown in Figure 5.7(a), the trajectory performance did not measure up to that of the previous scenario. The primary drawback is evident in the difficulty encountered when the episode started off from the right side of the cage, an issue previously discussed in Section 5.2 for Scenario 02. On a brighter note, the controller executed smoother trajectories for other episodes, specifically episodes 20, 21, 22, 25, and 26. Enhancements could likely be achieved in this scenario by incorporating additional fiducial markers.

In terms of the behavior presented related to the position variation along the Z axis, Figure 5.7(b) illustrates the obtained values.

In the context of the Z-axis control outputs, the behavior closely mimics that observed in Scenario 02. Generally speaking, the RL agent demonstrates a more refined control policy for Z-axis variations. It not only reaches the setpoint swiftly but also maintains it consistently across most episodes.

5.7 Scenarios Comparison

The objective of this section is to evaluate the scenarios by comparing and discussing metrics specifically for the successful cases. Table 5.7 compiles the successful episodes common to Scenarios 01, 02, 05, and 06.

Table 5.7: Common successful episodes presented for Scenarios 01, 02, 05, and 06.

Episode ID
1
2
4
5
6
7
10
11

In an effort to concentrate on scenarios that yielded meaningful data, Scenarios 03 and 04 are excluded from this assessment, given their failure to converge in 25 out of 26 instances.

5.7.1 Success Rate

The first metric up for evaluation is the success rate, which provides a holistic view of performance across the experiment, rather than focusing on specific episodes. Table 5.8 aggregates the success rates for the target scenarios under consideration.

Table 5.8: Success rates presented for Scenarios 01, 02, 05, and 06.

Scenario 01	**Scenario 02**	**Scenario 05**	**Scenario 06**
42.31%	53.85%	**65.38%**	**65.38%**

Based on the information in Table 5.8, it's evident that the Combined estimator method holds a distinct advantage, registering the highest success rates irrespective of the controller type used. This suggests that relying solely on a fiducial-based estimator—without any supplemental mechanisms for long-range cage detection—is suboptimal. Furthermore, the RL controller offers a slight boost in AUV performance when fiducial-based estimators are used. This improvement is likely attributed to the RL agent's rapid response capabilities, which help the AUV maintain a lock on the fiducial markers. A slower controller response might risk veering the AUV off course, potentially causing it to lose sight of the cage and hence of the fiducial markers.

5.7.2 Accumulative Error

As detailed in Chapter 3, accumulative error serves as a relevant metric to evaluate the performance of estimators and controllers over time. Specifically, this metric quantifies the sum of discrepancies between the desired setpoints and the actual states (e.g., position, orientation) of the AUV as it navigates through different scenarios [12]. If the accumulative error grows non-linearly over time, it might indicate system instability or a poor control strategy that fails to correct errors effectively. Therefore, in the context of this work, smaller values mean a better control policy. Table 5.9 gathers the accumulative error values for each episode assessed in this Section.

Table 5.9: Accumulative error values presented for the successful episodes for Scenarios 01, 02, 05, and 06.

Episode ID	Scenario 01	Scenario 02	Scenario 05	Scenario 06
1	167.82	74.50	163.94	**42.00**
2	**120.46**	250.46	116.38	150.45
4	116.81	205.13	115.33	**112.29**
5	**90.30**	237.62	143.30	138.99
6	228.60	208.32	230.07	**113.19**
7	142.34	239.31	133.15	**116.48**
10	451.69	533.55	471.92	**288.01**
11	338.31	359.91	408.74	**215.54**

From the results presented in Table 5.9, the following aspects can noticed:

- **Scenario 01 (PID + Fiducial-based Estimation):** The PID controller aims to minimize heading error, and it performs well in scenarios where the AUV's initial position allows for easy fiducial marker detection. Since heading alignment is more straightforward and quicker to achieve than precise position control, this setup offers advantages in specific circumstances.
- **Scenario 06 (RL + Combined Estimation):** The RL agent seems to excel in position error correction, partly because the Combined estimator offers a richer data set for decision-making. This strategy is particularly effective for a broader range of starting positions and orientations relative to the cage, giving it a more versatile edge.
- **Estimator Contributions**: The Combined estimator tends to provide more accurate and versatile estimates, offering the RL agent a more ro-

bust data source for decision-making. In contrast, the fiducial-based estimator excels in scenarios where the AUV's initial position is conducive to marker detection, which tends to be when the AUV is head-on with the cage.

- **PID vs. RL**: Both controllers have their strong suits depending on the scenario. PID shows comparable performance to the RL agent in cases where the fiducial estimator can fully capitalize on the AUV's initial orientation, i.e., episodes 2, 4, and 5.
- **Versatility and Adaptability:** While the PID controller shows competitive performance in specific conditions, the RL agent seems to be more adaptable across a range of scenarios, especially when paired with a versatile estimator like the Combined one.

5.7.3 Convergence Time

In this study, the convergence time is considered as a metric, which requires the AUV to not only reach the target position but also stabilize within a spatial margin of 0.5 meters around the docking point. It's worth noting that we have consciously excluded heading tolerance from this metric. The objective behind this decision is to assess how different controllers prioritize the controlled variables, such as position and orientation, in their strategies. Table 5.10 represents the convergence time values, in seconds, for each episode assessed in this Section.

Table 5.10: Convergence time values presented for the successful episodes for Scenarios 01, 02, 05, and 06.

Episode ID	Scenario 01	Scenario 02	Scenario 05	Scenario 06
1	37.78	**13.54**	37.83	14.16
2	55.18	**50.45**	56.30	54.81
4	87.19	**35.15**	90.97	35.22
5	88.79	69.32	88.65	**54.22**
6	66.38	**22.14**	66.69	22.52
7	80.33	**23.99**	79.37	27.75
10	99.11	**38.53**	90.91	127.95
11	137.57	**44.15**	133.18	44.57

The data in Table 5.10 provides a comprehensive view of the convergence time across the successful episodes for Scenarios 01, 02, 05, and 06. It's

immediately apparent that Scenario 02 consistently achieves the fastest convergence times in most episodes, indicating that the RL agent control strategy is highly efficient in guiding the AUV to the target within the shortest period. Scenario 06 also shows competitive timings but lags slightly behind Scenario 02. The noted variations can be attributed to the current lack of a decision-making mechanism for switching between the CNN-based and fiducial-based estimators, present in Scenario 02. While each estimator excels in its own domain—fiducial-based being more precise but range-limited, and the Combined estimator capable of working at greater distances.

Interestingly, Scenario 05 doesn't significantly improve over Scenario 01 in terms of time-to-convergence, despite the addition of a more sophisticated estimator. This suggests that the gains in estimator accuracy do not necessarily translate into faster convergence, which could be a subject for future investigation. Scenario 01, being the baseline, generally takes the longest time to converge, reaffirming the benefits of advanced control and estimation strategies implemented in the other scenarios.

It is also worth noting the peculiar case of Episode 10 in Scenario 06, where the AUV took an unusually long time to converge, skewing the general trend. This outlier indicates that even advanced control schemes like RL can sometimes lead to non-optimal behaviors, warranting further research to fine-tune the controller's policy.

5.7.4 Heading Error

As detailed in Chapter 3, the heading error serves as a relevant evaluation metric for assessing the performance and reliability of the proposed methods to perform the auto-docking task. The heading error is the angular difference between the AUV's current orientation and the desired target orientation. In this work, heading error is quantified in degrees, a unit that lends itself to more intuitive understanding and interpretation. This may facilitate a clearer comprehension of the system's performance, especially for readers who may not be deeply versed in control theory. Table 5.10 represents the heading error values, in degrees, for each episode assessed in this Section.

Table 5.11: Heading error values presented for the successful episodes for Scenarios 01, 02, 05, and 06.

Episode ID	**Scenario 01**	**Scenario 02**	**Scenario 05**	**Scenario 06**
1	**0.29**	8.83	0.31	-108.64
2	0.43	-66.43	**0.11**	10.01
4	**0.04**	-157.89	0.42	-164.55
5	**0.27**	-57.27	0.31	-48.00
6	**-0.23**	-105.79	-0.23	-135.85
7	0.10	40.22	**-0.08**	-65.57
10	0.42	-140.34	**-0.14**	1.06
11	**-0.03**	-124.63	0.43	13.27

The data presented in Table 5.11 offers insights into the heading control performance across different scenarios. At first glance, Scenario 01 consistently outperforms the others, presenting minimal heading errors across multiple episodes. Scenario 05 follows closely, whereas Scenarios 02 and 06 present significantly higher heading errors, sometimes veering into triple-digit angles away from the target.

The anomalous behavior in Scenario 06, which employs the RL agent for control, can be attributed to the inadequacy of the reward function in emphasizing heading alignment. Essentially, the RL agent's reward function seems more optimized for other metrics and does not sufficiently penalize heading errors. This suggests that future iterations of the RL agent should include a more robust reward function that strategically weighs the importance of heading error. Doing so could bring about a more balanced and precise navigational strategy, improving the AUV's overall orientation alignment with the target.

From another perspective, by incorporating additional information about the heading into the TD3 critic could enhance the RL agent's performance in maintaining proper orientation. The TD3 algorithm already uses two critic networks to estimate the Q-values, which are essential in policy optimization. By feeding the heading error as an additional input feature to these critics, the model could gain a more nuanced understanding of the environment and adjust its action policies accordingly [71].

5.8 RL vs PID

In this Section, the focus shifts to evaluating the pure performance of the proposed control algorithms, PID and RL, by using ground truth pose data for the AUV. This approach allows us to isolate the performance of the controllers from the limitations or inaccuracies of the estimators used in previous experiments. By removing the estimator variable from the equation, we can directly assess how well each control methodology adapts to the auto-docking task and discern the advantages that RL potentially brings to the table over traditional PID control.

5.8.1 Trajectory

Certainly, the first aspect under discussion pertains to the smoothness exhibited by the trajectories as proposed by each respective controller. Figures 5.8 and 5.9 visually illustrate the 2D paths traversed by the controllers under consideration. The contour and fluidity of these paths serve as significant indicators for the performance quality of the control algorithms. Specifically, a smooth trajectory is often suggestive of a well-tuned control system, which is paramount for tasks that require high precision, such as auto-docking in underwater robotics.

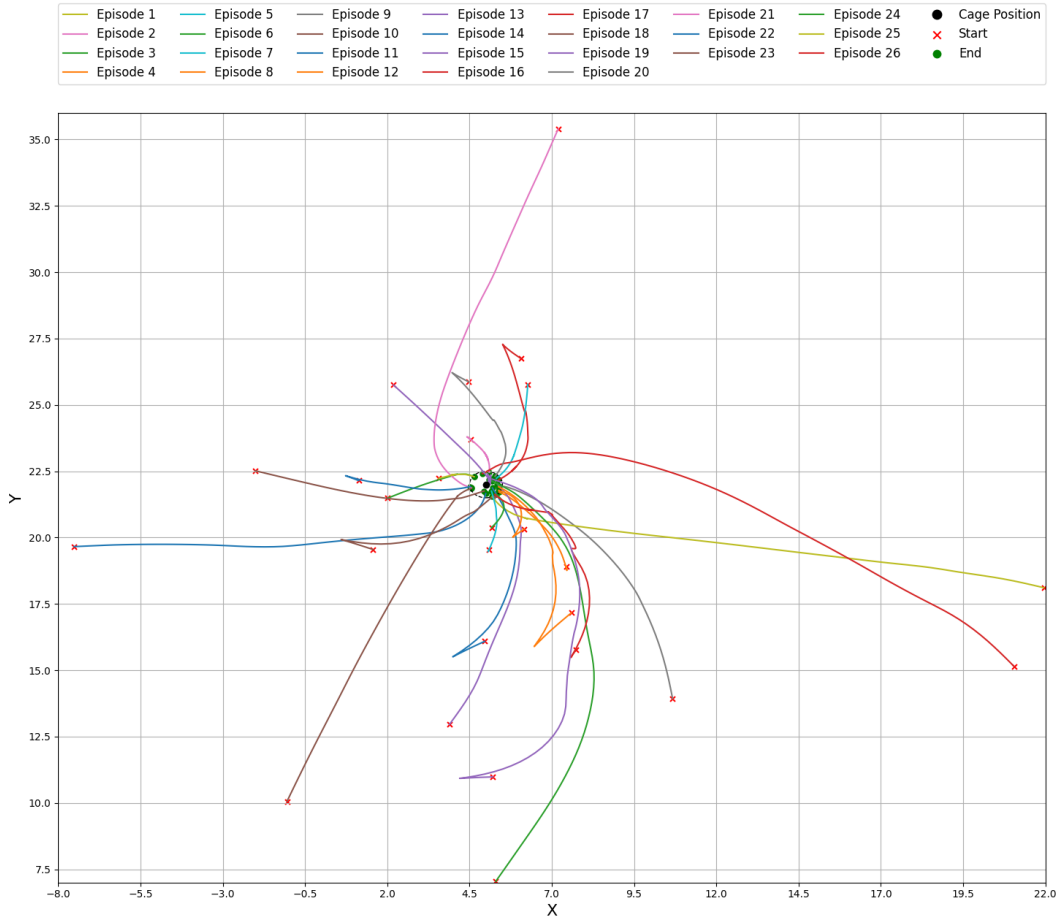


Figure 5.8: Trajectory presented by RL-based controller with the cage pose estimation ground truth as input.

The PID-controlled trajectory tends to be more direct, optimizing for efficiency both in terms of computational load and power consumption. This kind of straight-to-target behavior is generally prized in robotics applications, where resource optimization, be it battery life or computational cycles, is crucial [52].

On the other hand, the RL-based approach has the advantage of not requiring prior knowledge of the system dynamics. This feature becomes increasingly valuable in more complex tasks that involve dynamic environments, such as path planning with obstacle avoidance or Simultaneous Localization and Mapping (SLAM). While RL methods may have a higher upfront computational cost for training, they could offer superior performance and adaptability in scenarios where the system dynamics are either poorly understood or subject to change.

Thus, while PID may offer a more straightforward and resource-efficient solution for the auto-docking task, RL-based methods might provide more robust and versatile solutions for tasks that have additional complexities or uncertainties.

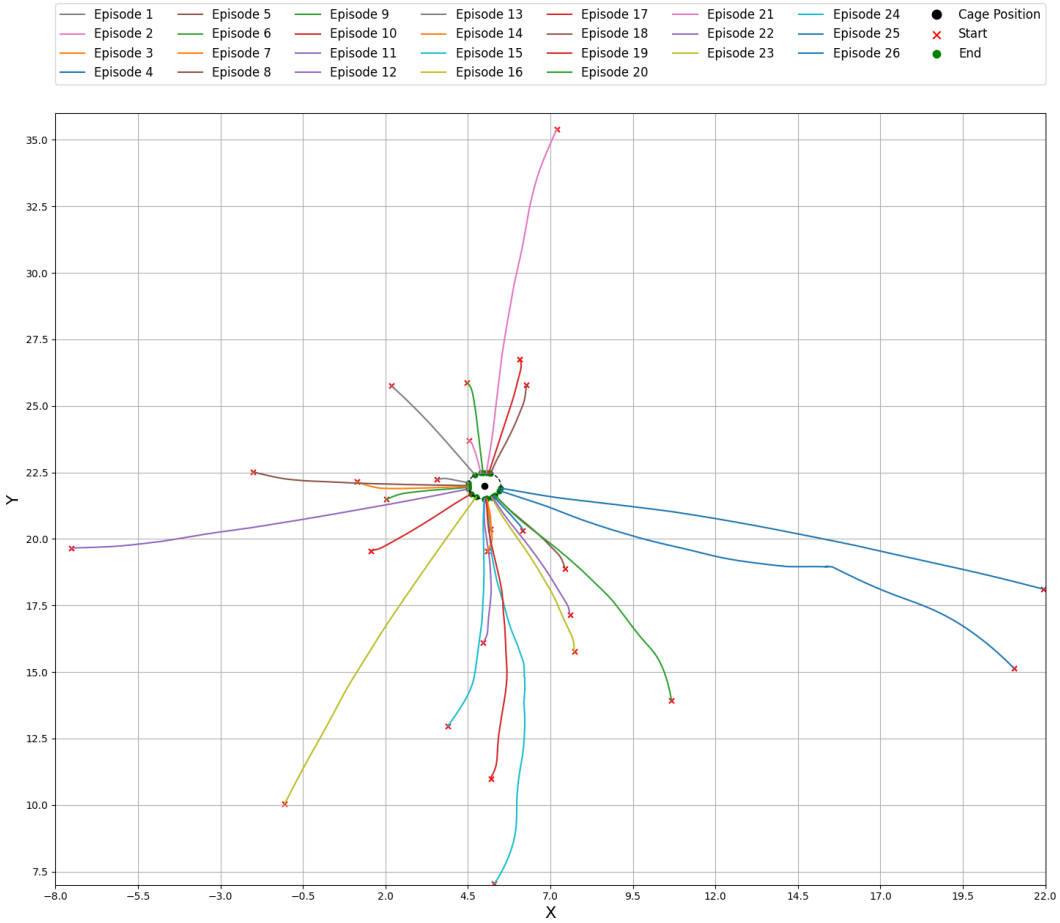


Figure 5.9: Trajectory presented by PID-based controller with the cage pose estimation ground truth as input.

From the result presented in Figure 5.9, an intriguing observation is made regarding the more "natural maneuvers" exhibited by the RL agent. The inherent flexibility of reinforcement learning allows the agent to adapt to complex and dynamic environments. This adaptability can result in nuanced behaviors, particularly valuable in unstructured underwater scenarios, which present their own unique challenges.

However, this adaptability also manifests as suboptimal performance in simpler, obstacle-free environments, as observed in the experiments. The generalized hyperparameters provided by the Stable Baselines library may not be finely tuned for the specific requirements of the auto-docking task at hand.

This brings to light a classic trade-off in machine learning: the ease of implementation versus the specificity of control over the algorithm's parameters. Utilizing out-of-the-box solutions like Stable Baselines often necessitates sacrificing some degree of control for the sake of rapid prototyping and implementation ease.

5.8.2 Accumulative Error

The discrepancy in the accumulative error between the PID-based and the other controller, however, raises questions about the efficiency of the PID controller in balancing between orientation and positional error. This could be due to the tuning of the PID gains, where perhaps they aren't optimized for this specific application. In a more complex environment like underwater robotics, simple PID might not cut it when dealing with multidimensional error spaces involving both position and orientation. Figure 5.10 depicts the curves associated to each method.

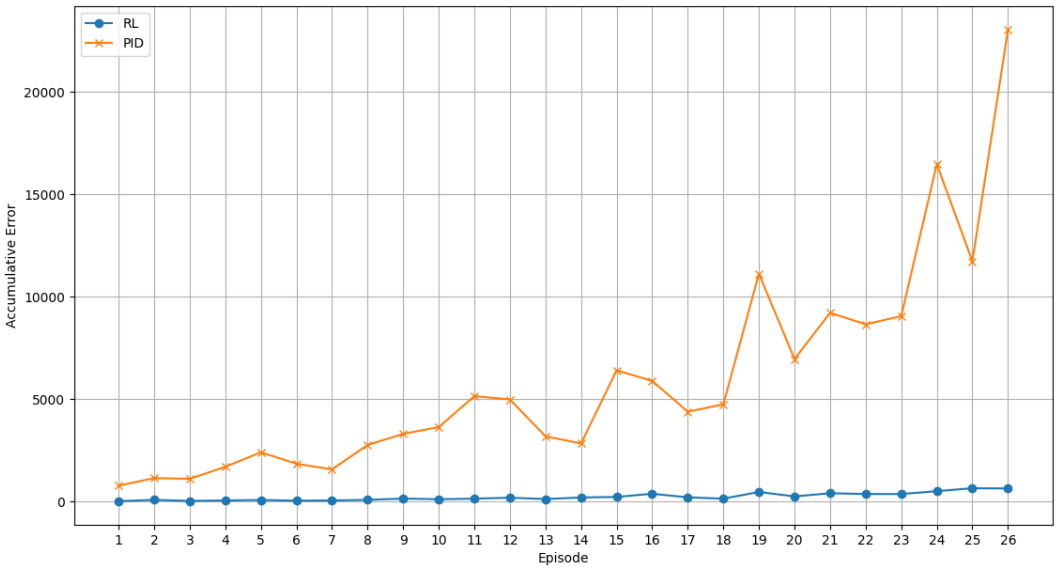


Figure 5.10: Convergence time obtained by the two solutions. RL performance is represented by the blue curve and the PID's by the orange one.

The fact that the PID-based controller has higher accumulative errors could suggest that it is less robust to certain disturbances or model inaccuracies. It might also indicate that the controller isn't as effective at managing the coupling between orientation and position, which is often a crucial aspect that touches the auto-docking task. Additionally, the strategy adopted by the RL agent discussed before and denoted as a "natural maneuver" can be related to this trade-off. To keep the orientation close to the aimed one.

5.8.3 Convergence Time

As detailed previously, the convergence time is a relevant metric to evaluate the controllers' performance. Figure 5.11 illustrates the convergence time values obtained for each method.

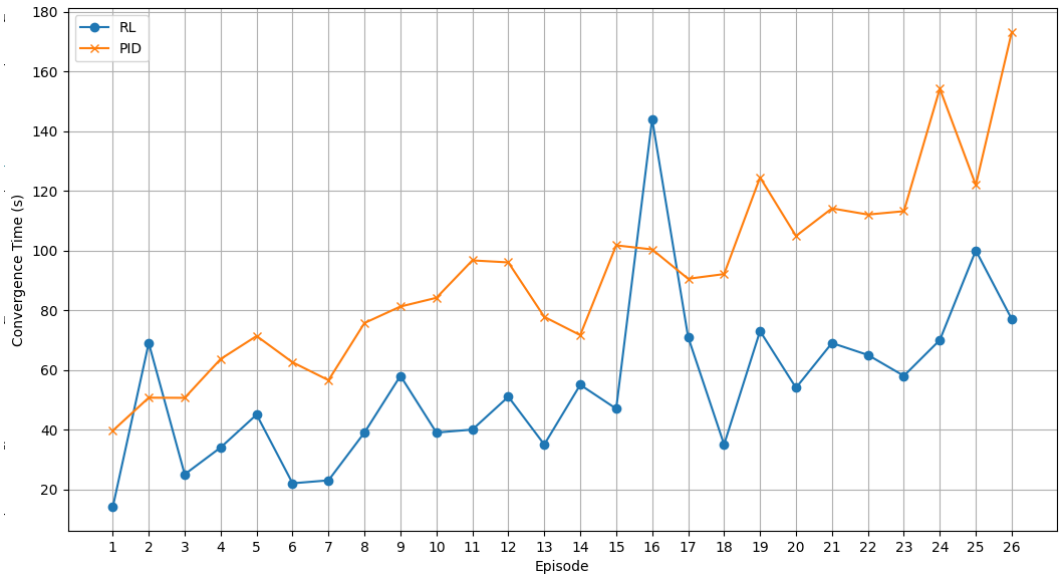


Figure 5.11: Convergence time obtained by the two solutions. RL performance is represented by the blue curve and the PID's by the orange one.

Naturally, the PID-based controller's inherent design to minimize error in a straightforward manner often results in quicker convergence times. This is generally an expected outcome, given that the PID control loop aims to directly minimize the positional and orientation errors to bring the AUV to its target state.

However, it's intriguing to note the specific episodes where the RL agent outperformed the PID controller in terms of convergence time, such as episodes 16 and 2. This suggests that the RL agent has the potential to adapt its strategy for more efficient task completion under certain conditions. In these episodes, the RL agent seems to have found a policy that allowed it to go straight toward the goal, thereby reducing the overall time required to complete the task.

This provides valuable insights for further development. For instance, it might be beneficial to investigate the specific states and actions taken during these episodes to understand what led the RL agent to adopt a more direct path.

5.8.4 Heading Error

The heading error is also evaluated considering the absence of the estimators. Table 5.12 gathers the result presented by the two methods.

Table 5.12: Heading errors presented by RL and PID based controllers, in degree.

Episode ID	**RL**	**PID**
1	142.01	**2.44**
2	-70.80	**11.04**
3	-55.22	**2.57**
4	113.52	**36.76**
5	**-16.51**	-17.75
6	-157.93	**-22.66**
7	78.14	**3.19**
8	-48.99	**18.00**
9	-121.35	**12.68**
10	-126.31	**20.73**
11	**-18.46**	-25.71
12	-9.49	**-7.88**
13	-94.46	**14.81**
14	48.42	**2.52**
15	76.78	**-13.10**
16	**1.75**	-16.43
17	**7.20**	11.27
18	-127.31	**-3.38**
19	**33.05**	-54.73
20	56.37	**-49.40**
21	-67.38	**-16.78**
22	154.03	**-1.72**
23	96.81	**0.73**
24	12.17	**11.94**
25	-30.54	**-30.45**
26	-108.11	**1.27**

The heading error behavior, as shown in Table 5.12, reveals that the PID method is generally more effective at minimizing heading error across most episodes, as evidenced by the lower absolute values. It is particularly noteworthy that the PID method has more episodes with error magnitudes closer to zero, suggesting a more precise orientation control.

However, it is not universally better; there are instances where the RL method outperforms the PID-based control. This may indicate that the RL controller can be more robust or better tuned for specific scenarios or

edge cases. This divergence in performance could be a function of how well each algorithm adapts to varying conditions or unknown disturbances in the underwater environment.

When compared to prior behavior, if the RL method previously demonstrated higher heading errors similar to those currently displayed, this would indicate a consistent limitation of the RL approach in its current configuration. If, however, this is a deviation from earlier results, it may signify that changes to the system or the RL algorithm have adversely impacted its performance. Similarly, if the PID-based approach has maintained its low-error profile, this demonstrates the robustness and reliability of PID control for this application.

In sum, while the PID control generally exhibits superior performance in heading error reduction, the RL controller's occasional outperformance hints at the potential for further optimization or suitability under certain conditions. Therefore, both methods have merits and drawbacks, which can be considered for system-level trade-offs in your underwater robotic applications.

6
Conclusion

This research proposed fundamental experiments to understand better the auto-docking task performed by AUVs, with a spotlight on cage pose estimation and controlling methodologies. While the primary objective was to cultivate an efficient and robust system to enable AUVs to dock with underwater structures or targets with precision, the exploration was far-reaching. The research was conducted through various scenarios, each building on the other to evaluate the proposed system's efficacy. In the initial scenarios that used a fiducial-based approach for pose estimation paired with a traditional PID controller for visual servoing, the system demonstrated favorable outcomes with successful dockings up to a distance of 3 meters. However, the system faced limitations, especially when the fiducial markers were obscured.

To push the envelope further, later scenarios introduced a CNN-based estimator for pose estimation. The implementation of this novel addition eliminated the need for dataset labeling, streamlining the experimental workflow. Furthermore, the adoption of transfer learning techniques opens doors for leveraging other machine learning algorithms in addressing challenges specific to underwater environments. However, it was observed that a pure machine learning-based pose estimator was not successful, a finding that merits deeper investigation. Combining fiducial and CNN pose estimation did substantially elevate the system's capabilities.

Overall, the insights obtained from this research are help in comprehending the complexities of auto-docking systems for AUVs. It emphasizes that integrating precise pose estimation techniques is a critical factor for achieving reliable and high-precision auto-docking. Future exploration challenges should delve into refining pose estimation models, formulating resilient control strategies, and contending with environmental variances and sensor noises.

In conclusion, this research serves as a reference for future authors. By addressing specific challenges in navigation tasks, particularly in the nuanced areas of auto-docking and pose estimation, the study not only offers a solution framework but also uncovers lessons integral to the iterative process of research and development. These obtained insights form a knowledge base that can be useful for upcoming projects.

6.1 Future Work

While the RL controller has shown its potential, there is a clear avenue for further work in refining its algorithms, training methodologies, and feature engineering. More sophisticated RL techniques could be incorporated to improve performance and reliability. Specifically, while the RL algorithm in this work was trained with the ground-truth pose, training it with the estimated pose by CNN or fiducials could allow it to adapt to their specific strengths and weaknesses.

Additionally, considering an agent to perform the entire auto-docking task is one potential future challenge. As the RL agent proposed in this work presented a satisfactory performance, an end-to-end approach evaluation can be considered.

An end-to-end RL agent could offer a holistic, adaptable solution to manage the complex dynamics and uncertainties inherent in underwater environments. The aspiration to utilize an RL agent to handle the entire auto-docking task is an ambitious yet plausible future direction. This could dramatically simplify the system architecture by eliminating the need for hand-crafted algorithms or rules for specific sub-tasks. It would also potentially offer better adaptability to environmental changes or unexpected situations.

As Figure 6.1 outlines, the envisioned end-to-end RL model for auto-docking would likely involve a series of interconnected components, each contributing to the overall effectiveness and reliability of the system. Such a model might incorporate:

- **State Estimation**: To provide real-time awareness of the vehicle's state and the environment.
- **Control**: To generate the required control commands to most efficientlt and safely move towards the docking station.
- **Continuous Learning**: To adapt the agent's policies based on the outcomes of its actions and external feedback.
- **Obstacle Avoidance**: Obstacle avoidance is a relevant task covered by the auto-docking challenge. Due to its nature, the RL agent could be trained with a reward function that comprises the awareness of avoiding collision.

The promising results obtained so far with the simpler RL agent in this work provide a compelling basis for this future extension. Achieving this end-to-end capability would be a significant milestone in underwater robotics,

offering a level of autonomy and adaptability currently unattainable with traditional control methods.

Figure 6.1 illustrates the high-level workflow for this potential future task.

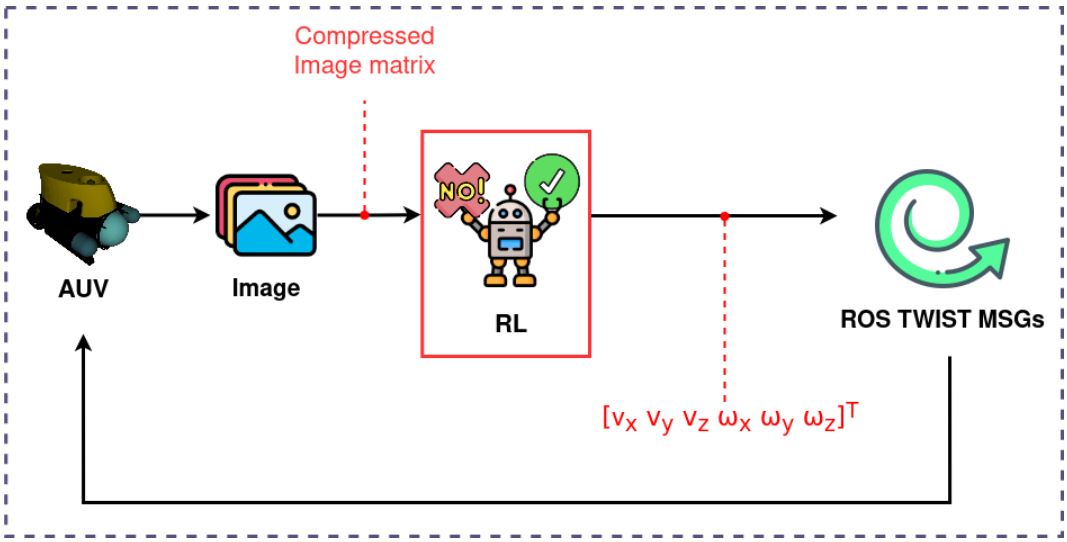


Figure 6.1: High-level workflow for an end-to-end application considering an Rl agent as the principal actor.

In addition to these focal points, other considerations for future work could be to fortify the algorithms against the uncertainties and vicissitudes of underwater conditions. Enhancing the simulator's rendering engine could better simulate real-world conditions, thereby facilitating more effective training of RL agents. As observed, increasing the number of fiducial markers could likely yield better system performance. Comparative evaluations involving diverse RL algorithms or hybrid models that marry the strengths of both RL and PID controls could also yield valuable insights. Ultimately, the litmus test for any algorithm's effectiveness would be its performance during real-world deployments since simulations, despite their sophistication, can only approximate the myriad complexities that define underwater environments.

Bibliography References

[1] HOCKSTEIN, N. G.; GOURIN, C.; FAUST, R. ; TERRIS, D. J.. **A history of robots: from science fiction to surgical robotics**. Journal of robotic surgery, 1(2):113–118, 2007.

[2] FAHIMI, F.. **Autonomous robots**. Modeling, Path Planning and Control, 2009.

[3] BLIDBERG, D.. **The development of autonomous underwater vehicles (auv); a brief summary**. 01 2001.

[4] HUVENNE, V. A.; ROBERT, K.; MARSH, L.; LO IACONO, C.; LE BAS, T. ; WYNN, R. B.. **Rovs and auvs**. In: SUBMARINE GEOMORPHOLOGY, p. 93–108. Springer, 2018.

[5] WHITCOMB, L. L.. **Underwater robotics: Out of the research laboratory and into the field**. In: PROCEEDINGS 2000 ICRA. MILLENNIUM CONFERENCE. IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION. SYMPOSIA PROCEEDINGS (CAT. NO. 00CH37065), volumen 1, p. 709–716. IEEE, 2000.

[6] GRIFFITHS, G.. **Technology and applications of autonomous underwater vehicles**, volumen 2. CRC Press, 2002.

[7] HOBSON, B. W.; BELLINGHAM, J. G.; KIEFT, B.; MCEWEN, R.; GODIN, M. ; ZHANG, Y.. **Tethys-class long range auvs-extending the endurance of propeller-driven cruising auvs from days to weeks**. In: 2012 IEEE/OES AUTONOMOUS UNDERWATER VEHICLES (AUV), p. 1–8. IEEE, 2012.

[8] FURLONG, M. E.; PAXTON, D.; STEVENSON, P.; PEBODY, M.; MCPHAIL, S. D. ; PERRETT, J.. **Autosub long range: A long range deep diving auv for ocean monitoring**. In: 2012 IEEE/OES AUTONOMOUS UNDERWATER VEHICLES (AUV), p. 1–7. IEEE, 2012.

[9] ALLOTTA, B.; CAITI, A.; COSTANZI, R.; FANELLI, F.; FENUCCI, D.; MELI, E. ; RIDOLFI, A.. **A new auv navigation system exploiting unscented kalman filter**. Ocean Engineering, 113:121–132, 2016.

[10] SANS-MUNTADAS, A.; KELASIDI, E.; PETTERSEN, K. Y. ; BREKKE, E.. **Learning an auv docking maneuver with a convolutional neural network**. IFAC Journal of Systems and Control, 8:100049, 2019.

[11] YAHYA, M. F.; ARSHAD, M. R.. **Position-based visual servoing for underwater docking of an autonomous underwater vehicle**. In: 2016 IEEE INTERNATIONAL CONFERENCE ON UNDERWATER SYSTEM TECHNOLOGY: THEORY AND APPLICATIONS (USYS), p. 121–126, 2016.

[12] TEO, K.; AN, E. ; BEAUJEAN, P. J.. **A robust fuzzy autonomous underwater vehicle (auv) docking approach for unknown current disturbances**. IEEE Journal of Oceanic Engineering, 37(2):143–155, 2012.

[13] CHRIST, R.; WERNLI SR, R.. **A user guide for remotely operated vehicles**, 2014.

[14] LARSEN, M. B.. **Autonomous navigation of underwater vehicles**. 2001.

[15] YAZDANI, A. M.; SAMMUT, K.; YAKIMENKO, O. ; LAMMAS, A.. **A survey of underwater docking guidance systems**. Robotics and Autonomous systems, 124:103382, 2020.

[16] KAWASAKI, T.; NOGUCHI, T.; FUKASAWA, T.; HAYASHI, S.; SHIBATA, Y.; OKAYA, N.; FUKUI, K.; KINOSHITA, M. ; OTHERS. **" marine bird", a new experimental auv-results of docking and electric power supply tests in sea trials**. In: OCEANS' 04 MTS/IEEE TECHNO-OCEAN'04 (IEEE CAT. NO. 04CH37600), volumen 3, p. 1738–1744. IEEE, 2004.

[17] KIMBALL, P. W.; CLARK, E. B.; SCULLY, M.; RICHMOND, K.; FLESHER, C.; LINDZEY, L. E.; HARMAN, J.; HUFFSTUTLER, K.; LAWRENCE, J.; LELIEVRE, S. ; OTHERS. **The artemis under-ice auv docking system**. Journal of field robotics, 35(2):299–308, 2018.

[18] LWIN, K. N.; MUKADA, N.; MYINT, M.; YAMADA, D.; MINAMI, M.; MATSUNO, T.; SAITOU, K. ; GODOU, W.. **Docking at pool and sea by using active marker in turbid and day/night environment**. Artificial Life and Robotics, 23:409–419, 2018.

[19] ALBIEZ, J.; JOYEUX, S.; GAUDIG, C.; HILLJEGERDES, J.; KROFFKE, S.; SCHOO, C.; ARNOLD, S.; MIMOSO, G.; ALCANTARA, P.; SABACK, R. ;

OTHERS. **Flatfish-a compact subsea-resident inspection auv**. In: OCEANS 2015-MTS/IEEE WASHINGTON, p. 1–8. IEEE, 2015.

[20] ALLEN, B.; AUSTIN, T.; FORRESTER, N.; GOLDSBOROUGH, R.; KUKULYA, A.; PACKARD, G.; PURCELL, M. ; STOKEY, R.. **Autonomous docking demonstrations with enhanced remus technology**. In: OCEANS 2006, p. 1–6. IEEE, 2006.

[21] MCEWEN, R. S.; HOBSON, B. W.; MCBRIDE, L. ; BELLINGHAM, J. G.. **Docking control system for a 54-cm-diameter (21-in) auv**. IEEE Journal of Oceanic Engineering, 33(4):550–562, 2008.

[22] WANG, T.; ZHAO, Q. ; YANG, C.. **Visual navigation and docking for a planar type auv docking and charging system**. Ocean Engineering, 224:108744, 2021.

[23] CARRERAS, M.; HERNÁNDEZ, J. D.; VIDAL, E.; PALOMERAS, N.; RIBAS, D. ; RIDAO, P.. **Sparus ii auv—a hovering vehicle for seabed inspection**. IEEE Journal of Oceanic Engineering, 43(2):344–355, 2018.

[24] SOLA, Y.. **Contributions to the development of deep reinforcement learning-based controllers for AUV**. PhD thesis, ENSTA Bretagne-École nationale supérieure de techniques avancées Bretagne, 2021.

[25] CHRISTENSEN, L.; DE GEA FERNÁNDEZ, J.; HILDEBRANDT, M.; KOCH, C. E. S. ; WEHBE, B.. **Recent advances in ai for navigation and control of underwater robots**. Current Robotics Reports, 3(4):165–175, 2022.

[26] SATO, Y.; MAKI, T.; MASUDA, K.; MATSUDA, T. ; SAKAMAKI, T.. **Autonomous docking of hovering type auv to seafloor charging station based on acoustic and visual sensing**. In: 2017 IEEE UNDERWATER TECHNOLOGY (UT), p. 1–6. IEEE, 2017.

[27] CHENG, W.; TEYMORIAN, A. Y.; MA, L.; CHENG, X.; LU, X. ; LU, Z.. **Underwater localization in sparse 3d acoustic sensor networks**. In: IEEE INFOCOM 2008-THE 27TH CONFERENCE ON COMPUTER COMMUNICATIONS, p. 236–240. IEEE, 2008.

[28] DIAMANT, R.; LAMPE, L.. **Underwater localization with time-synchronization and propagation speed uncertainties**. IEEE Transactions on Mobile Computing, 12(7):1257–1269, 2012.

[29] BRANCA, A.; STELLA, E. ; DISTANTE, A.. **Autonomous navigation of underwater vehicles**. In: IEEE OCEANIC ENGINEERING SOCIETY. OCEANS'98. CONFERENCE PROCEEDINGS (CAT. NO.98CH36259), volumen 1, p. 61–65 vol.1, 1998.

[30] DA COSTA BOTELHO, S. S.; DREWS, P.; OLIVEIRA, G. L. ; D. S. FIGUEIREDO, M.. **Visual odometry and mapping for underwater autonomous vehicles**. In: 2009 6TH LATIN AMERICAN ROBOTICS SYMPOSIUM (LARS 2009), p. 1–6, 2009.

[31] SCHETTINI, R.; CORCHS, S.. **Underwater image processing: state of the art of restoration and image enhancement methods**. EURASIP Journal on Advances in Signal Processing, 2010:1–14, 2010.

[32] BOUTHILLIER, X.; VAROQUAUX, G.. **Survey of machine-learning experimental methods at NeurIPS2019 and ICLR2020**. PhD thesis, Inria Saclay Ile de France, 2020.

[33] GEIGER, A.; LENZ, P. ; URTASUN, R.. **Are we ready for autonomous driving? the kitti vision benchmark suite**. In: 2012 IEEE CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, p. 3354–3361, 2012.

[34] GUTIERREZ, R. M.; YU, H.; RONG, Y. ; BLISS, D. W.. **Comparison of uas-to-ground small-scale fading in residential and mountainous desert terrains**. IEEE Transactions on Vehicular Technology, 68(10):9348–9358, 2019.

[35] MNIH, V.. **Machine learning for aerial image labeling**. Citeseer, 2013.

[36] ŽLAJPAH, L.. **Simulation in robotics**. Mathematics and Computers in Simulation, 79(4):879–897, 2008.

[37] MANHÃES, M. M. M.; SCHERER, S. A.; VOSS, M.; DOUAT, L. R. ; RAUSCHENBACH, T.. **Uuv simulator: A gazebo-based package for underwater intervention and multi-robot simulation**. In: OCEANS 2016 MTS/IEEE MONTEREY, p. 1–8. IEEE, 2016.

[38] TECHNOLOGIES, U.. **Unity**, 2023.

[39] DHURANDHER, S. K.; MISRA, S.; OBAIDAT, M. S. ; KHAIRWAL, S.. **Uwsim: A simulator for underwater sensor networks**. Simulation, 84(7):327–338, 2008.

[40] CORPORATION, N.. **Nvidia**, 2023.

[41] KONRAD, A.. **Simulation of mobile robots with unity and ros: A case-study and a comparison with gazebo**, 2019.

[42] KÖRBER, M.; LANGE, J.; REDISKE, S.; STEINMANN, S. ; GLÜCK, R.. **Comparing popular simulation environments in the scope of robotics and reinforcement learning**. arXiv preprint arXiv:2103.04616, 2021.

[43] KONRAD, A.. **Simulation of mobile robots with unity and ros: A case-study and a comparison with gazebo**, 2019.

[44] KÖRBER, M.; LANGE, J.; REDISKE, S.; STEINMANN, S. ; GLÜCK, R.. **Comparing popular simulation environments in the scope of robotics and reinforcement learning**. arXiv preprint arXiv:2103.04616, 2021.

[45] PLATT, J.; RICKS, K.. **Comparative analysis of ros-unity3d and ros-gazebo for mobile ground robot simulation**. Journal of Intelligent & Robotic Systems, 106(4):80, 2022.

[46] COLLINS, J.; CHAND, S.; VANDERKOP, A. ; HOWARD, D.. **A review of physics simulators for robotic applications**. IEEE Access, 9:51416–51431, 2021.

[47] RIVERA, Z. B.; DE SIMONE, M. C. ; GUIDA, D.. **Unmanned ground vehicle modelling in gazebo/ros-based environments**. Machines, 7(2), 2019.

[48] JALAL, F.; NASIR, F.. **Underwater navigation, localization and path planning for autonomous vehicles: A review**. In: 2021 INTERNATIONAL BHURBAN CONFERENCE ON APPLIED SCIENCES AND TECHNOLOGIES (IBCAST), p. 817–828. IEEE, 2021.

[49] KÖSER, K.; FRESE, U.. **Challenges in underwater visual navigation and slam**. AI technology for underwater robots, p. 125–135, 2020.

[50] RIVES, P.; BORRELLY, J.-J.. **Visual servoing techniques applied to an underwater vehicle**. In: PROCEEDINGS OF INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION, volumen 3, p. 1851–1856. IEEE, 1997.

[51] ESPIAU, B.; CHAUMETTE, F. ; RIVES, P.. **A new approach to visual servoing in robotics**. ieee Transactions on Robotics and Automation, 8(3):313–326, 1992.

[52] CHAUMETTE, F.; HUTCHINSON, S. ; CORKE, P.. **Visual servoing**. Springer handbook of robotics, p. 841–866, 2016.

[53] KRAGIC, D.; CHRISTENSEN, H. I. ; OTHERS. **Survey on visual servoing for manipulation**. Computational Vision and Active Perception Laboratory, Fiskartorpsv, 15:2002, 2002.

[54] MALDONADO-VALENCIA, R.; RODRIGUEZ-GARAVITO, C.; CRUZ-PEREZ, C.; HERNANDEZ-NAVAS, J. ; ZABALA-BENAVIDES, D.. **Planning and visual-servoing for robotic manipulators in ros**. International Journal of Intelligent Robotics and Applications, 6(4):602–614, 2022.

[55] DONG, G.; ZHU, Z.. **Position-based visual servo control of autonomous robotic manipulators**. Acta Astronautica, 115:291–302, 2015.

[56] METNI, N.; HAMEL, T.. **A uav for bridge inspection: Visual servoing control law with orientation limits**. Automation in construction, 17(1):3–10, 2007.

[57] NARAYANAN, A. S.. **Qr codes and security solutions**. International Journal of Computer Science and Telecommunications, 3(7):69–72, 2012.

[58] OLSON, E.. **Apriltag: A robust and flexible visual fiducial system**. In: 2011 IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION, p. 3400–3407. IEEE, 2011.

[59] ALSALAM, B. H. Y.; MORTON, K.; CAMPBELL, D. ; GONZALEZ, F.. **Autonomous uav with vision based on-board decision making for remote sensing and precision agriculture**. In: 2017 IEEE AEROSPACE CONFERENCE, p. 1–12, 2017.

[60] OŠČÁDAL, P.; HECZKO, D.; VYSOCKỲ, A.; MLOTEK, J.; NOVÁK, P.; VIRGALA, I.; SUKOP, M. ; BOBOVSKỲ, Z.. **Improved pose estimation of aruco tags using a novel 3d placement strategy**. Sensors, 20(17):4825, 2020.

[61] ZHAO, B.; LI, Z.; JIANG, J. ; ZHAO, X.. **Relative localization for uavs based on april-tags**. In: 2020 CHINESE CONTROL AND DECISION CONFERENCE (CCDC), p. 444–449. IEEE, 2020.

[62] NASTESKI, V.. **An overview of the supervised machine learning methods**. Horizons, b 4:51–62, 2017.

[63] OUALI, Y.; HUDELOT, C. ; TAMI, M.. **An overview of deep semi-supervised learning**, 2020.

[64] GARDNER, M.; DORLING, S.. **Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences**. Atmospheric Environment, 32(14):2627–2636, 1998.

[65] ŠARŪNAS RAUDYS. **Evolution and generalization of a single neurone: I. single-layer perceptron as seven statistical classifiers**. Neural Networks, 11(2):283–296, 1998.

[66] HORNIK, K.; STINCHCOMBE, M. ; WHITE, H.. **Multilayer feedforward networks are universal approximators**. Neural Networks, 2(5):359–366, 1989.

[67] LI, Z.; LIU, F.; YANG, W.; PENG, S. ; ZHOU, J.. **A survey of convolutional neural networks: Analysis, applications, and prospects**. IEEE Transactions on Neural Networks and Learning Systems, 33(12):6999–7019, 2022.

[68] KHALID, S.; KHALIL, T. ; NASREEN, S.. **A survey of feature selection and feature extraction techniques in machine learning**. In: 2014 SCIENCE AND INFORMATION CONFERENCE, p. 372–378, 2014.

[69] DENG, L.; YU, D.. **Deep learning: Methods and applications**. Foundations and Trends® in Signal Processing, 7(3–4):197–387, 2014.

[70] THAKUR, P.; SHEOREY, T. ; OJHA, A.. **Vgg-icnn: A lightweight cnn model for crop disease identification**. Multimed Tools Appl, 82:497–520, 2023.

[71] NIAN, R.; LIU, J. ; HUANG, B.. **A review on reinforcement learning: Introduction and applications in industrial process control**. Computers Chemical Engineering, 139:106886, 2020.

[72] **Deep Reinforcement Learning**. Unknown Publisher, 7 2019. Published: 11 July 2019.

[73] Wiering, M.; Otterlo, M., editors. **Reinforcement Learning: State-of-the-Art**. Adaptation, Learning, and Optimization. Springer Berlin, Heidelberg, 1 edition, 3 2012. Published: 05 March 2012.

[74] SUTTON, R. S.; BARTO, A. G.. **Introduction to Reinforcement Learning**. MIT Press, 1998.

[75] SZLAK, L.; SHAMIR, O.. **Convergence results for q-learning with experience replay**, 2021.

[76] DANKWA, S.; ZHENG, W.. **Twin-delayed ddpg: A deep reinforcement learning technique to model a continuous movement of an intelligent robot agent**. In: PROCEEDINGS OF THE 3RD INTERNATIONAL CONFERENCE ON VISION, IMAGE AND SIGNAL PROCESSING, 2019.

[77] YUAN, X.; WANG, Y.; ZHANG, R.; GAO, Q.; ZHOU, Z.; ZHOU, R. ; YIN, F.. **Reinforcement learning control of hydraulic servo system based on td3 algorithm**. Machines, 10(12):1244, 2022.

[78] POPESCU, D. C.; CERNAIANU, M. O. ; DUMITRACHE, I.. **Automatic rough alignment for key components in laser driven experiments using fiducial markers**. Journal of Physics: Conference Series, 1079(1):012013, aug 2018.

[79] TOBIN, J.; FONG, R.; RAY, A.; SCHNEIDER, J.; ZAREMBA, W. ; ABBEEL, P.. **Domain randomization for transferring deep neural networks from simulation to the real world**. In: 2017 IEEE/RSJ INTERNATIONAL CONFERENCE ON INTELLIGENT ROBOTS AND SYSTEMS (IROS), p. 23–30, 2017.

[80] RAFFIN, A.; HILL, A.; ERNESTUS, M.; GLEAVE, A.; KANERVISTO, A. ; DORMANN, N.. **Stable baselines3**. 2019.

[81] PASZKE, A.; GROSS, S.; MASSA, F.; LERER, A.; BRADBURY, J.; CHANAN, G. ; OTHERS. **Pytorch: An imperative style, high-performance deep learning library**. Advances in Neural Information Processing Systems, 32, 2019.

[82] PATIL, M.; WEHBE, B. ; VALDENEGRO-TORO, M.. **Deep reinforcement learning for continuous docking control of autonomous underwater vehicles: A benchmarking study**. In: OCEANS 2021: SAN DIEGO–PORTO, p. 1–7. IEEE, 2021.

[83] ERIKSSON, H.-E.; PENKER, M.; LYONS, B. ; FADO, D.. **UML 2 Toolkit**. John Wiley & Sons, 2003.

[84] FOUNDATION, O. S. R.. **ROS: Robot Operating System**. `http://www.ros.org/`.

[85] INC., D.. **Docker: Empowering app development for developers**. `https://www.docker.com`.

A
Computational Environment

During the software development phase, a *ThinkPad P1 Gen 4* was used as the main computer. This control unit presents the following technical aspects:

- *Intel Core i7* processor with clocking of 300 GHz.
- Random Access Memory of 64 GB size.
- *NVIDIA Quadro graphics card RTX A200*, with 3328 Compute Unified Device Architecture (CUDA) cores and a dedicated memory of 6 GB.

The operating system to support the software development is GNU/Linux. This open-source operating system has been widely adopted in various computing environments, from personal computers to large enterprise systems. It is widely regarded as a stable, secure, and reliable operating system that provides users with a flexible and customizable computing experience [43]. Overall, Linux's open-source nature, high performance, large community, hardware compatibility, and robust libraries and tools make it a popular choice for robotics development. Moreover, The Linux Operating System figures as a mandatory requirement to use the Robot Operating system (ROS) as a robotics framework. For the proposed work, the distribution Debian/Ubuntu 20.04 Focal Fossa was used. This distribution version is a Long Term Support release, which means five years, until April 2025, of free security and maintenance updates, guaranteed.

A.1
Robot Operating System

ROS is an open-source, meta-operating system that provides software tools and support for robot applications. It is developed by a community of contributors and maintained by the Open Source Robotics Foundation (OSRF). As mentioned before, ROS and Linux work together to provide a robust platform for building, deploying, and running complex robotic systems [84].

In a nutshell, it provides a standard set of software libraries and tools to develop robot applications. It is designed to support code reusability,

sharing, and collaboration between roboticists, enabling them to develop and deploy complex robotic systems more efficiently. ROS offers a standard communication interface, allowing different parts of a robot system to interact and exchange data, as well as libraries for motion planning, perception, control, and more. Additionally, ROS provides a large community of developers and users who contribute software packages and tools to the platform, making it a thriving ecosystem for robotics research and development [84].

The ROS Melodic Morenia distribution was used in the scope of this work, which counted with the *ros-melodic-desktop-full* installation package to make the graphical tools available during the development phase. The ROS installation and configuration was accomplished based on a Docker image, built considering all the required dependencies and environment variables values to run this work output application in any Linux-based distribution. Figure A.1 illustrates the Docker stack implemented for this work.

A.2 Docker

Docker is a platform that allows developers to create, deploy, and run applications inside containers efficiently. It uses a containerization technology that packages applications and dependencies together in a single unit, which can be easily moved between development, testing, and production environments [85]. Docker containers provide a consistent and reproducible environment, allowing applications to run consistently and reliably across different environments. Taking the software development applied to robotics into account, according to [41], Docker offers critical advantages, including:

- **Isolation**: Docker containers provide an isolated environment for each component of the robotics system, reducing the risk of interference and dependencies between them.
- **Portability**: Docker containers can be easily moved between environments, making it easier to test and deploy robotics systems on different hardware platforms.
- **Scalability**: Docker containers can be easily scaled up or down based on the demands of the robotics system, providing a flexible and scalable solution.
- **Reproducibility**: Docker containers can be used to recreate the exact environment and dependencies required to run the robotics system, ensuring reproducibility and reducing the risk of issues arising from differences in the environment.

– **Simplified Maintenance**: Docker containers make it easier to manage and maintain the components of a robotics system, reducing the time and effort required to keep the system up-to-date.

The *docker-ce* (version 20.10.23) and *nvidia-docker2* (version 2.8.0) packages were used in this work. The ROS Melodic Morenia target Linux distribution (Debian Ubuntu 18.04 Bionic) for this project was installed in the correspondent Docker image, as even the other tools required for the development and testing phases, such as the simulation and reinforcement learning tools. More details about the Docker development paradigm towards robotics applications are depicted in [84].

A.2.1 Development Environments

Figure A.1 illustrates the interaction between the entities described, which are considered the computational base stack for developing this work. As Figure A.1 shows, two distributions were considered for the development environment. One (Debian Ubuntu 18.04 Bionic) to accommodate the development applications, such as ROS and Gazebo, and the another one (Debian Ubuntu 20.04 Focal) to accommodate the development local tooling. Considering this scope, the advantages mentioned in the previous Section could be applied, highlighting the main one related to the portability to deploy the result of this work in any Linux distribution since both of the environments share the same kernel main resources.

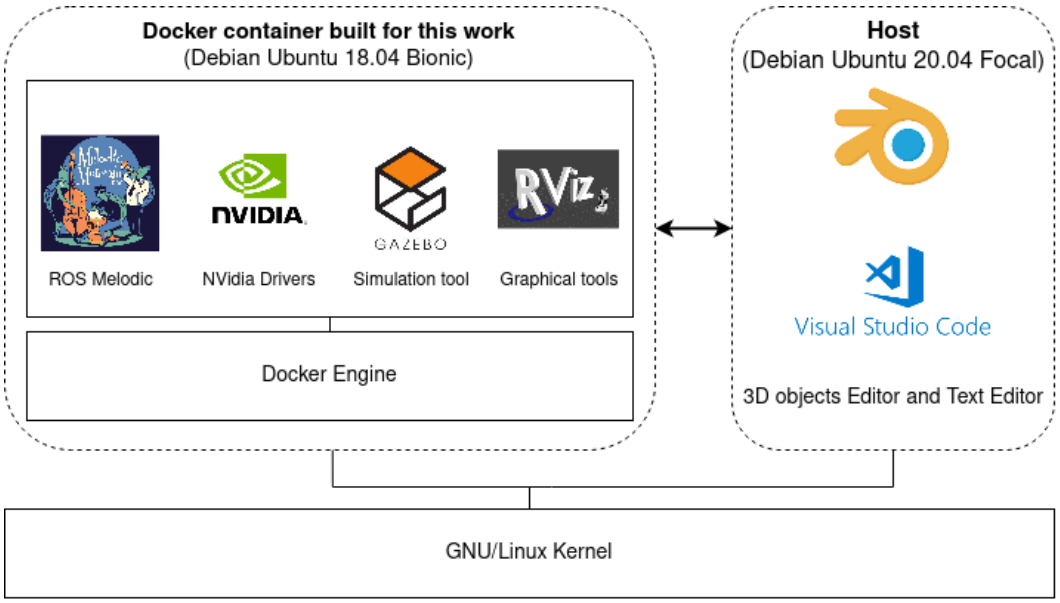


Figure A.1: Development Environment designed for this work.

A.2.2 Experimental Environments

A docking architecture was implemented to run the training algorithms of the machine learning and RL models, detailed in Chapter 3, and also the simulation environment in the target servers. Figure A.2 illustrates the architecture design to deploy the experiment processes into the remote servers.

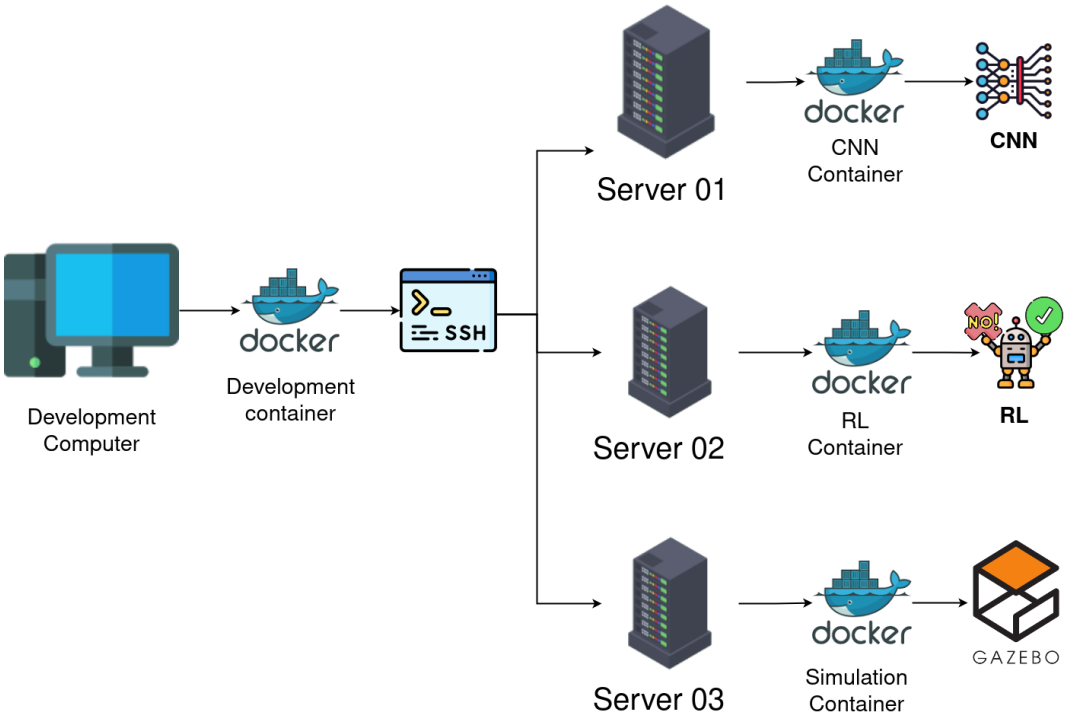


Figure A.2: Software deployment diagram for the usage of remote servers.

As Figure A.2 depicts, the deployment sequence of the machine learning applications on the remote servers consists of:

1. **Development computer** (which has its configuration detailed in Section A)**:** runs the development container, which has installed the main tools used to develop the code, such as IDEs and debugging tools. The development container is also responsible for managing remote access to the servers. The development container design is detailed in Figure A.1.

2. **Server 01:** it was used to run the CNN applications. The CNN applications have their own containers since the dependencies and configurations are specific for this type of application.

3. **Server 02:** it was used primarily to train the RL agent, with the same assumptions for the CNN containers. The RL training step also took into account *Server 02*, since it requires more resources once the simulation application must be running in order to execute the learning episodes.

4. **Server 03:** it was used to run the simulation and required processes.

A.3 Conventions

Throughout the development of the application associated with this work, software conventions were applied to leverage the code readability, consistency, and maintenance, aiming to keep the developed packages' style and definitions up to the "good" rules defined by the ROS community. Most of those rules are described in the ROS Enhancement Proposal (REP) pages [83].

A.3.1 File Tree

As the proposed work consists of a multidisciplinary application, a well-structured file tree is required since it can make it easier to track changes and monitor the progress of projects between the authors. Moreover, it makes possible the package organization by the usage of git submodules[1]. Figure A.3 depicts the file tree system adopted for this work.

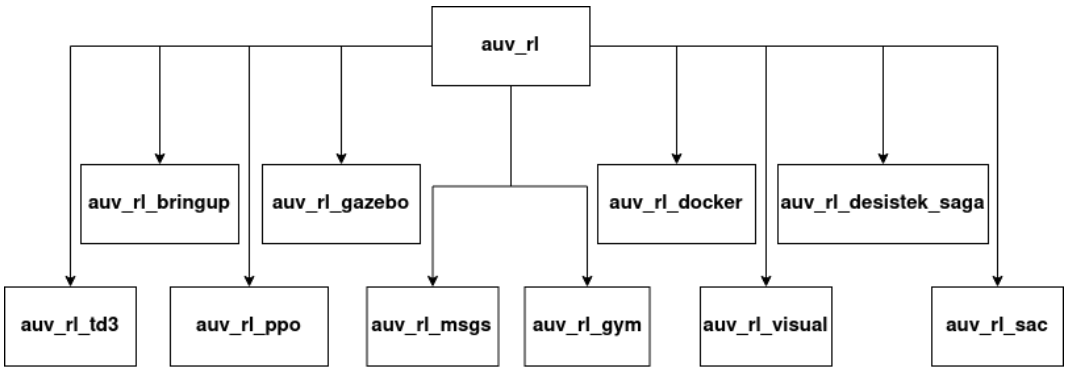


Figure A.3: File tree adopted for this work.

The package naming convention and the scope definition of each package were based on the REP 144 [83].

As Figure A.3 shows, eleven main packages are divided according to each discipline covered by this work:

[1]Git is a distributed version control system that is used to manage software projects and track changes to source code over time. Linus Torvalds created it in 2005 as a free and open-source alternative to proprietary version control systems. Git allows multiple developers to collaborate on a project by enabling them to share and merge changes to the codebase. It provides features such as branch management, merge tracking, and conflict resolution, making it a popular choice for software development teams.

- ***auv_rl***: It corresponds to the meta-project level, where the git submodules are managed and the main bring-up files, to run the application, are located.
- ***auv_rl_bringup***: A ROS package that concentrates all files responsible for initiating all required resources for running the main application.
- ***auv_rl_gazebo***: A ROS package aimed to hold the gazebo worlds and 3D object files that are used for the simulation. It also describes the interaction between ROS and Gazebo, by setting the plugins behavior.
- ***auv_rl_docker***: A non-ROS package that holds the files required to install and run the docker development environment.
- ***auv_rl_desistek_saga***: A set of ROS packages that aim to simulate the chosen AUV model.
- ***auv_rl_td3***: A ROS package that implements the Twin Delayed Deep Deterministic Policy Gradient (TD3) method, a RL technique considered in this work, detailed in Section 2.5.4.
- ***auv_rl_ppo***: A ROS package that implements the Proximal Policy Optimization (PPO) method, a RL technique considered in this work.
- ***auv_rl_msgs***: A ROS package that concentrates the topic, action, and service custom messages used by the packages of this work.
- ***auv_rl_gym***: A ROS package that concentrates the files required to create the reinforcement learning environment structure to evaluate the agent and appropriately computes the rewards, state, and actions.
- ***auv_rl_visual***: A set of ROS packages to enable the visual auto-docking software stack.
- ***auv_rl_sac***: A ROS package that implements the Soft Actor-Critic method, a RL technique considered in this work.

Additionally, two other third-party packages were considered in this work: the ***apriltag_ros***[2] and ***openai_ros***[3], used for relative pose estimation and a framework to run RL techniques considering Gazebo as primary simulator.

Each package has its inner structure that follows the conventions of its discipline or tool when applied. More details about each package can be found throughout this work. After this work be released, all packages should be public, with free access to the robotics community.

[2]https://github.com/AprilRobotics/apriltag_ros
[3]https://bitbucket.org/theconstructcore/openai_ros.git

A.3.2 Coordinate frames

In robotics, a common coordinate system used for defining the position and orientation of objects is the Cartesian coordinate system. However, this type of reference works for local tasks. In order to create a "universal" reference for robots and objects in an environment, global references are used. According to [12], global systems such as Earth-Centered Earth Fixed (ECEF) and geodetic systems describe the position of an object using a triplet of coordinates. Local systems such as East-North-UP (ENU), North-East-Down (NED), and Azimuth-Elevation-Range (AER) systems require two triplets of coordinates: one triplet describes the location of the origin, and the other triplet describes the location of the object concerning the origin.

The system adopted for this project is the ENU since it is advised for short-range Cartesian representations of geographic locations. According to [12], an ENU system uses the Cartesian coordinates (x_{East}, y_{North}, z_{Up}) to represent position relative to a local origin. The local origin is described by the geodetic coordinates (lat_0, lon_0, h_0). Note that the origin does not necessarily lie on the surface of the ellipsoid.

- The positive $x_{East-axis}$ points east along the parallel of latitude containing lat_0.
- The positive $y_{North-axis}$ points north along the meridian of longitude containing lon_0.
- The positive $z_{Up-axis}$ points upward along the ellipsoid normal.

Figure A.4 represents the ENU coordinate system, adopted for this project. In the Figure, the reference frame for the ENU system is the ECEF. It follows the REP 105 guideline, which describes how the global and local frames should be represented.

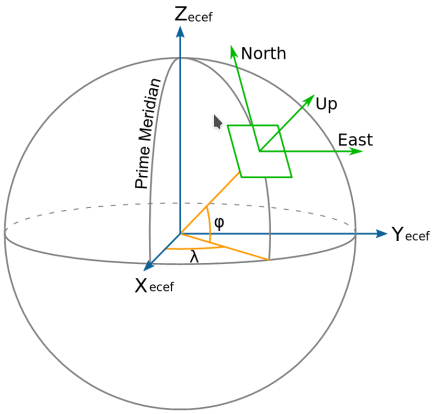


Figure A.4: ENU coordinate system [83].