FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Scheduling Algorithms for Underwater Pick and Place Tasks for Offshore WindFarm Intervention Tasks

Tiago Filipe Carvalho do Vale

Mestrado em Engenharia Eletrotécnica e de Computadores

Supervisor: João Vasco Couto Amorim Marques

Co-Supervisor: Andry Maykol Gomes Pinto

October 30, 2022

Resumo

Vivemos numa época em que a necessidade de energias renováveis é a maior de sempre, com a energia eólica a tornar-se um dos maiores focos entre os países europeus. Como resultado, tem havido elevada investigação e desenvolvimento no domínio da energia eólica em contexto aquático. Contudo, um dos maiores problemas com parques eólicos flutuantes é o seu custo de manutenção.

Manipuladores de teleoperação têm sido utilizados para realizar uma vasta gama de tarefas subaquáticas, tais como inspecção e limpeza. Contudo, este tipo de solução tem problemas tais como má visibilidade, má percepção de imagem, e fadiga do operador. Consequentemente, esta dissertação foca-se em manipuladores robóticos autónomos que podem ser montados em AUVs para realizar este tipo de tarefas.

O problema que este trabalho procura resolver consiste em determinar a melhor ordem de movimentos e acções a serem executadas pelo manipulador para realizar uma tarefa de pick & place. Este problema pode ser categorizado como um Travelling Salesman Problem (TSP) e procura assim determinar a rota que apresenta o menor custo para viajar através de uma série de pontos de interesse, considerando as restrições e limitações do manipulador e do ambiente subaquático.

Esta dissertação propõe um sistema multicritério para agendar e executar missões subaquáticas, focando-se em tarefas de pick & place. Este sistema contém algoritmos de scheduling que funcionam com o planeamento de trajetórias em tempo real, juntamente com desvio de obstáculos. O trabalho também considera parâmetros relevantes para condições subaquáticas, tais como a distância percorrida pelo braço, o movimento total das juntas, a segurança do movimento, o tempo total de planeamento e o torque máximo sentido pelas juntas. O sistema utiliza a framework ROS para simular as missões, permitindo a sua validação antes da sua execução num cenário real.

Além disso, os resultados obtidos fornecem informações essenciais, tais como a capacidade, o tempo de execução e as restrições dos algoritmos utilizados: Dijkstra's, GRASP e A*.

Uma vez completos, o sistema e os algoritmos foram validados através de uma série de testes, num cenário de pick & place altamente congestionado em condições secas e subaquáticas. Concluindo, este trabalho oferece uma contribuição valiosa para a inspecção de estruturas marítimas, o estudo de algoritmos de scheduling e os procedimentos de manipulação para as missões de pick & place em cenários autónomos subaquáticos.

Abstract

We live in times where the need for renewable energies is the highest it has ever been, with wind power becoming one of the biggest focuses among European countries. As a result, there has been high research and development on offshore wind power. However, one of the biggest problems with floating offshore wind farms is the maintenance cost.

Teleoperation manipulators have been used to perform a wide range of underwater tasks such as inspection and cleaning. However, this type of solution has problems such as poor visibility, image perception, and the operator's fatigue. Consequently, this thesis focuses on autonomous robotic manipulators that can be mounted on AUVs to perform these tasks.

The problem this thesis seeks to solve consists of determining the best order of movements and actions to be performed by the manipulator to perform a pick & place task. This problem can be categorised as a Travelling Salesman Problem (TSP) and thus seeks to determine the route that presents the lowest cost to travel through a series of points of interest while considering restrictions and constraints of the manipulator and the underwater environment.

This thesis proposes a multi-criteria system for scheduling and performing underwater missions, mainly pick & place tasks. This system contains scheduling algorithms that work with real-time path planning and obstacle avoidance. The work also considers parameters relevant to underwater conditions, such as the distance of the end-effector, the total movement of the joints, the safety of the movement, the total planning time and the maximum torque of the joints. The system uses the ROS framework to simulate the missions, allowing their validation before performing them in a real scenario.

In addition, the obtained results provide essential information, such as the capability, execution time and constraints of the used algorithms: Dijkstra's, GRASP and A*.

Once completed, the system and the algorithms were validated through a series of tests in a highly congested pick place scenario in dry and underwater conditions. Thus, it is a valuable contribution to the inspection of maritime structures, the study of scheduling algorithms and the manipulation procedures for pick & place missions in autonomous underwater scenarios.

Agradecimentos

Esta Tese de Mestrado é minha, foi feita por mim e tem, indubitavelmente, o meu nome inscrito. Mas seria injusto dizer que a realizei sozinho, pois muitas pessoas me acompanharam neste desafio que é apenas o final de um ciclo académico e o início da minha vida profissional.
São muitas as pessoas que foram e continuarão a ser importantes para mim neste desígnio que significa a minha ambição de ser mais e melhor.

A essas pessoas não poderia deixar de agradecer vivamente:
Ao professor Andry Maykol Gomes Pinto pelo saber e opiniões valiosas que me transmitiu, tendo sindo um pilar muito importante na realização deste trabalho.
Ao meu orientador João Vasco Couto Amorim Marques pela disponibilidade e simpatia desde o primeiro dia no CRAS (Centro de Robótica e Sistemas Autónomos).
Ao Engenheiro Renato Jorge Silva pela constante disponibilidade, amizade e conhecimento partilhado.
A todos os meus colegas no CRAS pela camaradagem, simpatia e apoio, bem por me proporcionarem um fantástico ambiente de trabalho desde o primeiro dia em que iniciei este desafio.
Ao Engenheiro João Manuel Dionísio pela disponibilidade e saber partilhados.
À Faculdade de Engenharia da Universidade do Porto e ao INESC pela oportunidade e por terem criado todas as condições para que este trabalho fosse possível.
Aos meus pais, por serem um pilar muito importante na minha vida e por me proporcionarem tudo o que sempre precisei. Sem eles, nada do que fiz até hoje teria sido possível. À minha mãe, por todo o carinho e disponibilidade e por querer sempre o melhor para mim. Ao meu pai, por todo o carinho e paciência e por me ensinar que tudo está ao nosso alcance desde que nós trabalhemos para isso.
Aos meus avós e a toda a minha família, por me mostrarem que o carinho e o respeito são das coisas mais importantes da vida.
À Anastasia, por ter sido um grande pilar durante todo este percurso académico. Pelo carinho, paciência e motivação em todos os momentos em que o mesmo se mostrou difícil.
A todos os meus amigos, pois sem as noites de estudo, apontamentos partilhados e discussões produtivas nada disto teria sido possível.
A todos os que participaram, diretamente ou indiretamente, neste percurso da minha vida.

Que o próximo desafio seja ainda melhor.

Tiago Filipe Carvalho do Vale

"Don't wish it was easier
wish you were better.
Don't wish for less problems
wish for more skills.
Don't wish for less challenge
wish for more wisdom."

Jim Rohn

Contents

Resumo i

Abstract iii

Agradecimentos v

Abbreviations xv

1 Introduction 1
- 1.1 Context and Motivation . . . 1
- 1.2 The problem . . . 3
- 1.3 Objectives . . . 3
- 1.4 Document Structure . . . 4

2 Background 7
- 2.1 Manipulators . . . 8
  - 2.1.1 Stepper Motors . . . 8
  - 2.1.2 Encoders . . . 10
- 2.2 Kinematics . . . 11
  - 2.2.1 Forward Kinematics . . . 11
  - 2.2.2 Inverse Kinematics . . . 12
  - 2.2.3 Jacobian Matrix . . . 12
- 2.3 Motion Planning . . . 13

3 State of the art 17
- 3.1 Underwater Manipulators . . . 17
- 3.2 Motion Planning . . . 19
  - 3.2.1 Rapidly-exploring Random Trees (RRT) . . . 20
  - 3.2.2 Probabilistic Roadmap Method (PRM) . . . 21
  - 3.2.3 Kinematic Planning by Interior-Exterior Cell Exploration (KPIECE) . . . 22
  - 3.2.4 Optimization-based Algorithms . . . 23
- 3.3 Pick & Place . . . 23
- 3.4 Scheduling . . . 25
  - 3.4.1 Travel Salesman Problem (TSP) . . . 27
- 3.5 Critical Review . . . 30
  - 3.5.1 Manipulators . . . 30
  - 3.5.2 Motion Planning . . . 30
  - 3.5.3 Scheduling . . . 31

4 Developed Work **33**
4.1 Methodology . . . 33
4.2 Used Tools . . . 34
4.2.1 ROS - Robot Operating System . . . 34
4.2.2 MoveIt . . . 34
4.3 Architecture . . . 36
4.3.1 Node Architecture . . . 36
4.4 Pick & Place . . . 38
4.5 Reach Bravo 7 Manipulator . . . 40
4.6 Bravo 7 controller . . . 42
4.7 Scheduling . . . 43
4.7.1 Dijkstra's greedy algorithm . . . 45
4.7.2 GRASP (Greedy randomized adaptive search procedure) . . . 47
4.7.3 A* algorithm . . . 49
4.7.4 Metrics . . . 52

5 Results **55**
5.1 Environment . . . 55
5.2 Motion Capture System . . . 59
5.3 Scheduling Algorithms . . . 59
5.3.1 Final Cost . . . 60
5.3.2 GRASP and number of iterations . . . 61
5.3.3 Execution Time . . . 64
5.3.4 Sensibility test - Weights . . . 64
5.3.5 Specific Scenarios . . . 65
5.4 Testing the system . . . 68

6 Conclusions and Future Work **71**
6.1 Conclusions . . . 71
6.2 Future Work . . . 72

A Sensibility Test Results **73**
A.1 Sensibility Test Result Tables . . . 73

B Testing the system **77**
B.1 Dry Tests . . . 77

References **79**

List of Figures

1.1 Subsea Operations Offshore . . . 2
1.2 Commercial divers performing offshore inspections . . . 2
1.3 Developed Model for the TSP of this work . . . 4

2.1 Reluctance Stepper Motor . . . 9
2.2 Hybrid Stepper Motor . . . 9
2.3 Stepper motor toque-speed curve . . . 9
2.4 Output signals from a rotary absolute encoder . . . 11
2.5 Example of a Robotic 7 DoF Structure . . . 11
2.6 RRT - Case of random point appearing inside an obstacle boundary . . . 14

3.1 Kraft Raptor . . . 18
3.2 Eca robotics 7E . . . 18
3.3 Trident Project: Girona 500 I-AUV with a 7 DoF manipulator . . . 19
3.4 ROV Holland I with two Schilling Orion 7P manipulator . . . 19
3.5 Road-map chosen from bounding boxes generated from the octree map . . . 22
3.6 Final path connecting the start and goal . . . 22
3.7 Trees of Motion using KPIECE . . . 23
3.8 Proposed system architecture . . . 24
3.9 Concept of skills . . . 25
3.10 System hierarchy . . . 25
3.11 Gant Chart for all states for a 4x3 open-shop problem . . . 26
3.12 Online Performance-aware task scheduler . . . 27
3.13 Comparison of the results of path planning algorithms . . . 28
3.14 Paths of the path planning algorithms . . . 29
3.15 Simulations results for TSP instance using GRASP . . . 29
3.16 Motion planners comparison: Arm planning time per simulation . . . 31
3.17 Motion planners comparison: Arm execution time per simulation . . . 31
3.18 Planning Time of different Algorithms in a 7 DoF arm . . . 31

4.1 Developed Model divided into three layers . . . 34
4.2 Simulated scenario with obstacles . . . 36
4.3 Proposed system architecture . . . 37
4.4 Node Architecture . . . 37
4.5 Pick & Place - Manipulator steps . . . 39
4.6 Reach Bravo 7 by Blueprintlab . . . 40
4.7 Bravo 7 Configuration 1 . . . 41
4.8 Bravo 7 Configuration 2 . . . 41
4.9 Reach Bravo 7 Work Space . . . 41

4.10 Reach Bravo 7 Joint Frames 42
4.11 Diagram of the controller 43
4.12 Graph for the specific TSP 44
4.13 Dijkstra's greedy Representation 46
4.14 A* Method's function 49

5.1 Model of Developed Environment 56
5.2 Collision objects in the simulator 56
5.3 Model of Pick Objects 57
5.4 Pick Objects - Transparent Model 57
5.5 3D Printed Pick Objects 57
5.6 Final Setup in a dry environment 58
5.7 Final Setup in an underwater environment 58
5.8 Final costs of the 3 algorithms 62
5.9 GRASP: Iterations vs Cost 62
5.10 Three Algorithms: Iterations vs Cost 63
5.11 GRASP: Iterations vs Execution Time (ms) 63
5.12 Three Algorithms: Iterations vs Execution Time (ms) 65
5.13 Underwater Scenario - During Tests 69
5.14 Bravo 7 end-effector trajectory comparison 70

List of Tables

2.1 D-H parameters 12

4.1 Reach bravo 7 joints limits 41
4.2 Standard Denavit–Hartenberg parameters for Bravo 7 42
4.3 Weights of the metrics 54

5.1 Nodes name and corresponding number 59
5.2 Direct Comparison between the scheduling algorithms - using RRT Connect . . . 60
5.3 Direct Comparison between the scheduling algorithms - using BKPIECE 61
5.4 Direct Comparison between the scheduling algorithms - using PRM 61
5.5 Scheduling algorithms with execution time - using RRT Connect 64
5.6 Scheduling Algorithms - First Scenario 66
5.7 Scheduling Algorithms - Second Scenario 67
5.8 Scheduling Algorithms - Third Scenario 68
5.9 Scheduling Algorithms - Fourth Scenario 69

A.1 Sensibility test - First set of of weights 73
A.2 Sensibility test - Second set of of weights 74
A.3 Sensibility test - Third set of of weights 74
A.4 Sensibility test - Fourth set of of weights 75
A.5 Sensibility test - Fifth set of weights 75

Abbreviations and Symbols

AAV	Autonomous Aerial Vehicle
AUV	Autonomous Underwater Vehicle
ASV	Autonomous Surface Vehicle
BKPIECE	Bi-directional KPIECE
DC	Direct Current
DOF	Degrees of Freedom
EPM	Electric Powered Manipulators
EST	Expansive Space Trees
FCL	Collision Check Library
GRASP	Greedy randomized adaptive search procedure
HM	Hydraulic Manipulators
KPIECE	Kinematic Planning by Interior-Exterior Cell Exploration
HSV	Hue Saturation Value
KDL	Kinematics and Dynamics Library
MTP	Motion and Task Planners
OPML	Open Motion Planning Library
PRM	Probabilistic Roadmap Method
ROS	Robot Operating System
ROV	Remotely Operated Vehicle
RRT	Rapidly-exploring Random Trees
SAS	Synthetic Aperture Sonar
SRDF	Semantic Robot Description Format
URDF	Unified Robot Description Format

Chapter 1

Introduction

1.1 Context and Motivation

We live in times where the need for renewable energies is the highest it has ever been. This is mainly due to pollution and the higher scarcity of non-renewable energy sources. Thus, countries all across the globe have been prioritizing the development and dissemination of renewable energy technologies to provide a sustainable energy supply to meet the world's current energy demand [1]. The European Union has established in its Green Deal plan the ambitious goal of net-zero Green House Gas emissions by 2050, and among all clean energies, wind power is becoming one of the biggest focuses among European countries. This is due to its technological maturity and improvements. However, the noise nuisance, ecological impacts, and scarcity of adequate onshore locations delay its development. As a result, there has been high research and development on offshore wind power [2].

However, one of the biggest problems with floating offshore wind farms is the cost of maintenance: since these are away from the coast, there are additional constraints and difficulties. The costs of travelling to site (using boats or helicopters) and the need for divers and specialised professionals make the most of it. As Martinez and Inglesias state in [2], "Operation and maintenance (O&M) costs, which comprise the hire of staff, port facilities and equipment, and the repair of damaged components, often account for a substantial share of the total costs (25%). In offshore projects, these costs present an additional variable component that accounts for expenses related to travelling to site – time and fuel spent". Examples of such subsea operations can be seen in Figures 1.1 and 1.2.

This is where Unmanned Aerial Vehicles (UAVs), Autonomous Underwater Vehicles (AUVs), Autonomous Surface Vehicles (ASVs) and Remotely Operated Vehicles (ROVs) come into play. These types of vehicles came to facilitate multiple sea activities, such as the inspection and maintenance of marine structures such as offshore windFarms.

Wind turbines can be classified based on different criteria such as direction of the rotating axis, power transmission method or foundation type. For the first, turbines can be vertical or horizontal axis. In terms of power transmission, they can have a direct drive or a gearbox transmission. For

the latter, wind turbines can be bottom-fixed or floating: the first being connected directly to the ground by a steel tube pile or a jacket, and the second using anchoring systems. Even though wind turbines can be separated on the basis of different criteria due to the wide variety of characteristics, they all have underwater components that need maintenance and inspection [3, 4, 5].

Figure 1.1: Subsea Operations Offshore [1]

Figure 1.2: Commercial divers performing offshore inspections [2]

Robotic manipulators are used in multiple industrial sectors such as automotive, electronics, building, aerospace, and textile. These have been executing a wide variety of applications such as handling, welding, dispensing, polishing and painting [6, 7]. In the last years, robotic manipulators have been mounted on AUVs to be able to perform a wide range of underwater tasks such as pipe inspection, salvage of sunken objects, surface cleaning, valve operating, drilling, cable laying and repair, clearing debris, and geological sampling [8]. By combining an AUV with a robotic manipulator, it is possible to reach difficult reaching locations and, at the same time, execute complex tasks such as welding, cleaning or doing maintenance.

[1] RS Diving Contractor
[2] Blueye Robotics

The majority of commercial underwater manipulators are not servo-controlled and do not support kinematic control approaches. They use traditional teleoperation approaches that rely on human operator skills and experience. The operator has visual feedback of the scene through a camera or other sensoring systems and performs the tasks by manipulating the robot's arm. However, in most cases, the pilot has to interact with an enormous quantity of information while adjusting parameters such as the field of view and lighting, which can result in a high cognitive load, sometimes resulting in failed or prolonged missions due to operator fatigue. In addition, simple tasks from an industrial robotics perspective can become challenging even for a skilled operator due to poor visibility and poor image perception [8, 9, 10]. Due to the problems arising from teleoperation in underwater maintenance tasks, it is necessary to perform pick & place tasks autonomously using an autonomous manipulator. In addition, to perform underwater pick and place tasks more efficiently and more safely, it is necessary to develop and implement scheduling algorithms that work with real-time path planning and obstacle avoidance, while considering parameters that are relevant to underwater conditions such as the travelled distance by the end-effector, the total movement of the joints, the security, the total planning time and the maximum torque of the joints. In summary, this work looks to study scheduling algorithms for an approach to performing autonomous pick & place tasks using an autonomous robotic manipulator.

1.2 The problem

The problem the current thesis seeks to solve consists of determining the best order of movements and actions to be performed by the manipulator to perform a pick & place task. When working with multiple objects and possible operations, a problem surges in knowing the order of the tasks to perform. In addition, multiple constraints need to be considered in this method, such as restrictions of the pick & place itself that require that certain movements are ordered in a specific manner. In summary, this thesis consists of the implementation of a scheduler with multiple scheduling algorithms that is capable of taking into account all the restrictions to solve a problem of the Travel Salesman Problem (TSP) type and thus determine the route that presents the lowest cost to travel through a series of points of interest. The representation of the TSP, an NP-hard computational problem, can be seen in Figure 1.3, which represents the possibilities of movements for the manipulator to perform the pick & place task. Furthermore, there needs to be a way of evaluating the different scheduling possibilities in an underwater environment to allow the decision of the best path to perform. For this problem, fixed obstacles will also be considered, such as the working tables, the pick objects and other area restricting objects.

1.3 Objectives

The main goal of this work is to develop and test scheduling algorithms for a new approach for pick & place tasks using a 6 DoF manipulator in underwater scenarios, for the maintenance and inspection of offshore wind turbines in wind farms. The system will be initially developed in a

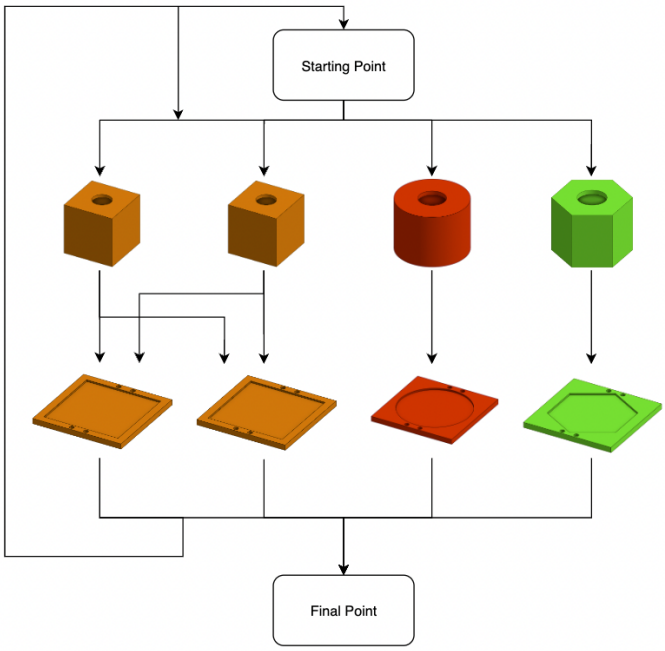

Figure 1.3: Developed Model for the TSP of this work

simulation environment, with the ultimate goal of its implementation and testing with a Bravo 7 manipulator owned by the Center for Robotics and Autonomous Systems from INESC TEC (Institute of Systems and Computer Engineering, Technology and Science), using an underwater mounted motion capture system. Therefore, the main contributions of this thesis include:

- The development of scheduling algorithms for underwater pick and place scenarios;
- A study on the performance of the developed scheduling algorithms;
- The development of a multi-criteria Scheduler for pick & place operations in underwater manipulation;
- The study of the performance of the developed Scheduler according to different metrics;
- Development of a novel ROS (Robot Operating System) and Moveit based architecture for pick & place operations;
- A study on the performance of the pick & place system for underwater scenarios, resorting to a motion capture system, in a controlled underwater environment, recreating a pick & place scenario.

1.4 Document Structure

This thesis is divided into five chapters, according to the sequence of the work developed throughout the project. Chapter 1 introduces the context and motivation for this work, along with a brief

description of the solved problem. Afterwards, the objectives are presented. The chapter 2 discusses essential in-depth topics for this work, along with definitions and explanations of some of its theoretical parts. The chapter 3 presents the state of the art on the topics fundamental to obtaining the necessary knowledge to execute this thesis. The chapter 4 presents the developed work in a detailed way. The chapter 5 presents the results of the developed work, along with a comprehensive review of the results. Lastly, chapter 6 presents the conclusions of the conducted work and identifies topics for further work.

Chapter 2

Background

Today's market of robotic manipulators, including the experimental and prototypes, is composed of mainly two types of robotic arms: those that run on hydraulic power and those that run on electric power. Each one presents advantages and disadvantages [11].
Tasks such as autonomous systems inspections are used in many applications such as transportation, safety and agriculture. Such activities have the need for systems that can perform specific tasks in a safe, precise and autonomous way. This is where Motion and Task Planners (MTP) come into play: it plans the robot's navigation from start to goal position by defining a sequence of actions/movements [12, 13].
Autonomous Systems can perceive the environment through sensors by building environment maps. The planner will use these, together with constraints, to plan the sequence of steps of the path and, simultaneously, avoid possible obstacles [12].
In addition to underwater manipulators, motion planners and path planners, multiple theoretical and fundamental aspects will be needed to perform a pick & place scenario in underwater conditions. Robot Operating System (ROS) will be the meta-operating system used in this work. This open-source tool provides libraries, plugins and communication frameworks for the development of robotic applications. Furthermore, it also has software libraries to work with navigation, visualization of movement during simulation and motion planning [12]. A pick & place scenario will be developed using design and drafting tools in order to simulate the work. The scenario will include the manipulator, the objects and any other obstacles that end up being interesting for the study. The scenario will then be inserted into the Gazebo simulator, which will be the software used to simulate and test the developed work. This software consists of a simulator that makes possible the test of algorithms, the design of robots, the development of regression tests and the training of AI systems during realistic scenarios [14].
The architecture and the tasks for the path planner will be developed for a pick & place underwater scenario. The manipulator's kinematics will also be studied in order to control the multiple DOFs of the robotic arm.
With the objective of picking up the objects with the manipulator, their identification and recognition need to be done beforehand. A study about computer vision and image processing libraries

needs to be done to identify the best option. Special attention needs to be given to the fact that water and light transmission impedes colour dispersion and image clarity in an underwater scenario, which can cause errors. Many researchers developed algorithms to improve underwater images by estimating gravity gradient coefficients. Features such as colour, shape, contour, pixel intensity, texture, edges and corners can be used in such detection, and identification algorithms [15, 16, 17]. The work will be tested in a real environment by submerging the Bravo 7 manipulator in a pool owned by FEUP (Faculdade de Engenharia da Universidade do Porto) along with multiple test objects. The Qualisys system will be used in the pool to capture and evaluate the performance of the pick & place tasks. This professional optical motion capture system, which contains multiple sensors, will be used to calculate the exact position of the rigid body that is being picked up by the manipulator [18]. By comparing this position to the planned one, we can measure the system's accuracy.

2.1 Manipulators

Manipulators that run on electric power have two main components: the motors and the encoders. Motors are the components that allow the motion of each DoF of the manipulator and the encoders are the sensors that allow the estimation of position of the motion of each DoF.

2.1.1 Stepper Motors

The stepper motors are mounted on each DoF of the manipulator, making its motion possible. These are designed to rotate at a specific number of degrees for every electric pulse the control unit transmits. The voltage can control the operation of these motors applied to its terminals, making the rotation possible by controlling the frequency at which the motor is energised [19]. These are used in multiple robots due to their simplicity of control and the low system cost. At the same time, these have high accuracy and can produce high output torques at low angular velocities, making them perfect for manipulators that perform pick & place tasks. In addition to this, the torque can be applied with only a direct-current excitation of the motor. Their drives are digital, making these easily controllable with a simple digital controller, and their mechanical construction is simple, robust and highly reliable.
The position of the motors is achieved by the magnetic alignment between the teeth of the stepper's state and the rotor. These can be divided into two main designs: variable-reluctance stepper motors and hybrid stepper motors. The first type's magnetic flux is provided by the stator excitation, while in contrast, in the hybrid design, the interaction between the magnetic flux is produced by a rotor-mounted permanent magnet and that resulting from the stator winding's excitation. Hybrid steppers are more suited to limited-movement, high-resolution applications where variable-reluctance motors are suited to high-speed motions. The motors that belong to the second type produce a continuous torque, which means that the motor will retain its position even when the drive has no energy, which is useful for fail-safe applications. On the other hand, reluctance

stepper motors have a lower mass and cost than hybrid ones [20]. These two types can be seen in Figures 2.1 and 2.2, respectively.

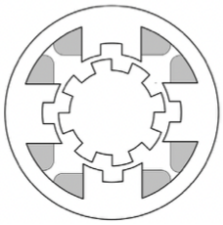

Figure 2.1: Reluctance Stepper Motor [20]

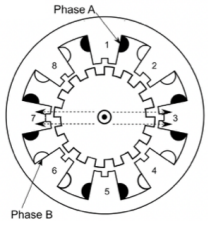

Figure 2.2: Hybrid Stepper Motor [20]

The stepper motors can be controlled in either a close or open loop. In a close loop, the motor will move a specified and accurately known distance by changing one excitation phase. The position is controlled by generating a pulse train of known length, converted by a translator to control the power given to the motor. On the other hand, in an open-loop system, there is no feedback from the motor or load to the controller, which means that the position cannot be detected and compensated [20].

In this type of motors, when the speed increases, the torque output will decline in a non linear way, declining faster in a high-speed stage. This decline can be observed in Figure 2.3.

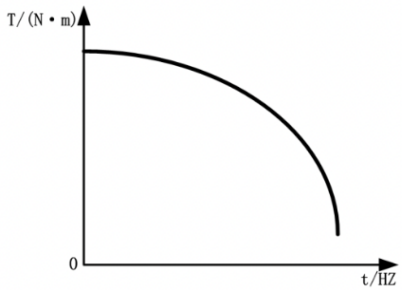

Figure 2.3: Stepper motor toque-speed curve [21]

This torque is the difference between electromagnetic torque T_0 and the drag torque T_L, while the stepper motor drag torque is $T_L = K_T \omega$ and the load torque is the accelerating torque $J \cdot \dot{\omega}$. The motion equation can be seen in equation 2.1.

$$J \cdot \dot{\omega} = T_0 - K_T \omega \tag{2.1}$$

with T_0 being the electromagnetic torque, J the rotational inertia of the load, K_T the torque constant and ω the rotary stepper motor angular velocity. Assuming that the motor speed accelerates from 0, $t = 0$ and $\omega = 0$, the equation for the angular velocity can be seen in equation 2.2.

$$\omega = \frac{T_0}{K_T}\left(1 - e^{-t/\tau}\right) \tag{2.2}$$

with $\tau = J/K_T$ and the frequency of drive pulse $\dot{K}$ being f. With this, the evolution of the frequency in time can be given by equation 2.3.

$$f(t) = f_m - f_m e^{-t/\tau} \tag{2.3}$$

with $f_m = T_0/(K_T \theta_b)$ being the highest frequency of the stepping motor operation and τ the time constant [21].

2.1.2 Encoders

Encoders are used in motion systems because they provide a way to estimate the position and velocity of the system, feedback that will then be used for motion control. Encoders can be linear or rotary, where the first measures the position and the velocity of a moving object in a straight line, such as pick and place machines used in electronics assembly. On the other hand, the rotary type can measure an object's angle and angular velocity. This type is used in many manipulators since they can measure the angles of each DoF.

Encoders can also be divided into optical and magnetic. The first type has higher accuracy and resolution and uses an optical sensor to detect the light transmitted or reflected from a disk. When the sensor receives this light, it outputs a high signal. When the opposite occurs, it outputs a low signal, tracking the movement. On the other hand, magnetic encoders use magnetic poles distributed around the wheel to detect the movements. A magnetic sensor detects the changes in the magnetic field along with the motion and outputs a digital signal. Magnetic encoders can be used in various fluid environments with less power, but they usually have less resolution and accuracy than optical encoders. In addition, the magnetic type may be preferable when there is a possibility of having anything blocking the light from an optical encoder, such as fluids or fog [1].

Encoders can also be absolute or incremental. In the incremental type, a single code-generated track can indicate the velocity of the motion but not the direction. In order to detect the direction of the movement, an encoder with two channels and two code tracks needs to be incorporated by these with a 90º difference. On the other hand, absolute rotary encoders provide the object's position. This is due to the fact that the sensor scans a specific code that represents a particular position code. Due to how they work, the position can be measured even after the encoder is turned off for a while, something that doesn't happen in the incremental type. Absolute encoders can be single-turn or multi-turn. Single-turn encoders measure the motion across 360º, whereas multi-turn encoders have an additional track to measure the number of revolutions done along the time.

Overall, absolute encoders are better for this type of system since these can estimate the position of the motion even after the system is turned off, eliminating the need for extra sensors [22]. An example of the output signals from an absolute rotary encoder can be seen in Figure 2.4, where A

[1] Automate.org - Encoder Basics for Motion Control Engineers https://www.automate.org/tech-papers/encoder-basics-for-motion-control-engineers

and B are the signals from the tracks and PRBS1 and PRBS2 are used to read the position of the tracks [23].

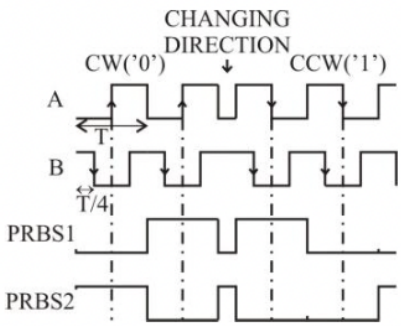

Figure 2.4: Output signals from a rotary absolute encoder [23]

2.2 Kinematics

2.2.1 Forward Kinematics

With the objective of planning the motion of the manipulator, its kinematics needed to be studied. The forward kinematics needed to be defined to describe the kinematic relationship between the pose of the robot's end-effect and its joints angles and to perform a kinematic control of the manipulator. An example of the structure of a 6 DoF manipulator can be seen in Figure 2.5 [24].

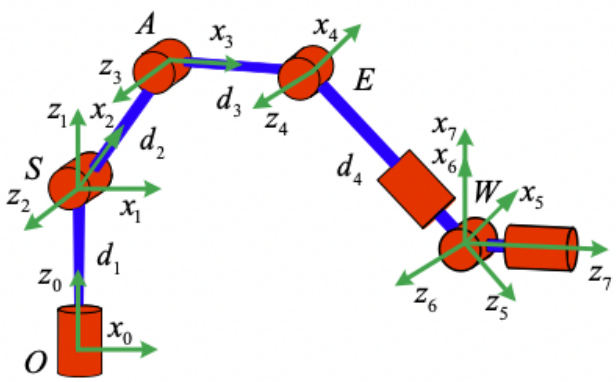

Figure 2.5: Example of a Robotic 6 DoF Structure [24]

With the structure of the manipulator well defined, the coordinate transform matrix of adjacent joints can be obtained using a homogeneous transformation matrix ${}^{i-1}_{i}\mathbf{T}$ seen in equation 2.4.

$$ {}^{i-1}_{i}\mathbf{T} = \begin{bmatrix} c(\theta_i) & -s(\theta_i) & 0 & a_{i-1} \\ s(\theta_i)c(\alpha_{i-1}) & c(\theta_i)c(\alpha_{i-1}) & -s(\alpha_{i-1}) & -d_i s(\alpha_{i-1}) \\ s(\theta_i)s(\alpha_{i-1}) & c(\theta_i)s(\alpha_{i-1}) & c(\alpha_{i-1}) & d_i c(\alpha_{i-1}) \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.4} $$

with c being the *cos* and s the *sin*.
When the values of all joint angles are defined, the homogeneous transformation matrix of the end-effector relative to the base can be obtained by multiplying the multiple homogeneous transformation matrices seen in equation 2.5.

$$ {}^{0}_{7}T = {}^{0}_{1}T^{1}_{2}T^{2}_{3}T^{3}_{4}T^{4}_{5}T^{5}_{6}T^{6}_{7}T \tag{2.5} $$

For this case, the D-H parameters would be the ones presented in Table 2.1, with θ_i being the angle from X_{i-1} to X_i, measured about Z_i, a_{i-1} being the distance from Z_{i-1} to Z_i, measured along X_{i-1}, α_{i-1} being the angle from Z_{i-1} to Z_i, measured along X_{i-1} and d_i the distance from X_{i-1} to X_i, measured about Z_i [24].

Joint	θ_i	a_{i-1}	$\alpha_{i-1}(\mathbf{rad})$	d_i
1	θ_1	0	0	d_1
2	θ_2	0	$\pi/2$	0
3	θ_3	d_2	0	0
4	θ_4	d_3	0	0
5	θ_5	0	$\pi/2$	d_4
6	θ_6	0	$\pi/2$	0
7	θ_7	0	$\pi/2$	0

Table 2.1: D-H parameters [24]

2.2.2 Inverse Kinematics

Inverse kinematics allows finding the appropriate joint angles to get a specific desired position and orientation of the end-effector. In most cases, inverse kinematics solutions are non-linear and hard to find. The main methods to solve the inverse kinematics of a robot arm are the geometric and algebraic ones [25].

2.2.3 Jacobian Matrix

The Jacobian matrix reflects the relationship between the linear and angular velocity of the end-effector and the joint velocities. By knowing the joint velocity, angles and the parameters of the arm, the end-effector velocity can be calculated in terms of the Cartesian coordinates using the Jacobian matrix [25]. This relationship is seen in equation 2.6.

$$ \begin{bmatrix} v \\ \omega \end{bmatrix} = J(\theta)\dot{\theta} \tag{2.6} $$

where ω and v are the angular and linear velocity of the end effector, respectively, while $J(\theta)$ is the Jacobian matrix and $\dot{q}$ the joint velocity vector.
Writing the previous equation in an expanded matrix form results in the matrix shown in equation 2.7.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{\alpha} \\ \dot{\beta} \\ \dot{y} \end{bmatrix}_{6\times 1} = \begin{bmatrix} J_{11} & J_{12} & \cdot & \cdot & \cdot & J_{1n} \\ J_{21} & J_{22} & \cdot & \cdot & \cdot & J_{2n} \\ \vdots & \cdots & : & \vdots & \vdots & \vdots \\ \cdot & \cdots & \cdot & \cdot & \cdot & \cdot \\ J_{61} & \cdots & \cdot & \cdot & \cdot & J_{6n} \end{bmatrix}_{6\times n} * \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \\ \cdot \\ \vdots \\ \dot{q}_n \end{bmatrix}_{n\times 1} \tag{2.7}$$

where the columns of the jacobian matrix represent the effect of the variation of each joint velocity on the end-effector velocities [24].

2.3 Motion Planning

Motion planning is the act of getting a robot to automatically determine how to move in space while avoiding collisions with obstacles. For a $W = \mathbb{R}^3$ world that contains a robot and obstacles and $O \subset W$ being the obstacle region, which has a boundary, the fundamental path-planning problem is: given an initial pose of the robot, compute how to gradually move into the desired goal position avoiding the obstacle region. The system's output will be a path through a set of transformations of the manipulator from start to finish.
The set of all rigid-body transformations that can be applied to the robot is called the configuration space or C-space. In this space, the algorithm has to find a solution to compute the path avoiding the obstacles. This notion of configuration space allows the dynamics to be expressed using a body's precise degrees of freedom. Considering a 2-D world and letting $q = (x_t, y_t, \theta)$ be called the configuration, a point (x,y) would appear at some (x', y') in the world, by applying a rotation or translation, given by the matrix 2.8 [26].

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta & x_t \\ \sin\theta & \cos\theta & y_t \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{2.8}$$

using a three-by-three homogeneous transformation matrix. With this, we can state that C-Space is a 3-D manifold which each element can be described as a $q = (x_t, y_t, \theta)$. For a 3-D model, the concepts are similar, only that there are three translation parameters x_t, y_t, z_t and the 3-D rotations are easily explained and used by using quaternions.
Considering the presented possibilities for C, the parts that are prohibited due to the collision space need to be defined. Let $A(q) \subset W$ represent a closed set of points in the world occupied by the robot A when the transformation to configuration q was performed. A configuration places the robot into collision only if the robot and the obstacle try to occupy at least one common point in

W. The set of all non-colliding configurations, represented in equation 2.9, is called the free space [26].

$$C_{\text{free}} = \{q \in C \mid A(q) \cap O = \emptyset\} \tag{2.9}$$

with the complement being called the obstacle region in C-Space $C_{\text{obs}} = C/C_{\text{free}}$.
With C-space defined, a basic path-planning problem can be defined: given a robot description A, an obstacle description O, a C-space C, an initial configuration $q_1 \in C$, and a goal configuration q_G, compute the path $\tau : [0,1] \to C_{\text{free}}$ with $\tau(0) = q_I$ and $\tau(1) = q_G$.
Since the motion-planning problem is in a continuous C-space, discretization must be done to get an algorithmic solution. Solutions such as using combinatorial planning and using sampling-based planning can be developed. The first constructs structures in the C-space to capture the information discretely, while the second searches the C-space in an incremental way, characterizing the C_{free} space in a discrete way.
With the problem defined, motion planning algorithms can be used to search for the best motion inside the space by incrementally probing and searching the C_{free} for a path, revealing it gradually. Algorithms such as RRT (Rapidly-exploring random tree) that aggressively probe and explore the space by expanding incrementally from an initial configuration are used to solve the plannings. In this case, each iteration extends the algorithm's tree with a leaf vertex and edge, with each edge being a collision-free path between the initial and the goal configurations. The algorithm picks a random point in C and tries to connect the tree to it by extending the nearest point in the tree, repeating this until reaching the goal pose. If the random point is in an obstacle, the edge travels up to the obstacle boundary, a characteristic seen in Figure 2.6 [26].

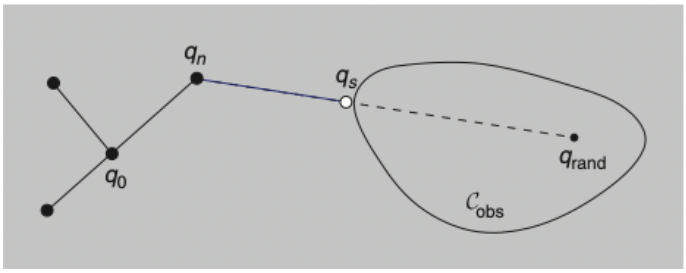

Figure 2.6: RRT - Case of random point appearing inside an obstacle boundary [26]

Alternatives to the RRT algorithm are, for example, the expansive space planner (ESP) that selects a vertex with a probability inversely proportional to the number of other vertices within a ball of a predefined size. The algorithm will then connect to a random configuration inside the predefined ball with an inversely proportional probability to the number of vertices inside a ball centred on the random configuration. The randomized potential field planner and the multiple-query approach probabilistic road map (PRM) are also used methods. The last chooses multiple random configurations, whose edges are formed by attempting to connect each configuration to all vertices within some radius. If the road map can be constructed satisfying accessibility and

connectivity parameters with a high probability, the path can be used for multiple initial-goal pairs.

It is essential to mention that sampling-based algorithms, such as RRT, usually achieve resolution completeness but may run forever if they cannot find the solution. In addition to these algorithms, path smoothing methods are usually used to correct crooked paths [26].

Chapter 3

State of the art

This chapter aims to provide the reader with an overview of the types of manipulators, motion planning algorithms, pick and place architectures, scheduling algorithms and object recognition and pose estimation algorithms used in underwater systems. Thus, the first phase explains a small description of the various underwater manipulators and their characteristics. A study about the multiple motion planning algorithms is performed, exposing their advantages, disadvantages and best use cases. An explanation of system architectures used in previous works for pick & place systems is also presented. In addition, a study about the scheduling algorithms is also performed based on the state of the art. At last, a critical review for each topic is written, comparing the multiple approaches and commenting on which fits better for this work.

3.1 Underwater Manipulators

Today's market of robotic manipulators, including the experimental and prototypes, is composed of mainly two types of robotic arms: those that run on oil-hydraulic and those that run on electric power [11].
Hydraulic Manipulators are controlled by the regulation of the hydraulic fluid flow. This flow is managed by the control of electro-hydraulic valves, which can be located in an external valve pack or integrated into the body of the manipulator. In the latter case, the system needs the valves to be oiled and pressurized to maintain an internal pressure higher than the external pressure. The pressure is maintained to avoid the ingress of water while using in underwater environments.
Since hydraulic systems do not need mechanical components such as gears and levers, they have the capability to produce an output torque bigger than the force applied. Therefore, these have a higher power-to-weight ratio compared to electric-powered ones. However, hydraulically driven manipulators have disadvantages such as the possibility of leakage of hydraulic fluid, price, need for complementary equipment, size, weight and poor positioning accuracy. The latter is due to the difficulty of the valve's control, which makes the use of this type of manipulator not ideal for automatic robotic functions or high-accuracy tasks. One example of this type of manipulator is the Kraft Raptor, developed by Kraft TeleRobotics, which can be seen in Figure 3.1 [11].

On the other hand, electric Powered manipulators are mainly built with brushless DC motors with harmonic drive gears, making it possible to have a high precision of motion and torque control. Its high accuracy is its main advantage compared to hydraulic manipulators. Therefore, these manipulators can find a use for precise subsea intervention operations. However, they often lack speed, reliability, strength or force requirements for industrial interventions [11]. One of the leading commercial manufacturers of underwater electric manipulators is ECA Robotics. Their manipulator 7E can be een in Figure 3.2 [11]. If used in underwater conditions, these may also have oil-filled actuators to stop water ingress, which also helps the lubrication and cooling of the system.

Figure 3.1: Kraft Raptor, Copyright © 2017 by Kraft TeleRobotics, Inc [11]

Figure 3.2: Eca robotics 7E, Copyright © 2017 by ECA Group [11]

Due to the importance of inspection and maintenance done with ROVs in underwater scenarios for the offshore industry, different approaches have been studied, such as the fixed and the free-floating base manipulation. In the first, the vehicle is docked before executing the task. Sensoring and visual techniques, such as a sonar system, are used to locate the objective. After, hydraulic grasps are used to dock the vehicle into the station to start the inspection process. Methods that have the a priori knowledge about the location of the objective arealso used in valve-turning tasks or plugging/unplugging a connector in an underwater scenario, as in the ALIVE (Autonomous Light Intervention Vehicle) AUV, explained in [27], a UAV that has the capability of autonomously docking before intervening in subsea structures. On the other hand, in the free-floating base manipulation methods, the UAV is floating in the water. The authors in [28] worked in an underwater manipulator with free-floating capabilities since the UAV does not dock before the task. This method may induce errors in the goal objective due to potential disturbances in the manipulator motion. However, techniques such as redundancy control for the kinematics can lead to the improvement of the results [29].

Until now, there have been some projects in the area of combining the exploring capabilities of a UAV with the operating capabilities of a manipulator such as the in TRIDENT project, which can be seen in Figure 3.3, developed by the Genova Robotics and Automation Laboratory of the University of Genova, equipped with the Girona 500 I-AUV with a 7 DoF manipulator, both working autonomously [30]. Additionally, there have been other projects such as the Holland I, which

can be seen in Figure 3.4, - an Irish Marine Institute-owned work-class ROV (remotely operated vehicle, equipped with two seven function manipulators and primarily used for scientific missions such as collecting samples and putting it in sample containers, and the ROV called Hercules that performs sediment core sampling [8].

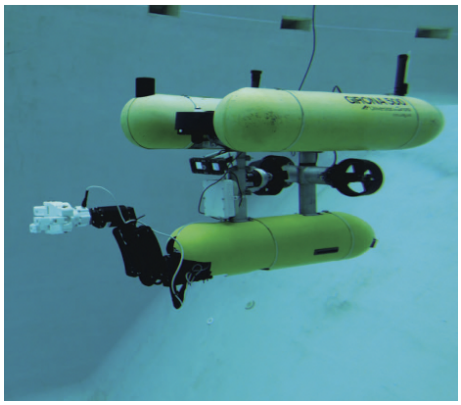

Figure 3.3: Trident Project: Girona 500 I-AUV with a 7 DoF manipulator [29]

Figure 3.4: ROV Holland I with two Schilling Orion 7P manipulator [8]

3.2 Motion Planning

Motion and Task Planners (MTP) plan the robot's navigation from start to goal position by defining a sequence of actions/movements. Autonomous Systems are able to perceive the environment through sensors by building environment maps. The planner can use these, together with constraints, to plan the sequence of steps of the path and, simultaneously, avoid possible obstacles [12].

Following the work done in [31], the robotic manipulation of objects can be divided into 4 phases: 1) Pre-grasp manipulation, 2) Grasp acquisition, 3) Post-grasp transport and 4) Goal. In the first phase, the arm follows a path to reach a position close to the object done and executed by the motion planner. Following, in the Grasp Acquisition phase, the gripper makes contact with the object. The third phase, Post-grasp transport, consists of moving the object to the goal position after it is grasped, consisting of the control of the movement of the manipulator and the grip's force. The last phase consists of the placement of the object in its goal position [31].

According to [32], a manipulator can be controlled through different autonomy levels:

- Joint Control - the operation of the manipulator is performed by the control of each joint within their range of motion. This is done manually by the operator, which is complex and slow. However, it allows the training of the operator.

- End Effector Jogging - the operation is done by moving the end effector without the manual control of each joint. This method has several drawbacks, such as no collision avoidance

or singularity. However, it may allow experienced operators a more accessible and effective end effector control.

- Step-Teleoperation - the operation is performed by the operator's teleoperation of the manipulator's centred base frame. This method allows a manual operation using the manipulator's collision and singularity avoidance method. Each teleoperation goal is given a collision and singularity avoiding motion plan that can be reviewed before the execution.
- Interactive Markers - ROS contains a functionality-focused control interface that allows the operator to drag-and-drop markers to provide a selection of either the velocity or the end effector. The user is shown a preview of the end effector pose before its execution.
- Pose Control with Motion Planning - This method uses packages, such as the MoveIt, that take care of the motion planning when the operator provides the initial and goal poses. The operator does not have the pose control during the process because the planning is done using pre-chosen algorithms. Advantages of this type of control are the level of autonomy and simplicity. However, these algorithms have some drawbacks because they can be unexpected or unintuitive due to their inseparability.

The method chosen for this work to control the manipulator was the Pose Control with Motion Planning due to its autonomy since this work's primary focus is the automation of the inspection and maintenance of wind farms through a manipulator. This section explains how and where multiple algorithms can be used to plan the movement of the manipulators.

3.2.1 Rapidly-exploring Random Trees (RRT)

One of the motion planning algorithms is the Rapidly-exploring Random Trees (RRT), an efficient and simple method for configuring high-dimensional spaces. This method was used in [33] to generate a path planning algorithm for a 7DOF manipulator of a humanoid robot and in [29] to control the 8-DOF GIRONA500 I-AUV. Its main advantage compared to other sampling-based algorithms is its searching and convergence speed and the fact that it retains the benefits of other algorithms if combined. In the second case, the algorithm was used along with Occlusion-Free and Collision-Free Constraints.

The Bidirectional Rapidly-exploring Random Tree (Bi-RRT) is a variation from the RRT algorithm that generates two trees instead of just one. This algorithm is based on a greedy algorithm that uses two separated trees of paths that start at the start and goal poses. The trees explore the space around them using a greedy heuristic until they meet each other in space. This variation was introduced to overcome the weakness of the classic RRT algorithm, which is the slowness of exploring when the sampling is not adapted to the problem [29, 33]. This algorithm was also used in [34] to plan the motion of an aerial pick and place task due to its flexibility and speed. This task used a multi-rotor along with a 2-DOF manipulator .

RRT also was used in [12] for the motion planning of a turtle robot. The author compared the distance and time of the Rapidly-exploring Random Trees (RRT), the Probabilistic Road-map

Method (PRM) and the Bi-RRT algorithms for a single source and multiple destination points. The sampling-based RRT was the best algorithm for execution time and solution length, followed by the Bi-RRT and PRM. It was found that the RRT performs better in collision-free plannings in a single source, while the Bi-RRT achieves better results with multiple sources. On the other hand, Bi-RRT and PRM performance decrease when the path has action constraints. The author chose RRT because it supports control sequences and fast exploration in higher dimensional spaces, which facilitates the use of action constraints.

3.2.2 Probabilistic Roadmap Method (PRM)

The Probabilistic Roadmap Method is an algorithm used widely in path planning. It consists of two phases: the learning phase and the query phase. During the first, a data structure called a roadmap is constructed. This roadmap can be characterised with a graph $R = (N, E)$. With N being a set of robot configurations that can be chosen, while E is the nodes or edges in the configuration space. The algorithm will use the roadmap to connect the random first configuration to the neighbours. This connection is made if the edge obeys a threshold, which otherwise would mean that the edge could not be a part of a feasible path. This is done until the roadmap is complete. In the second phase, the query, the start and goal configurations are connected to the graph, and the shortest path is found by analysing which path satisfies a given constraint. This constraint can be the probability of being free, or a weight-constrained shortest path can be used [35, 36]. This algorithm was used in [36] to perform the path planning of a UAV in a 3D complex environment. The authors combined the PRM with an octree algorithm of the 3D point cloud captured by a laser scanner. An octree algorithm was combined since the data from the laser scanner would make any path planning algorithm very slow due to its size. Its road-map from the octree and result can be seen in Figure 3.5 and Figure 3.6, respectively. In the work described in [35], to control a 3-DOF manipulator mounted on a mobile base, the author uses the probability of the edge being collision-free as a constraint. But since the algorithm becomes very complex, the author decided to use a Lazy-PRM that uses a k-shortest path algorithm in conjunction with a labelling algorithm to remove classes of paths, such as non-collision free paths.

Based on the previous work, we can conclude that the PRM algorithm is good when the task needs multiple queries since queries can be answered very quickly with the same roadmap. On the contrary, collision checking can make probabilistic planners very slow for single queries when minor changes occur. This is why Lazy PRM should is used in single queries tasks, which makes the planner fast by minimising the number of collision checks [37].

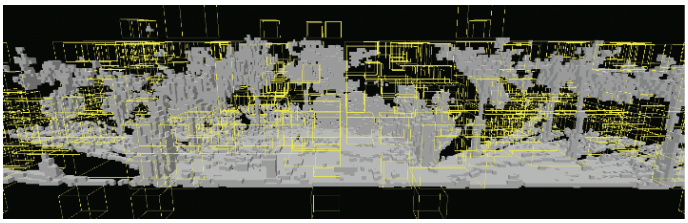

Figure 3.5: Road-map chosen from bounding boxes generated from the octree map [36]

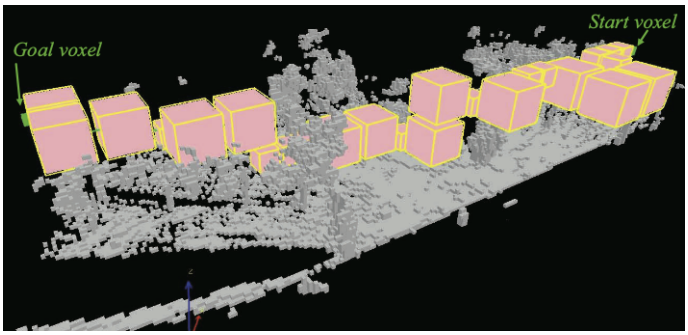

Figure 3.6: Final path connecting the start and goal [36]

3.2.3 Kinematic Planning by Interior-Exterior Cell Exploration (KPIECE)

Algorithms that solve motion planning problems using sampling-based tree planners are widely used in robotic systems. These systems explore the state space using a state tree and checking which motions are valid, starting from the origin until reaching the goal position. Thus, these algorithms depend on differential constraints due to the complexity of the systems they control. However, the author in [38] proposes using algorithms, such as the KPIECE method, that can be applicable in a variety of systems without relying on specific system properties, such as friction, mass and inertia. KPIECE can handle systems with complex dynamics with reduced run-time and memory occupancy by better using the information collected during the planning process. The method allows a decreased amount of forwarding propagation; thus, a lower number of states will be generated in the tree, leading to memory savings. The author divides this method into four components: 1) the use of projections from the state space to a discretized lower-dimensional Euclidean space to make the exploration easier, 2) keeping track of the explored space efficiently to allow the exploration of unexplored regions, 3) combining exploration information in regions to allow the section in a deterministic way and 4) evaluation of the progress of exploration. The tree of motions using KPIECE can be seen in Figure 3.7, where the states of the start of motions are the more significant vertices, and the intermediate states are the smaller vertices, which are not stored by the algorithm. The motion is achieved by forwarding integration. KPIECE was tested on three systems: a moving car in-plane, a blimp moving in space and a modular chain robot. The results proved that this method provided better answers with fewer steps and less computing time in systems such as these compared to techniques such as RRT and EST. However, in planning

the motion of a 7 DoF robotic arm, KPIECE does not provide better results than methods such as RRT-Connect or Single-query Bi-directional Lazy collision checking planner (SBL), which is an algorithm with the same expansion strategy as EST.

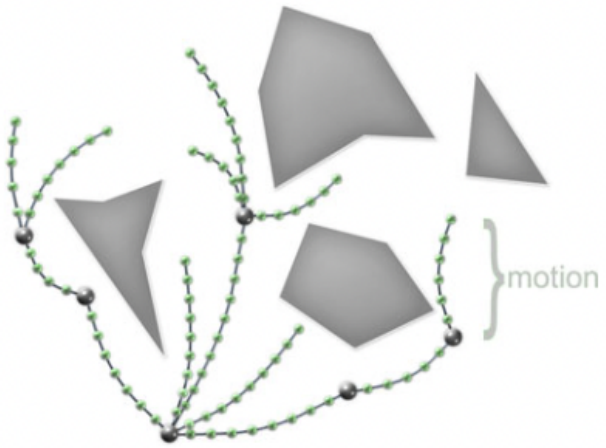

Figure 3.7: Trees of Motion using KPIECE [38]

3.2.4 Optimization-based Algorithms

Algorithms such as STOMP, CHOMP and TrajOpt are optimization-based algorithms. These are used for smooth trajectories in high-dimensional systems. The STOMP algorithm uses stochastic gradient methods to improve the trajectories, which are randomly developed around an initial seed. The second algorithm, CHOMP, generates sampling or search-based methods using functional gradient techniques. These will iteratively improve an initial trajectory. This type of method is susceptible to getting stuck in an infeasible local minimum due to its dependency on the initial seed [39].

3.3 Pick & Place

The authors in [40] propose an architecture for a pick & place system represented in Figure 3.8. The system is divided into three components: Simulation Environment, Interface Node and Agent. The simulation environment is the environment developed in the simulator that a real environment will later replace. Following right after is the interface node, which serves as a bridge between the simulation environment and the agent. This node allows a continuous stream of sensory data from the environment to the agent, such as camera images and the effector's feedback. In addition, it allows using a set of functions that allow interactions with the simulated environment. These include inverse and forward kinematics, joint motion planning and operation mode control.
Furthermore, the agent can also be described in three components: the sensory data adapter, the

solver and the agent. The first is the module that converts the joint's state information to a transformation tree that allows the user to keep track of the frames over time. The second is the decision-making module, which defines a plan to solve the desired task using the available skills and the initial environment analysis.

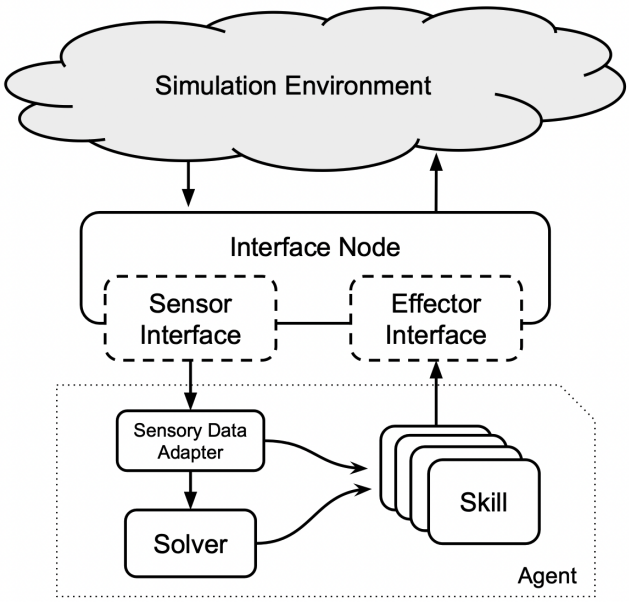

Figure 3.8: Proposed system architecture [40]

When working with pick & place scenarios, developing a list of agent skills is common. A skill can be defined as the capability to perform a specialized action with a semantic meaning. In this scenario, the action implies a physical motion by the manipulator, performed by the movement of one or more actuators. The author develops a set of skills that represent specific actions, where each one can be used for different tasks or more complex skills. In the author's case, the pre-defined skills were: simple move, which changes the pose of the manipulator; move XY Axis, where the x and y values are used to move the arm; pick an object, where the system moves the manipulator to grab an object; place object and push object. When performing a specific task, the algorithm focuses on using the lowest number of skills possible [40].
The authors in [41] set a similar definition for its control architecture by defining the high-level capabilities of the robot. This method divides the system into primitives, skills, tasks and missions. The primitives are the hardware program blocks characterized by the sensing and actuating capabilities. Each primitive performs a unique operation. The skills are the robot's abilities, each one with its input parameters, executing instructions and way to verify if the execution was successful. This concept can be seen in Figure 3.9. Tasks are the set of skills used to reach a goal. These define the input parameters of skills, assess the robot's state variables, and change these with the execution of the tasks. For last, the missions are sequences of tasks to achieve specific

goals. These capture high-level behaviours obtained after the execution of tasks and define the input for the next tasks.

Figure 3.9: Concept of skills [41]

Dividing a robot's system like this makes it possible to describe complex behaviours more systematically and with more flexibly. In addition to this, it facilitates the reusability of these systems. This hierarchy for a robotic system is represented in Figure 3.10.

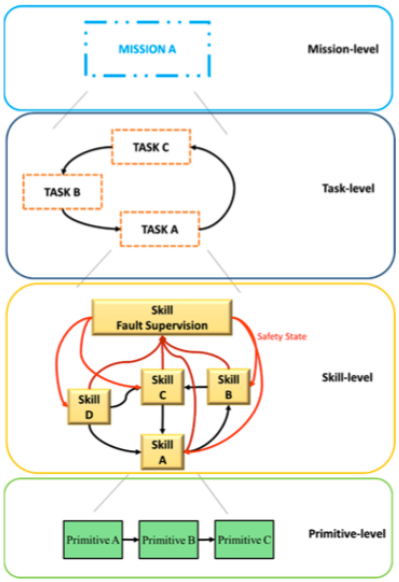

Figure 3.10: System hierarchy [41]

3.4 Scheduling

There are multiple types of pick & place tasks, such as stacking, where the robot stacks parts that come off a production line or from previous operations; sorting, where the robot sorts different objects into separate conveyors; packaging, consisting of a robot putting objects into a packaging container and others such as bin picking, quality control, assembly and scheduling [1].

[1] Pick and Place Tasks for a Robot - https://robodk.com/blog/10-perfect-pick-and-place-tasks-for-a-robot/

The task that this work will focus on, scheduling, seeks to plan a time task schedule that minimizes the defined parameters. There are methods to solve scheduling tasks such as exact, approximate, heuristics and meta-heuristics methods [42]. Usually, the scheduling of the jobs has several measures of performance, such as minimizing the sum of completion time of all the jobs, the weighted sum of these, minimizing the maximum lateness and such [43]. The work performed in [44] uses a polynomial-time approximation algorithm to solve a routing open shop problem with a metric travelling salesman problem. Heuristic methods can also be used successfully to solve open-shop problems with preemption by minimizing the average completion time [45]. Other works also prove good results with heuristic methods [46]. A Gant chart for a 4x3 open-shop problem is seen in Figure 3.11.

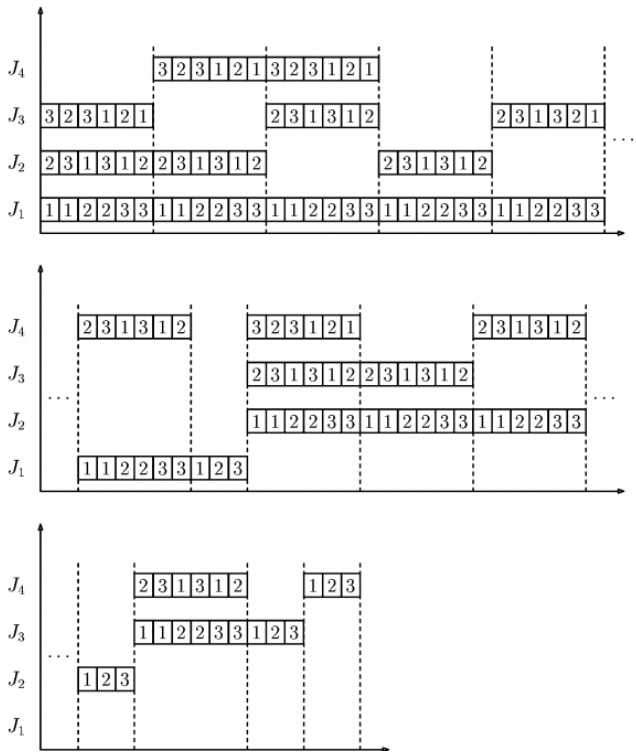

Figure 3.11: Gant Chart for all states for a 4x3 open-shop problem [23]

The work performed in [47] presents the use of hybrid genetic-based heuristics to solve a problem with a dimension greater than 4x3. When working with more significant scenarios with a job size more prominent than 10, the hybrid genetic-based heuristics had better results than classical meta-heuristics.

Standard greedy scheduling rules such as FIFO (first in, first out) and SPT (shortest processing time) are used most when working with static queues. These operate in $O(n)$ time for selecting a single picking candidate, where n is the number of items accessible. However, the overall solution is generally sub-optimal in efficiency, leading to a 40% loss in efficiency. These rules can be refined by using local optimum criteria and heuristic considerations according to the particularity

of the problem to enhance the results. The main advantage of the FIFO rule is that its complexity is constant, making the computational cost low since it does not require a complex sorting of elements. When working with simple underload cases, this algorithm can be proven optimum. On the other hand, it can be inefficient and sub-optimal with overload and more complex situations.
The SPT rule states that the first object that will be processed is always the one with the shortest processing time. Based on previous works, this strategy has better results than FIFO in most cases, and there are ways to improve both these algorithms by using more constraints to the specific problem [48, 49].
The authors in [50, 51] present an open shop scheduling problem with sequence-dependent setup times to minimize make-span. The following algorithms are presented: two new meta-heuristics called multi-neighbourhood search simulated annealing and hybrid simulated annealing, two constructive heuristics named LTMPT and LTRMPT and three additional meta-heuristics: the variable neighbourhood search, simulated annealing, and genetic algorithm. These were evaluated in randomly generated test instances. The paper in [52] proposes a path strategy for energy-harvesting NVPs considering the power switching overhead. The scheduling problem is solved with an offline heuristic scheduling strategy with Tabu search (TS) to solve the problem. Besides this, an online performance-aware scheduling algorithm is proposed to schedule the tasks dynamically. The results showed that the chosen heuristics algorithms reduced the execution time by 24.8% with the offline strategy and 22.5% with the online one compared to previous works. As seen in Figure 3.12, the online method uses the optimized parameters generated from the offline strategy to dynamically decide the best scheduling results according to online parameters.

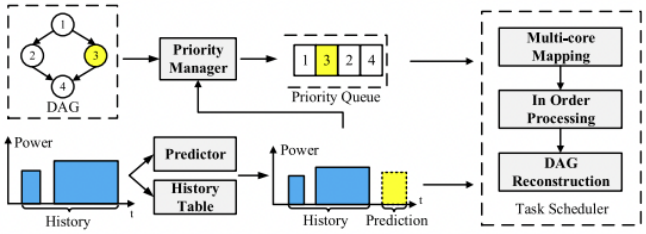

Figure 3.12: Online Performance-aware task scheduler [52]

3.4.1 Travel Salesman Problem (TSP)

This work addresses the scheduling problem of the Travel Salesman Problem (TSP) type to determine the lowest cost to travel through a series of points, which in this case belong to a pick & place task. By working with n nodes and evaluation criteria, such as distances or costs between them, TSP aims to describe the best route option that can be covered without repetition by considering the optimum value [53]. The work in [54] uses an optimal jerk confined planning algorithm under a greedy rule to generate a smooth function, improving the machining quality. In [55], the author seeks to solve a travelling salesman problem, starting from city number 1, which must travel to

all N cities one at a time. The author uses chromosomes, genetic operators in each generation, ordinary mutation, selection operators and a greedy crossover operator since the path length is known in advance, and there is no need for a primary binary coding.

The work presented in [56] uses a greedy genetic algorithm (GGA), combining greedy with genetic methods to solve a TSP problem. After the genetic operations, such as crossover and mutation, only the better element will replace the parents, using a greedy method. This method proved to be better than standard genetic algorithms (GA), as GA is unable to guarantee achieving the optimal solution to TSP problems. In addition, the author concludes that using accurate heuristics can improve the performance of genetic algorithms. Furthermore, algorithms such as the particle swarm optimisation (PSO) algorithm and ant colony (AC) algorithm have been used in path search processes. The main problem with these is that they are generally computationally expensive and often fall into some local optimum, even though they are easy to implement [57].

The authors in [57] state that interpolation methods apply smooth curves to the paths, which is an advantage if speed and acceleration are important parameters to consider. On the other hand, they do not have good results with irregular paths, making their application more valuable to simple path problems. Such algorithms are the B-spline curve and Dubin's curve.

When focusing on geometric search algorithms to obtain the shortest path between nodes, the A-star algorithm is a search algorithm based on Dijkstra's method. It uses a heuristic function to estimate the distance from the current to the final node. This algorithm often has better results than the first while having a good real-time performance but is prone to fall into local optimums. In addition to the traditional method, there are improved methods such as the Bidirectional A-star algorithm, the Breadth-First-Search (BFS) and the Depth-First-Search (DFS). The first simultaneously searches for the start and goal node, the second uses queues, and the third uses recursion. The authors in [57] also present the Geometric A-star algorithm to improve the irregular path generated by the generic A-star algorithm. The results of the conducted experiments can be seen in Figure 3.13, with some of the paths seen in Figure 3.14.

Map	Path parameters	A-star	BFS	Dijkstra	Bestfirst	Bidirectional A-star	DFS	Geometric A-star	Reduce scope/%
6	Running time/s	316.334	322.962	316.334	396.642	316.334	394.83	295.142	6.7~25.5
	Number of nodes	131	131	131	136	131	198	109	16.8~44.9
	Number of turns	36	33	27	32	30	1(exclude)	27	0~25
	Max turning angle	45°	135°	45°	135°	45°	90°	45°	0~33.3
	Number of expansion nodes	2246	5936	6047	136	1611	198	109	19.9~98.2
	Total distance/m	158.167	161.481	158.167	197.851	158.167	197.415	147.571	6.7~25.5

Figure 3.13: Comparison of the results of path planning algorithms [57]

The author in [53] seeks to solve a TSP problem by using a Greedy Randomised Adaptive Search Procedure (GRASP) method. Grasp is a meta-heuristic method composed of two steps: first, the construction of greedy procedures in order to identify feasible solutions to the problem

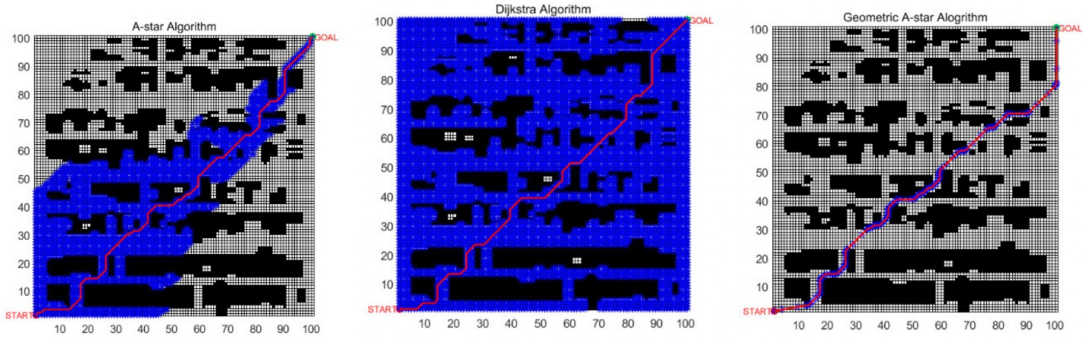

Figure 3.14: Paths of the path planning algorithms [57]

based on the best solution value; second, the method performs a heuristic procedure that improves the quality of the solutions at each iteration in a randomised sequence. The author also used a 2-Opt method to perform the local search phase. It was concluded that the results demonstrated the capacity to fulfil the purpose of the problem, even though the algorithm can take a long time if the number of iterations is high. The distance of the result in relation to the number of iterations can be seen in Figure 3.15.

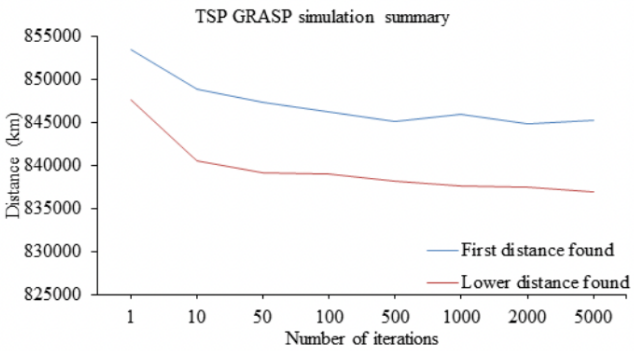

Figure 3.15: Simulations results for TSP instance using GRASP [53]

In addition, the work in [58] also presents a randomised adaptive version of the nearest neighbour, the GRASP algorithm. The author uses two implementations of LK as local search procedures to generate better results: LK-ABCC and LK-H, concluding that LK-ABCC presents lower runtimes than LK-H for all the investigated instances. However, the differences are minimal. The results showed improvements of 76% and 65% compared to the normal LK-ABCC and LK-H algorithms, respectively.

3.5 Critical Review

This section tries to summarize the state of the art performed before. In addition, it aims to compare and review the methods described critically. A selection of the techniques that will be used for this work is also made based on a priori results.

3.5.1 Manipulators

As stated in 3.1, even though hydraulic manipulators have the ability to produce higher torque and force, an electric-powered manipulator was the type of manipulator chosen for this work. This is because there is a high need for precision and accuracy to perform maintenance and inspection of wind turbines, which cannot be obtained with a hydraulic manipulator. In addition, using an HM would mean that the AUV to which the arm is attached would have to carry multiple heavy and oversized components such as valves or hydraulic tanks. Overall, an electric-powered manipulator was chosen due to its accuracy and a small need for complementary equipment.

3.5.2 Motion Planning

Ragel, R. et al. in [59] compared multiple motion planning algorithms for an Unmanned aircraft system (UAS) with an aerial 6DOF manipulator using the MoveIt framework. The work done was to 1) extend the manipulator, 2) pick up a bar, 3) place the bar above two bases and 4) contract the arm. By analysing only the arm motion planning the results are shown in Figure 3.16 and Figure 3.17 . The algorithms tested were:

- RRT - Rapidly-exploring Random Trees
- RRT* - Asymptotically optimal version of RRT
- RRT-Connect - Bidirectional version of RRT
- PRM - Probabilistic Roadmap Method
- PRM* - Asymptotically optimal version of PRM
- EST - Expansive Space Trees
- KPIECE - Kinematic Planning by Interior-Exterior Cell Exploration
- BKPIECE - Bi-directional KPIECE

The RRT* and BKPIECES algorithms were omitted in Figure 3.16 because their average planning time was 35 and 23 seconds, respectively. These values are very high compared to the other methods. The RTT and RRT-connect are the fastest methods with very similar planning times, with PRM and PRM* following. EST and KPIECE come next, with BKPIECE being the slowest, excluding RRT* and KBPIECES. This work concluded that RTT and RTT-connect are the best

algorithms for a 6DOF manipulator motion planning with collision avoidance. RTT-connect combines the random fast exploration from the RTT method with the Connect heuristic that connects the goal with the pose from each point with a fast solution.

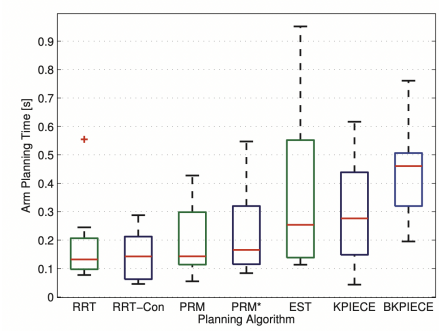

Figure 3.16: Motion planners comparison: Arm planning time per simulation [59]

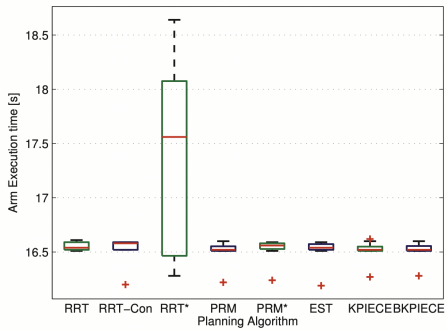

Figure 3.17: Motion planners comparison: Arm execution time per simulation [59]

As stated in 3.2.3, the authors in [38] compared the motion planning time using different algorithms of a 7 DoF arm called PR2 arm, from Willow Garage. The results are shown in the second column of the Table in 3.18. RTT-Connect and SBL were the methods that performed the best, followed by LBKPIECE, a lazy bidirectional implementation of KPIECE. Basic KPIECE and EST come after, with RRT being last in terms of planning time.

Algorithm	Arm Plan Time(ms)	Rigid Body Plan Time (ms)
RRT	456	3248
EST	187	3907
KPIECE	166	698
RRTConnect	21	1508
SBL	29	3943
LBKPIECE	37	1146

Figure 3.18: Planning Time of different Algorithms in a 7 DoF arm [38]

3.5.3 Scheduling

As seen in 3.4, there are multiple methods to solve scheduling problems, such as exact, approximate, heuristics and meta-heuristics methods. We also concluded from previous work that exact methods have worse results for complex problems than heuristic and meta-heuristic methods [45, 46], while having good results for static queues [48, 49]. Methods such as FIFO and SPT are usually improved by using constraints specific to the problem. In addition, when working with job sizes more significant than 10, meta-heuristics can have good results [47]. In conclusion, heuristic and meta-heuristic methods have the best results when working with more complex problems.

Chapter 4

Developed Work

This chapter presents the developed work for this thesis in detail. The methodology is presented along with a short description of the different layers used for this work. Afterwards, the used tools are presented, and the implemented architecture is exposed, showing its multiple nodes. This is followed by an explanation of the stages that were developed to perform the pick & place tasks. The way the motion planning was performed is also described with the respective planning algorithms. Additionally, a description of the implemented scheduling algorithms, along with their pseudo-code and characteristics, is presented. Finally, the evaluation metrics are presented in the chapter's final section.

4.1 Methodology

Considering the literature review presented in chapter 3, the algorithms that will serve as the basis for developing the solution were chosen. In order to plan the manipulator's trajectory, the algorithms RTT-Connect, BKPIECE, and PRM will be used, based on previous works. On the other hand, Dijkstra's greedy algorithm, GRASP (Greedy randomized adaptive search procedure) algorithm and the A* algorithm were chosen to solve the Travelling Salesman Problem. The first is a greedy algorithm, while the second and third are meta-heuristic and heuristic methods, respectively. The latter uses a heuristic cost to estimate the distance from the current point to the destination. On the other hand, the GRASP method relies on randomization to increase the diversity of the search [60].
Thus, it is possible to conclude that the proposed solution contains three different layers, represented in Figure 4.1:

- **Motion Planning** - where all needed trajectories for the movements of the manipulator are planned using motion planning algorithms;
- **Scheduling** - where the scheduling algorithms will choose the best order for the movements;
- **Execution** - where all the movements will be executed in the chosen order.

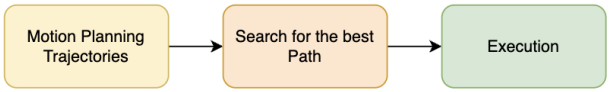

Figure 4.1: Developed Model divided into three layers

4.2 Used Tools

This section introduces the primary tools used to perform this work, as well as some of their functionalities, advantages, challenges and libraries. The first presented tool is Robot Operating System (ROS)[1] and the second is the Moveit Motion Planning Framework.

4.2.1 ROS - Robot Operating System

ROS was the meta-operating system used in this work. This open-source tool provides libraries, plugins and communication frameworks for the development of robotic applications. Furthermore, it has software libraries to work with navigation, visualization of movement during simulation and motion planning, which work independently. The libraries use a message passing layer that allows the connection between ROS nodes. These are managed via the ROS master node and are not based on any specific programming language [12, 31].

4.2.2 MoveIt

The MoveIt[2] framework covers various functionalities such as mobile manipulation, a forward and inverse kinematics solver, planning techniques and collision-detection methods through a plug-in-based mechanism. It can be used along with ROS and with a wide variety of third-party libraries such as Kinematics and Dynamics Library (KDL)[3], the Open Motion Planning Library (OMPL)[4] and the Collision Check Library (FCL)[5]. The first develops an independent application framework for modelling and computation of kinematic chains, such as robotic or biomechanical human models. The Open Motion Planning library is a sampling-based motion planning algorithms library. The latter, the Collision Check Library, is a library to perform multiple tasks such as collision detection, distance computing and tolerance verification [29, 31].
The modelling in the MoveIt framework uses unified robot description format (URDF) and semantic robot description format (SRDF) extensible markup language XML-based files. The first defines the shape, the collision properties and the kinematics parameters of the components of the system. These properties and parameters can be imported from computer-aided design software like FreeCAD or AutoCAD. At the same time, the SRDF describes the set of joints and links that

[1] ROS https://ros.org
[2] MoveIt https://moveit.ros.org
[3] KDL https://www.orocos.org/kdl.html
[4] OPML https://ompl.kavrakilab.org
[5] FCL http://wiki.ros.org/fcl

form a single entity. It is also possible to specify the entities' size, shape and colour for simulation purposes. Using the entity's descriptions along with a kinematics solver and planner makes it possible to perform planning and manipulation tasks [29, 31].

Based on the experience of past projects, MoveIt has multiple advantages:

- Easy definition of the kinematics by using the URDF file along with the KDL library, allowing the avoidance of the manual writing of the definition of the forward and inverse kinematics equations;
- Easy planning of collision-free paths through the definition of the collision geometry along with the use of the FCL library and the path-planning algorithms of the OMPL library;
- Reduced development time due to multiple libraries and pre-made software.

However, the platform can have multiple challenges, such as under-actuating the manipulator's degrees of freedom. In some cases, the algorithms can leave some DOFs non-actuated, resulting in a less complex but less efficient motion [29, 12].

The framework comes with the integrated OMPL library, which contains implementations of many path-planning algorithms. Such algorithms are Bi-directional KPIECE (BKPIECE), Path-Directed Subdivision Trees (PDST), Probabilistic Roadmap Method (PRM), Rapidly-exploring Random Trees (RRT), Expansive Space Trees (EST) and more. Furthermore, it also contains variants of these algorithms [29].
When working with pick & place scenarios, MoveIt provides multiple options for each task for grasp poses, which are ordered depending on the user's quality metrics. It works with three main sequential stages for pick & place plannings: 1) reachable and valid pose, 2) approach and translate and 3) plan. During the first stage, the collision-free pose accessibility is checked. Afterwards, in the second step, the framework computes the Cartesian path between the grasp pose and the pre-grasp pose. The collision filter between the arm and the object to grasp is disabled at the next stage. Afterwards, in the last stage (plan), the manipulation plan is computed using the previously computed trajectories [29].

With the Collision Check Library (FCL), the MoveIt planner can plan collision-free trajectories. It is possible to categorize the two types of obstacles:

- **A priori known objects** are the ones that already have their dimensions and locations defined. These are usually manually created by the operator. The simulator can represent these through a pre-made geometric shape set that represents simple obstacles. Examples of this type of obstacle can be seen in the developed environment represented in Figure 4.2;
- **Unknown** objects are the ones represented by sensor data. Octomaps, for example, can be integrated with MoveIt, since the framework includes an octree that makes it possible to map the voxels of the environment.

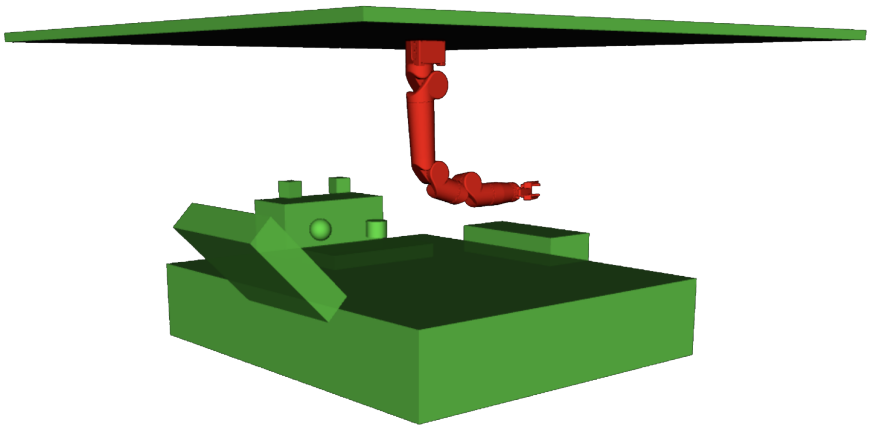

Figure 4.2: Simulated scenario with obstacles

4.3 Architecture

The developed system's architecture, which can be seen in Figure 4.3, is divided into three main components: Environment, Interface and Agent. The interface node serves as a bridge between the environment and the agent. The agent can be described in three components: the solver, the skills and the image processing module. The first node is the decision-making module, which defines a plan to solve the desired task using the available skills. The skills are the actions that imply physical motion in the manipulator, performed by the movement of the actuators, a calculation, or a transfer of data/variables. The skills can include moving to the object position, picking up the object, moving the object through the available space, putting down the object, adding the objects to the simulator or even saving all the plans. It is essential to mention that the communication between the manipulator and simulator is performed through the skills and never through the solver. The Image Processing module receives the images from the camera and performs object identification. The solver node will use the information from the image processing module to decide the sequence of skills to use. In addition, the interface node has a camera module to send the data from the environment to the agent. For this work, the image processing module was not used, and the poses of the objects were used as pre-known information. The solver had the positions of the objects beforehand in order to make the decisions.

4.3.1 Node Architecture

Inside the agent, the interaction between the two primary nodes: the solver and the skills node, seen in Figure 4.4, is a crucial factor in understanding the architecture. As stated before, the first is the one that makes all the decisions using the information received by calling the functions of the skills node.

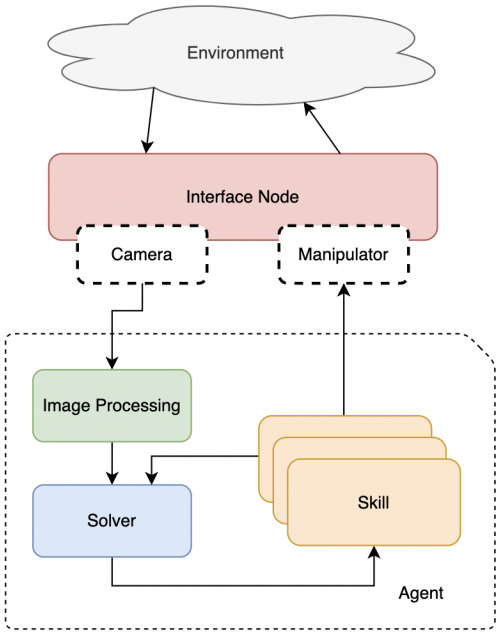

Figure 4.3: Proposed system architecture

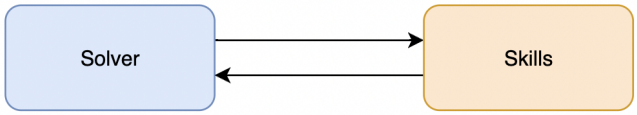

Figure 4.4: Node Architecture

The skills node contains actions, calculations and data transfers. The main skills can be described as:

- *Plan* - Plans the points of the trajectory of the arm for certain joint positions or Cartesian coordinates given by the solver while taking account of collision objects and the manipulator's restrictions;
- *Exec* - Executes a movement of the arm with a trajectory given by the solver, planned beforehand. The execution can be done on the simulated or real manipulator, depending on the stage of the program. If the execution is on the simulated one, the skill will directly connect with the MoveIt framework to move the manipulator. If the object is moving the real manipulator, the skill sends the trajectory to the Bravo's controller after verifying if the simulated movement was well performed;
- *ExecHand* - Executes a movement on the gripper, with the movement being sent by the solver. As in the previous skill, if the movement is to be done in the real manipulator, the action will be sent to the Bravo's controller only after the movement occurs in the simulator.

In addition to this, other skills have the objective of creating the structural objects and giving them their characteristics, such as pose and size (tables, ceiling and bases), creating and defining the pose and size of the pick objects (cubes, hexagon and cylinder) and adding all the objects into the simulated environment. In addition, a skill to calculate the values for each metric for a particular trajectory was developed (*SendMaximumValuesMetric*), having the objective of sending the values of the metrics to the solver.

The solver contains functionalities regarding defining parameters and restrictions, decision making, establishing connections, storing and reading the information, normalizing the metrics values, executing the scheduling algorithms and more. The functionalities can be described as:

- Defining the objects and their parameters;
- Defining the bases and their parameters;
- Establishing connection with the skills node;
- Communicating with the skills node to add the objects into the simulator;
- Communicating with the skills node to plan all the trajectories;
- Storing the plan's trajectories and other information into files;
- Communicating with the skills node to receive the maximum and minimum values for each metric;
- Storing the maximum and minimum values for the metrics into a file;
- Reading all the plans and the metrics values;
- Calculating the normalized values for the metrics using the maximum and minimum values;
- Defining the weights for each metric;
- Executing the scheduling algorithms: Dijkstra's greedy, GRASP or A*;
- Communicating with the skills node to execute the final movement.

4.4 Pick & Place

For the pick & place tasks, an architecture of the consecutive steps that have to be taken by the manipulator was developed and can be seen in Figure 4.5, considering only one object and one final destination. The process is divided into two types of movements: manipulator's movements and gripper's movements, represented in blue and orange, respectively.

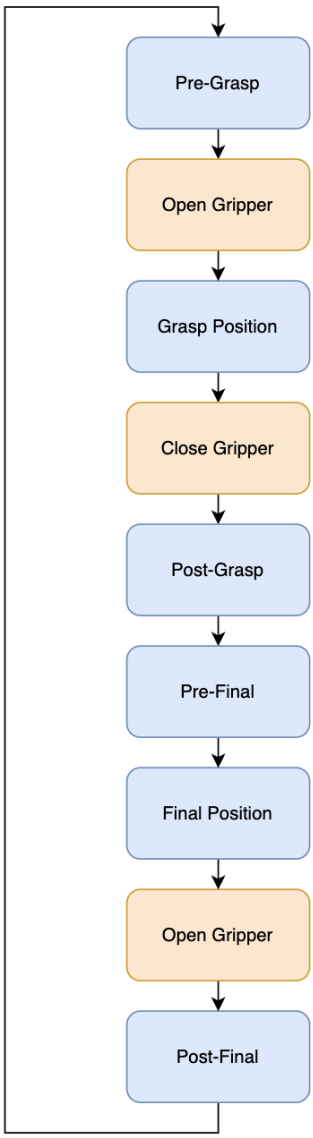

Figure 4.5: Pick & Place - Manipulator steps

For the pick & place tasks, an architecture of the consecutive steps that have to be taken by the manipulator was developed and can be seen in Figure 4.5, considering only one object and one final destination. The process is divided into two types of movements: manipulator's movements and gripper's movements, represented in blue and orange, respectively.

The task can be divided into [61]:

- Pre-Grasp - end-effector pose that is an offset from the object;
- Open Gripper - pose for the gripper where it is open and ready to grab an object;
- Grasp - pose of the end-effector in which its gripper is in position to grab the object;

- Close Gripper - pose for the gripper where it is grabbing an object;
- Post-Grasp - end-effector pose after it moved away from the surface with the object grasped;
- Pre-Final - end-effector pose that is an offset from the final position for the object;
- Final Position - end-effector pose in which the object is in the desired final position;
- Post-Final - end-effector pose after it moved away from the final position.

Since the objective is to move all four objects to their respective final bases, the scheme represented in Figure 1.3 represents the possibilities of movements for one iteration of the problem. It is essential to mention that an object can only be moved to a base with the same number of vertices as the object. With this restriction in mind, we can see that the cubes can be moved to two different bases, while the cylinder and hexagon can only be moved to their bases. After moving one object to its respective base, the next movement can either go to the starting point or another object, depending on the decision of the scheduling algorithm.

4.5 Reach Bravo 7 Manipulator

The Reach Bravo 7[6], seen in Figure 4.6, is the manipulator used during this work, a 7-function manipulator made for Inspection Class Vehicles developed by Blueprintlab. This arm was designed for advanced underwater applications and can easily be fitted into an ROV (Remotely operated underwater vehicle). In addition, the manipulator can be fitted with multiple end-effectors such as parallel jaws, quad jaws or even cable cutters depending on the desired task.

Figure 4.6: Reach Bravo 7 by Blueprintlab [7]

The manipulator has a reach of 0.9 meters with a 10kg full reach lift and a 20kg max lift capacity, more than enough for the pick & place task performed. Its end-effector accuracy is smaller than 1cm, and the arm has three control modes: Position, Velocity and Cartesian (XYZ), with

[6]Reach Bravo 7 https://blueprintlab.com/products/manipulators/reach-bravo/

the control being done through RS232, RS485 or Ethernet connection. Its working temperature is between -10ºC to 35ºC, with a depth of 300MSW. The configuration and its limits can be seen in figures 4.7 and 4.8, and the limits for its joints can seen in Table 4.1.

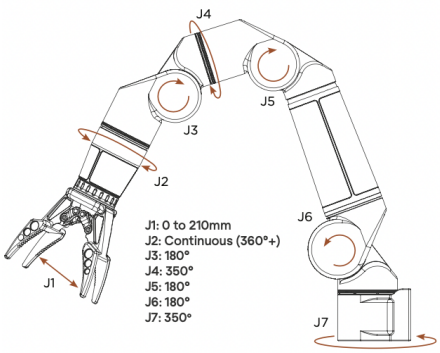

Figure 4.7: Bravo 7 Joint Configuration

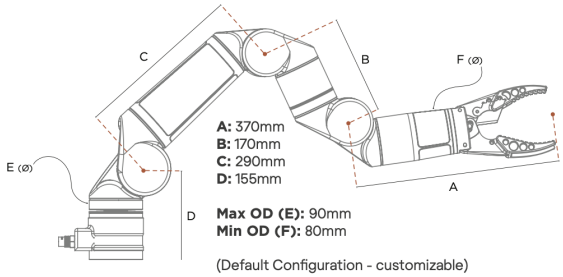

Figure 4.8: Bravo 7 Distances between joints

Joint	J1	J2	J3	J4	J5	J6	J7
Limits	0-210mm	continuous	180º	350º	180º	180º	350º

Table 4.1: Reach bravo 7 joints limits

Since $J7$, the base joint, is not continuous, the working space is not an entire sphere, as seen in Figure 4.9. The chosen environment and the movements of the arm considered this limitation to avoid unnecessary long trajectories.

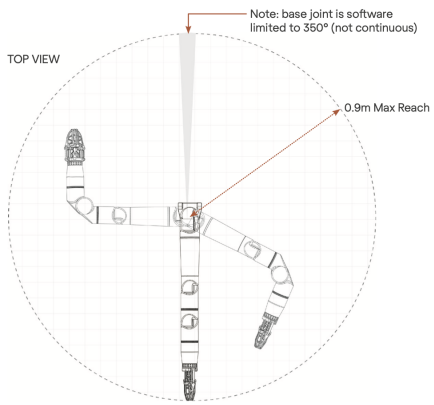

Figure 4.9: Reach Bravo 7 Work Space

The Table 4.2 shows the Denavit–Hartenberg parameters for the Bravo 7 where $\theta_x = tan^{-1}(\frac{5.2}{395.55})$. In addition, Figure 4.10 shows a representation of the bravo 7 joint frames. The x, y and z axis are represented in red, green and blue, respectively.

[7]Blueprintlab https://blueprintlab.com

Link	**r (m)**	θ**(rad)**	**d (m)**	α**(rad)**
1	0.1074	$\theta_0 + \pi$	0.046	$\pi/2$
2	0	$\theta_1 - \pi/2 + \theta_x$	0.293	0
3	0	$\theta_2 - \pi/2 - \theta_x$	0.0408	$-\pi/2$
4	-0.160	θ_3	0.0408	$-\pi/2$
5	0	θ_4	0.0408	$-\pi/2$
6	-0.2235	θ_5	0	$\pi/2$
7	0	$-\pi/2$	0.120	0

Table 4.2: Standard Denavit–Hartenberg parameters for Bravo 7

where d is the offset along previous z to the common normal, θ is the angle about the previous z from old x to new x, r is the displacement along x, and α is the angle about the common normal, from old z axis to new z axis.

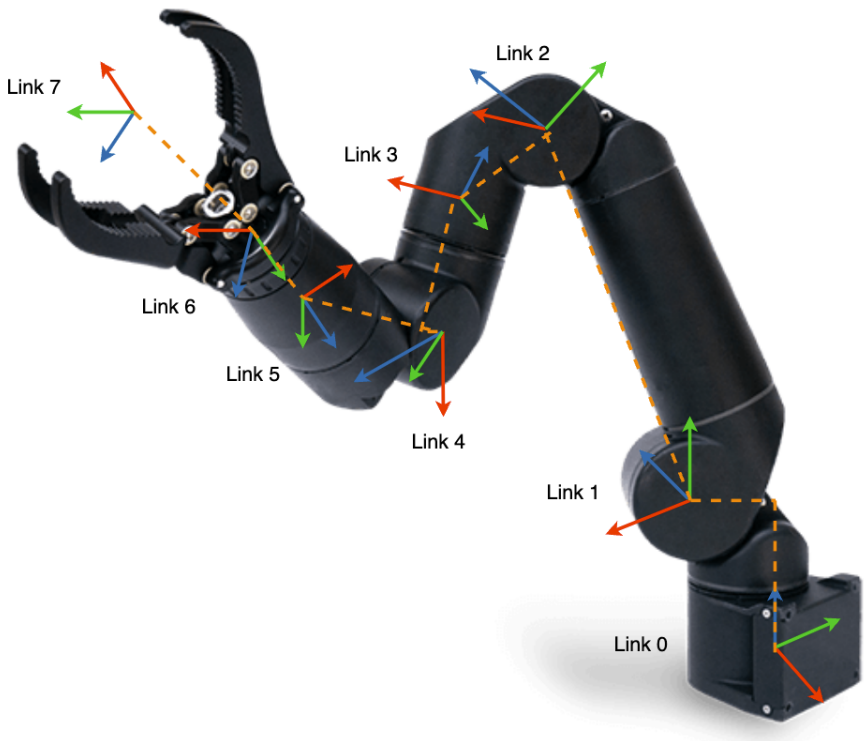

Figure 4.10: Reach Bravo 7 Joint Frames [8]

4.6 Bravo 7 controller

In order to perform the tests in the Bravo 7, the controller seen in Figure 4.11 is used. The controller receives the trajectory to be performed, calculates the individual velocities and accelerations

[8] Blueprintlab Bravo 7 https://blueprintlab.com/products/manipulators/reach-bravo/

for each joint, and moves the actuators of the manipulator. When the controller receives the trajectory points, the desired behaviour is that all joints arrive at the desired position simultaneously. For this to happen, the system calculates the velocities and accelerations considering the distance the joints need to move, resulting in all joints finishing their movements simultaneously.

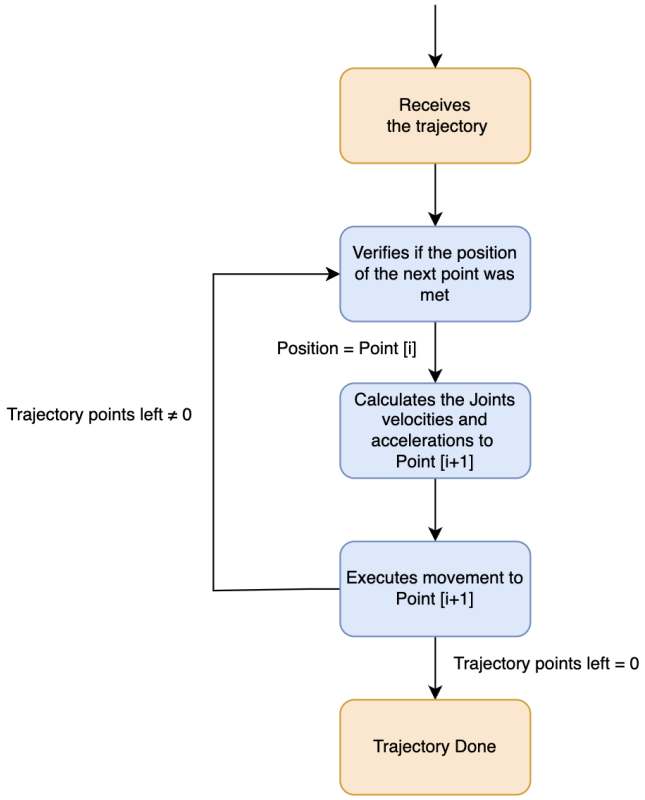

Figure 4.11: Diagram of the controller

The controller uses the encoder's values and a safety threshold to verify if the manipulator's position has already reached the goal point. If this condition is not met, the subsequent movements cannot be performed for safety reasons. In addition, constraints related to velocity and acceleration limits are also applied to the trajectories to limit the maximum values while executing the tasks. The controller also considers the physical limits of the manipulator's motors to ensure their good and lasting functioning.

4.7 Scheduling

Regarding the scheduling problem, and based on the scheme presented in Figure 1.3, the graph for this TSP can be seen in Figure 4.12, with 0 being the starting point, 9 the final point, which can only be reached after all the objects are in their respective bases, 1 to 4 being the objects, and 5 to 8 being the bases of the objects.

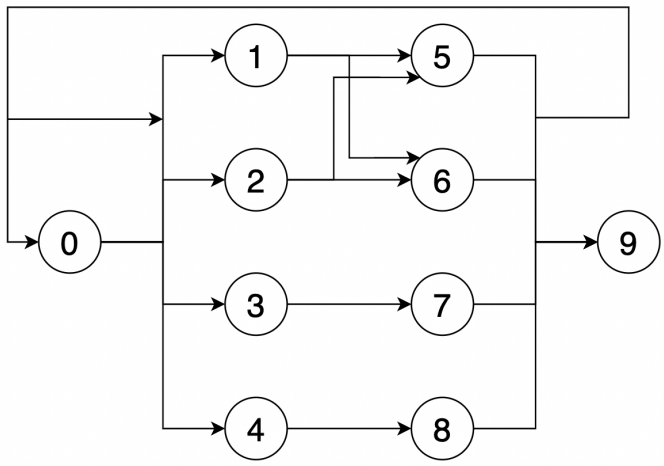

Figure 4.12: Graph for the specific TSP

In order to solve the scheduling problem, two $10*10$ matrices were created with i being the arm's current position and j the next position. The first, the cost matrix, represents the costs of the movement going from i to j. The second, the occupation matrix, represents the possibilities of moving from i to j: 0 if the movement can be performed, 1 if the move was already performed, and -1 if the move cannot be made. An example of a cost matrix can be seen in the $10*10$ matrix shown in equation 4.1.

$$\begin{bmatrix} nan & 0.4298 & 0.1292 & 0.6665 & 0.3071 & nan & nan & nan & nan & nan \\ nan & nan & nan & nan & nan & 0.6248 & 0.4110 & nan & nan & nan \\ nan & nan & nan & nan & nan & 0.4417 & 0.5221 & nan & nan & nan \\ nan & nan & nan & nan & nan & nan & nan & 0.4070 & nan & nan \\ nan & nan & nan & nan & nan & nan & nan & nan & 0.4211 & nan \\ 0.5186 & 0.3987 & 0.3710 & 0.4603 & 0.2716 & nan & nan & nan & nan & 0.5186 \\ 0.3539 & 0.3750 & 0.3521 & 0.3305 & 0.0995 & nan & nan & nan & nan & 0.3539 \\ 0.3609 & 0.4181 & 0.3467 & 0.3515 & 0.3517 & nan & nan & nan & nan & 0.3609 \\ 0.0031 & 0.2099 & 0.4139 & 0.2742 & 0.2520 & nan & nan & nan & nan & 0.0031 \\ nan & nan & nan & nan & nan & nan & nan & nan & nan & nan \end{bmatrix} \tag{4.1}$$

As we can see in the matrix, every movement that can't be done due to the restrictions of the problem has a value of *nan*, or none. On the other hand, every movement that can be performed has its cost on the matrix. This cost is calculated with the metrics described later in this paper. Stating an example: starting the matrix at $(0,0)$, the movement $(0,1)$, from the starting point to the first cube, has a cost of 0.4298. It is also important to mention that this matrix will not be modified during the process of the scheduling algorithms and will always maintain its values that derive from the path planning algorithms.

In addition to this, a $10 * 10$ occupation matrix was also developed. Unlike the last, this one will be updated throughout the process of choosing the best path, limiting the next steps that the manipulator can take. An example of an occupation matrix before starting the scheduling algorithm can be seen in 4.2. It is possible to see the similarity between the cost function and the occupation matrix at the beginning of the process since all the *nan*'s in the first matrix represent a -1 in the second, while the costs in the first represent a 0 in the second.

$$\begin{bmatrix} -1 & 0 & 0 & 0 & 0 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & 0 & 0 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & 0 & 0 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & 0 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & -1 & -1 & -1 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & -1 & -1 & -1 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & -1 & -1 & -1 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \end{bmatrix} \tag{4.2}$$

This matrix will be filled in throughout the process of scheduling. An example of the occupation matrix after the process can be seen in 4.3, with the final path being: $0-2-5-4-8-0-1-6-3-7-9$. By looking at the occupation matrix, we can confirm all the movements of the final path since these moves are set to 1 in the final matrix. It is possible to see that even though in the original occupation matrix, all the movements to the final point were not doable, in the final matrix, one of these movements was set to 1. This happens because when all the objects are already at their respective bases, the program sets the movements from the bases to the final point to 0, making one of these movements possible.

$$\begin{bmatrix} -1 & 1 & 1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 & 1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & 1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & 1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 1 & -1 \\ -1 & -1 & -1 & -1 & 1 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & 1 & -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 1 \\ 1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \end{bmatrix} \tag{4.3}$$

4.7.1 Dijkstra's greedy algorithm

The first implemented search algorithm to find the best path was Dijkstra's greedy algorithm, an algorithm that is used to find the shortest paths between nodes in a graph. When a source node is

given in, the algorithm finds the shortest path between that node and every other node in the graph that constitutes a valid movement, by verifying which of these movements is the shortest. Consequently, it operates in $O(n)$ time for selecting a single picking candidate, where n is the number of items accessible. In summary, the algorithm will only take into account the distance to the initial point $g(n)$, seen in equation 4.4 and in the Figure 4.13.

$$f(n) = g(n) \tag{4.4}$$

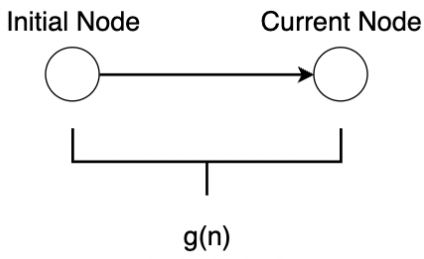

Figure 4.13: Dijkstra's greedy Representation

With the objective of finding the best path, the algorithm chooses the movement with the lowest cost of all the possible movements: the movements with a value of 0 in the occupation matrix and a valid cost. After choosing the next step, the occupation matrix will be updated depending on the move taken. This process will be repeated until we reach the final destination. This is shown in algorithm 1, with N being the number of positions.

Algorithm 1 Dijkstra's greedy algorithm

```
while source ≠ finalDestination do
    for destination = 1,2,...,N do
        costMove ← cost(source, destination);
        if costMove is valid and the move can be done then
            if cost < lowestCost then
                lowestCost ← cost;
                bestMove ← destination;
            end if
        end if
    end for
    source ← bestMove;
    finalPath ← finalPath ∪ {bestMove};
    Update the occupation matrix with the new restrictions;
end while
```

4.7.2 GRASP (Greedy randomized adaptive search procedure)

GRASP is a multi-start meta-heuristic method, seen in algorithm 2, composed of two main phases: construction and local improvement. The first phase is intended to generate a high quality and diverse solution, while the second is responsible for finding a local optimum with respect to a defined neighbourhood. The construction phase is similar to Dijkstra's greedy approach, except for the randomization during the first phase, increasing the diversity of the searched path. In addition to this, contrary to Dijkstra's method, each GRASP construction iteration can have different solutions, guiding the search to different possibilities and allowing the algorithm to explore a more significant portion of the search space [60].

Algorithm 2 GRASP algorithm

1: **while** criteria value is not reached **do**
2: $s \leftarrow constructionPhase()$;
3: $s' \leftarrow localSearchPhase(s)$;
4: **if** $costPath < lowestCost$ **then**
5: $lowestCost \leftarrow costPath$;
6: $bestPath \leftarrow path$;
7: **end if**
8: **end while**

Since the algorithm has a random component, we perform the construction and local search phases multiple times, choosing the path with the lowest cost of all iterations, seen in line 4 of algorithm 2. The criteria value represented in line 1 is the number of iterations that the algorithm performs, a value that will vary depending on what we want to study, as explained in chapter 5. This method has multiple advantages, such as a slight parameter dependency when it comes to reaching the result and simple implementation. It combines a random component with a greedy one, diversifying and intensifying the results.

4.7.2.1 Construction Phase

In the construction phase of the GRASP algorithm, seen in algorithm 3, a Restricted Candidate List (RCL) is selected by multiplying the Candidate List (CL), the possible next steps that the algorithm can take, by an α that can be defined in multiple ways (equation 4.5). The value of α can be constant, but it can also change at each iteration or through a random or adaptive scheme. In this case, α is selected through a random scheme to favour diversity since the problem that we are solving has a small number of possible steps. The possible values for α can be seen in equation 4.6. It is important to mention that the minimum size for the RCL is one element.

$$RCL = \alpha * CL \tag{4.5}$$

with:

$$\alpha \in [0.1, 0.2, 0.3, 0.4, 0.6, 0.7, 0.8, 0.9, 1] \tag{4.6}$$

This α restricts the number of possible steps that the algorithm can take. Even though the selection of α is random, the construction phase is a greedy function since the elements that are inserted on the RCL are the ones that have the lowest cost when evaluating greedily. After selecting the RCL, an element of the restricted list will be selected randomly to allow a more considerable search of the workspace. The randomly chosen element will be added to the partial solution, as seen in line 5 of the algorithm. If possible, the solution found with this phase will be used to reach a better one.

Algorithm 3 GRASP algorithm - Construction Phase

```
1: s ← represents partial solution;
2: while source ≠ finalDestination do
3:     RCL ← α * CL(s);
4:     x ← random element from RCL;
5:     s ← s ∪ x;
6: end while
```

4.7.2.2 Local Search Phase

The second phase of GRASP, or the local search phase, consists of refining the solution obtained in the first phase by applying a local search method. An intensification of the solution found by analysing neighbouring regions through local search leads to finding a local optimum.
In this phase, a 2-opt algorithm was used, which iteratively swaps the elements of the neighbourhood of a specific element in the cases where the swap can be performed based on the occupation grid. The cost of the temporary path is calculated, and the original path is changed if the new cost is lower than the original one (line 12). The only variation of the 2-opt method is the fact that we consider a neighbourhood of three elements. This is done because every object has to go to a base, meaning that a neighbourhood smaller than two elements would not make sense.

Algorithm 4 GRASP algorithm - Local Search Phase

```
1: s ← represents solution from the construction phase;
2: for all elements of the path do
3:     for all neighbours of element do
4:         if change is valid then
5:             auxiliarElement ← s[element];
6:             s[element] ← s[neighbour];
7:             s[neighbour] ← auxiliarElement;
8:         end if
9:     end for
10:    if costNewPath < costOriginalPath then
11:        costOriginalPath ← costNewPath;
12:        s ← newPath;
13:    end if
14: end for
```

The better the quality of the solution generated in the first phase, the greater the speed to find a local optimum through the local search phase. It is important to note that GRASP does not use previous information in the search process. However, GRASP is simple, fast and can be easily integrated with other search techniques.

4.7.3 A* algorithm

The third implemented algorithm was A*. Contrary to Dijkstra's method, it uses a heuristic function to help to determine the order of the nodes. The function can be seen in the equation 4.7 and Figure 4.14, with $g(n)$ being the cost from the start node to the current node and $h(n)$ the heuristic function that calculates the estimated cost of going from the current node to the end node.

$$f(n) = g(n) + h(n) \tag{4.7}$$

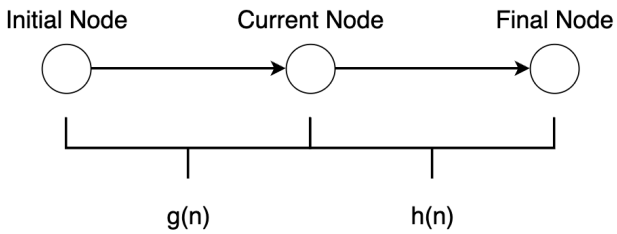

Figure 4.14: A* Method's function

Its time complexity is dependent on the heuristic function, and the number of nodes expanded grows exponentially with the depth of solution d. Therefore, it has a time complexity of $O(b^d)$, where b is the branching factor.
A-Star's search speed is positively correlated with its weight in the total estimated cost [62]. Considering $h(n)$ as the estimated value and $H(n)$ as the actual cost between the current node and the final goal:

- If $h(n) = H(n)$ the algorithm will never expand through unwanted paths, resulting in the best path with a high-speed computation;
- If $h(n) < H(n)$ the algorithm will find one of the shortest paths, but it will take longer;
- If $h(n) > H(n)$ it is difficult for the algorithm to reach an optimal path, even though the search will be fast.

For this algorithm to work, we need to guarantee it is complete. For this to happen, three conditions need to be met:

- The number of branches needs to be finite;
- The cost of the moves needs to be fixed;

- A* algorithm needs to be Optimal.

The first two conditions are met. However, for the A* algorithm to be optimal, two conditions need to be met:

- Admissability: the heuristic used to estimate the cost to the goal is optimistic, meaning that its value will never be higher than the real value;
- Consistency: the A* graph-search is consistent, meaning that the estimated distance from the neighbour to the goal, plus the cost of reaching that neighbour, is always less than or equal to the distance between the node and the goal.

If the first condition is guaranteed, the A* tree search will always find the path with the smallest cost. In some cases, a very good heuristic may be hard to find, mainly on a graph problem with many restrictions. For this problem, the heuristic is calculated using the minimum number of steps left to reach the final point and the cost of the smallest movement between any nodes. The formula is shown in equation 4.8.

$$h(n) = stepsLeft_{min} * cost_{lowest} \tag{4.8}$$

By using this heuristic, we guarantee its Admissibility; thus, the algorithm will always find one of the best paths.

The A* method, seen in algorithm 5, uses two main lists: the close list and the open list, seen in lines 1 and 2. The first has the already explored nodes, while the second contains the nodes that are left to be explored. The algorithm will start with picking the node with the lowest cost (f) and explore this node. If the node is not the goal, the algorithm will start to develop a list of the possible successors of the current node (line 11), considering the occupation matrix.
The next step is working through the list of successors, calculating their costs, $g(n)$ and $h(n)$, and checking if:

- Is there a node in the open list with a lower cost than the successor - if this happens, the successor will be skipped since the algorithm recognises that it will still evaluate a better option than the current successor;
- Is there a node in the close list with a lower cost than the successor - if this happens, the successor will be skipped since the algorithms recognise it has already evaluated a better option than the current successor;
- If the last condition does not apply, the algorithm will insert the successor into the open list to allow its evaluation later.

After this process, the current node will be inserted into the closed list. This process will be repeated until the open list is empty or until the node reaches the goal.

Algorithm 5 A* algorithm

```
OpenList ← {nodeStart};
CloseList ← {};
while OpenList is not empty do
    currentNode ← node with lowest f: f(currentNode) = g(currentNode) + h(currentNode);
    take currentNode out of the OpenList;
    if Node = goalNode then
        End the search;
    end if
    for node = 1,2,...,N do
        if node is a successor of currentNode then
            SuccessorList ← SuccessorList ∪ {node};
        end if
    end for
    for successorNode = 1,2,...,NUMBER of successors do
        if sucessorNode ≠ goalNode then
            successorNode.g = currentNode.g + cost(currentNode, successorNode);
            successorNode.h = numMinimumMovesLeft * minCost;
            if there is a node in the openList with a lower f than the successor then
                Skip this successor;
            end if
            if there is a node in the Closed list which has a lower f than successor then
                Skip this successor;
            end if
            if there are no nodes in the Closed list with lower f than the successor then
                OpenList ← OpenList ∪ {successor};
            end if
        end if
    end for
    CloseList ← CloseList ∪ {currentNode};
end while
```

4.7.4 Metrics

With the objective of creating a sorting method for the movements of the manipulator, metrics were defined by considering multiple variables, restrictions and requirements [63]. The best plan may not be the fastest or shortest; this way, the metric combines multiple elements to estimate the cost of a trajectory. The metric is composed of five components: planning time, movement of joints, distance travelled by the end-effector, security and torque, which together will result in a total cost that can be used to measure how good the trajectories are. These can be described as:

Definition 1. *Planning Time* - the time (t_p) the planner takes to compute the trajectory. Depends on the used planner and on the complexity of the trajectory: length and possible collisions. Thus, if the time is too high, it might not be adequate for real-time operations.

Definition 2. *Movement of Joints* - the total movement of the manipulator's joints (J_i). Calculated with the absolute value of the difference between the joint's angle at two consecutive trajectory points. The movement of the joints is the sum of all these differences for all the joints.

Definition 3. *Distance* - the total distance (D_j) that the end-effector has to travel to perform the trajectory. It is calculated with the sum of the distances between every two consecutive trajectory points.

Definition 4. *Security* - the minimum distance from the manipulator to all obstacles (d_o). This metric was developed to protect the manipulator and its surroundings, allowing the system to choose a more cautious trajectory.

Definition 5. *Maximum Torque* - the maximum torque (τ_{max}) in the joints of the manipulator caused by the grabbed object.

The torque can be defined as the distance multiplied by force, seen in equation 4.9.

$$\tau = d * F \tag{4.9}$$

For this case, since we want to calculate the maximum torque for all the joints, we need to calculate the torque for all joints in every point of the trajectories. This calculation can be performed with the equation 4.10.

$$\tau = J^T * w, \tag{4.10}$$

where τ is the vector of joint torques, J^T the transpose of the Jacobian matrix, and w the Cartesian wrench, which is seen in equation 4.11.

$$w = \begin{pmatrix} f \\ m \end{pmatrix}, \tag{4.11}$$

where f is the vector of forces and m the vector of the moment. For the moments where the manipulator is grabbing an object, the wrench can be defined as seen in equation 4.12, where the force of buoyancy and force of gravity can be described as in equation 4.13 and 4.14.

$$w = \begin{pmatrix} f_x \\ f_y \\ f_z \\ m_x \\ m_y \\ m_z \end{pmatrix} = \begin{pmatrix} f_x \\ f_y \\ f_z \\ m_x \\ m_y \\ m_z \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ F_{buoyancy} - F_{gravity} \\ 0 \\ 0 \\ 0 \end{pmatrix} \tag{4.12}$$

$$F_{buoyancy} = \rho * g * V, \tag{4.13}$$

$$F_{gravity} = m * g, \tag{4.14}$$

where ρ is the density of water, g is the gravitational acceleration, and m is the mass of the grabbed object. By calculating the torque in every joint for all the trajectory points, it is possible to see the maximum torque (τ_{max}) of each trajectory in all joints. This study did not consider the trajectory time metric because the Bravo 7 joint velocity controllers are unknown, which prevents determining this parameter.

Since we want to sum and compare values with different units, all values used for the metrics were normalised using the Min-Max feature scaling technique, presented in equation 4.15. By using the minimum and maximum values of the values, it is possible to work on a range of [0,1] and, simultaneously, compare values with different units.

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}}, \tag{4.15}$$

where $X_{norm} \in \mathbb{R}^n$, $n = 5$, X the absolute observation value and X_{min} and X_{max} the minimum and maximum values, respectively.

Considering the five metrics, the final cost (C) of each movement is the sum of the costs of each metric in the same movement, seen in equation 4.16.

$$C_{final} = C_{planningTime} + C_{movementJoints} + C_{eeDistance} + C_{security} + C_{maxTorque}, \tag{4.16}$$

where the cost of each metric, $Cost_{metric}$, is the weight (K_{metric}) of that metric multiplied by its normalised cost (N_{metric}), seen in equation 4.17.

$$C_{metric} = K_{metric} * N_{metric} \tag{4.17}$$

In a first iteration, the weights have the same value, as seen in Table 4.3.

Weight	Planning Time	Movement Joints	EE-Distance	Security	Maximum Torque
Value	1/5	1/5	1/5	1/5	1/5

Table 4.3: Weights of the metrics

Chapter 5

Results

This chapter presents the results obtained for the methods implemented to solve the problem this thesis addresses. Firstly, the Bravo 7 manipulator is presented along with its characteristics and restrictions. Afterwards, the developed environment, both simulated and real, is explained along with its characteristics and specifications. The results provided by the scheduling algorithms are explained in detail. Lastly, the steps performed to test the developed system are described, along with a presentation of the final results.

5.1 Environment

To allow the execution, study, testing and evaluation of a pick & place task, an environment was designed and built for underwater scenarios, containing the main cage, four tables, four objects and eight bases. The cage is a 2*2*1m structure representing an approximation of the manipulator's workspace. The two tables where the objects are on, in Figure 5.1, represent the initial positions of the objects in the experiment. While the other inclined and straight tables, represented in light purple and light blue, respectively, are the final destination for the same objects.

In the proposed problem, only static collision objects were used, and their representation in the simulated environment is presented in Figure 5.2. The simulator was developed to match the actual structure. In addition, to ensure that the manipulator does not leave the cage's limits, a ceiling collision object is also included. Finally, there is also an object (table 5 shown in Figure 5.2) on which all the tables are placed to eliminate possible unwanted trajectories for the manipulator. It is essential to mention that the manipulator is mounted onto the top part of the cage to simulate its real position in an AUV, where the manipulator would be mounted onto the lower part of the AUV to allow the performance of the designated tasks. In addition, using the AUV movement would immensely increase the workspace of the system.

Three types of picking objects were used: two cubes, a cylinder and a hexagon, each with a specific base that corresponds to their geometry. These are orange, red and green, respectively. This decision was based on their ability to be identified underwater. The first two cubes have edges of 80mm, while the cylinder has a height of 60mm and 80mm in diameter. The last, the hexagon,

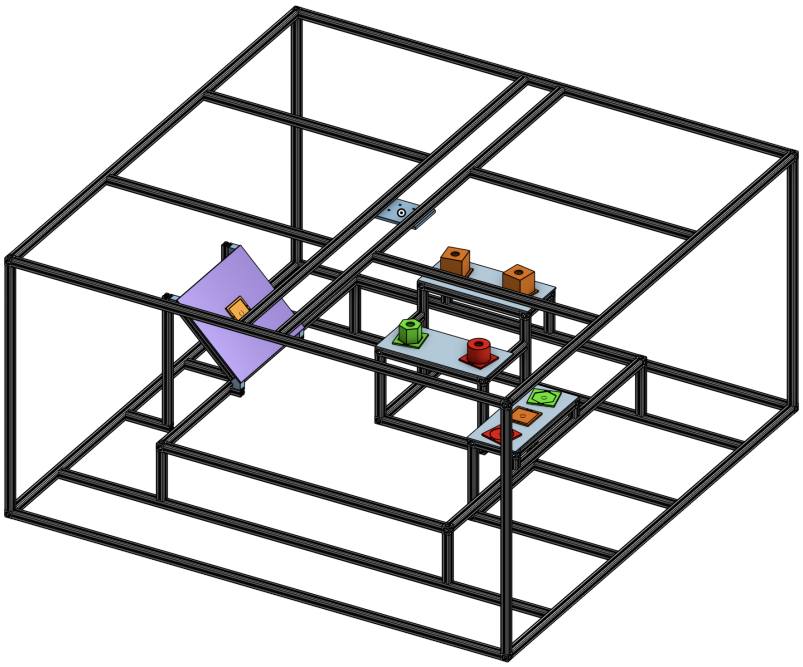

Figure 5.1: Model of Developed Environment

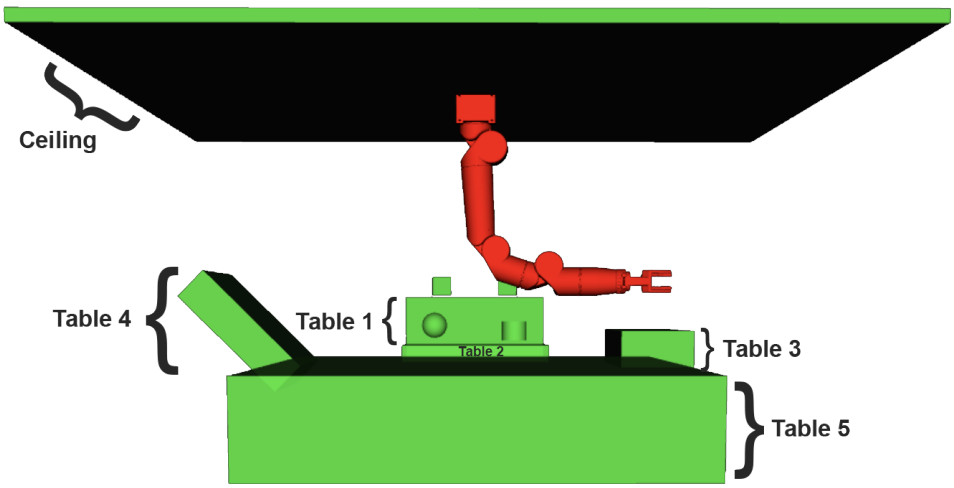

Figure 5.2: Collision objects in the simulator

has a height of 60mm and a 40mm edge. The models of the objects and the respective bases can be seen in Figure 5.3.

Since the objects were used underwater, each solid has a circular hole to allow the change of their weights during testing. In addition, circular neodymium magnets were used to ensure that

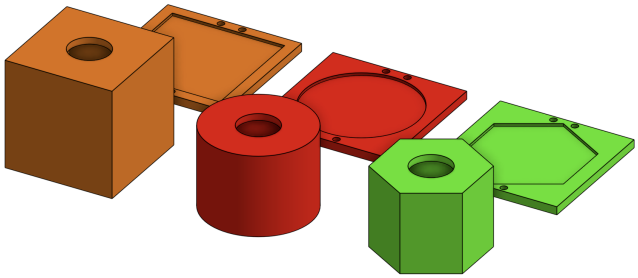

Figure 5.3: Model of Pick Objects

the solids would stay still and not float or fall from their bases. Furthermore, the objects and the bases contain circular holes for the magnets to be inserted. The holes for the magnets can be seen in the transparent models in Figure 5.4.

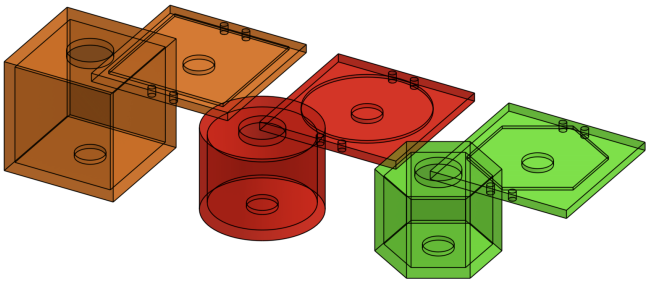

Figure 5.4: Pick Objects - Transparent Model

All the picking objects and respective bases were printed with PLA in a 3D printer to allow their use in the real tests. The objects can be seen in Figure 5.5.

Figure 5.5: 3D Printed Pick Objects

The cage was built recurring to extruded aluminium profiles and aluminium screws due to its good characteristics in underwater environments, such as its resistance to corrosion. In addition, the object's bases were screwed onto acrylic tables with appropriate sizes for the setup. The final structure can be seen in a dry and underwater environment in Figures 5.6 and 5.7, respectively.

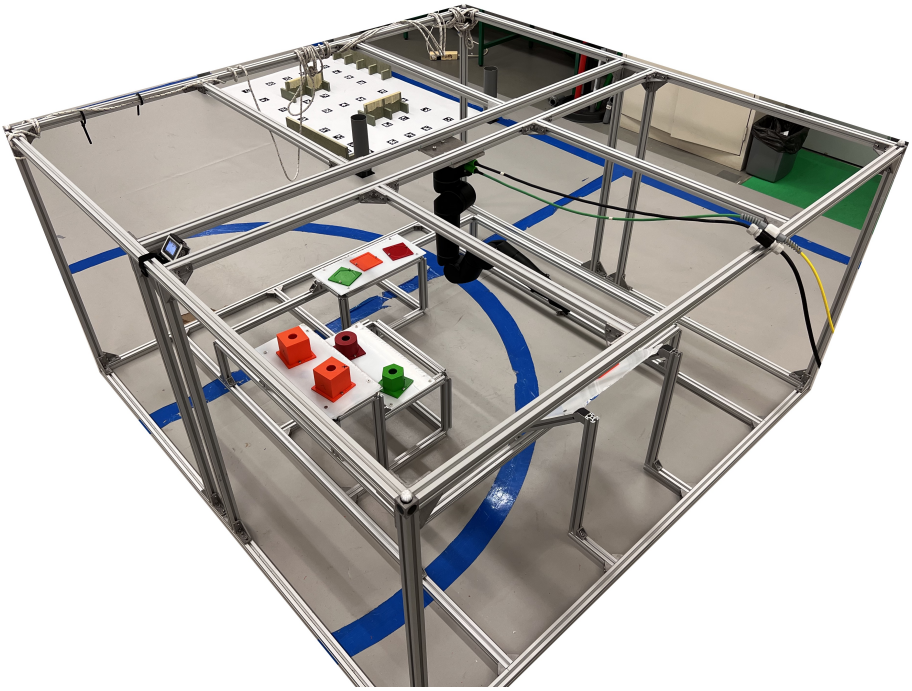

Figure 5.6: Final Setup in a dry environment

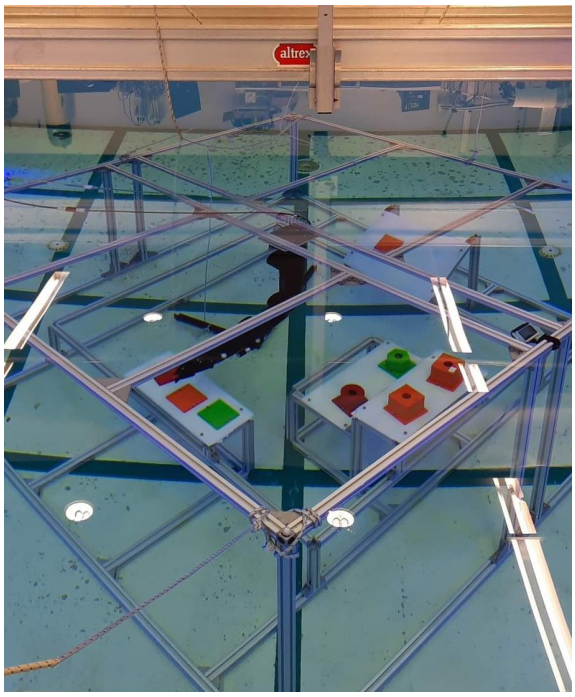

Figure 5.7: Final Setup in an underwater environment

5.2 Motion Capture System

Motion Capturing (MoCap) allows the recording of motions translating them into mathematically-usable signals. These signals correspond to several key points in space over time, allowing the operator to observe the evolution of the motions in a defined period. The three MoCap most used techniques are MoCap with markers, MoCap with magnetic sensors and Mocap with accelerometers. The first type works with the markers fixed to the object making the motion. A camera shoots the object and the software analyzes the position of the markers in each frame and calculates the movement of each part of the body. The tags can be made in different shapes and placed on different parts of the body/object, using different colours and characteristics such as fluorescent materials, light-reflecting materials and LEDs to facilitate their distinguishment. Mocap systems with magnetic sensors have sensors fixed in the object's body to measure the low-frequency magnetic field generated by a transmitter. The sensors will store the position and rotational information. At last, the Mocap systems with accelerometers use these sensors attached to the body to register linear acceleration and tilt angles.
There are multiple professional solutions such as VICON[1], Gypsy7[2] and Qualisys[3]. These systems can use various cameras and have many applications, such as automotive, OEM, robotics and UAV, marine and underwater, virtual reality, animation industries and much more. The main parameters of a MoCap system are their accuracy, the possibility of animation in real-time, freedom of movements, frames per second, interruptions, identification, external influence, price, training, portability and software complexity [64, 65, 66].

5.3 Scheduling Algorithms

This section will present the results regarding the developed scheduling algorithms for the TSP presented in Figure 4.12. The motion planning algorithms: RRT Connect, BKPIECE, and PRM will be used to test the scheduling algorithms in different cases with different motion planning trajectories.
To simplify the results, the resulting paths will be shown numerically instead of stating the names of the nodes. This is done according to the values in Table 5.1. The goal is the starting point geometrically, but the corresponding number will be different to facilitate the reading of the path results.

Number	0	1	2	3	4	5	6	7	8	9
Node	Starting Point	Cube 1	Cube 2	Cylinder	Hexagon	Base Cube 1	Base Cube 2	Base Cylinder	Base Hexagon	Goal

Table 5.1: Nodes name and corresponding number

[1] VICON https://www.vicon.com
[2] Gypsy 7 https://metamotion.com/gypsy/gypsy-motion-capture-system.html
[3] Qualisys https://www.qualisys.com

It is important to mention for all the results in this section that the Dijkstra's and A* algorithms always output the exact cost and path when using the same trajectories from the motion planning algorithms. However, the GRASP algorithm has a random component, which means that the results are not guaranteed to be the same even when using the same trajectories. The results of this algorithm are also influenced by the number of performed iterations, a characteristic that will be studied later in this section.

5.3.1 Final Cost

For the first case, using the RRT Connect algorithm to plan the trajectories, 1000 iterations for the GRASP, and the same weight for each metric, the results are presented in Table 5.2.

Algorithm	Final Path	Metric	K	N_{metric}	C_{final}
Dijkstra	0-4-8-0-3-7-0-1-5-0-2-6-9	Planning Time	1/5	3.647	3.821
		Movement Joints	1/5	1.504	
		EE-Distance	1/5	3.574	
		Security	1/5	4.603	
		Maximum Torque	1/5	5.779	
GRASP	0-4-8-3-7-1-6-2-5-9	Planning Time	1/5	1.991	3.001
		Movement Joints	1/5	1.750	
		EE-Distance	1/5	3.094	
		Security	1/5	2.875	
		Maximum Torque	1/5	5.298	
A*	0-4-8-3-7-2-6-1-5-9	Planning Time	1/5	3.001	2.974
		Movement Joints	1/5	1.200	
		EE-Distance	1/5	2.598	
		Security	1/5	2.791	
		Maximum Torque	1/5	5.280	

Table 5.2: Direct Comparison between the scheduling algorithms - using RRT Connect

The results shown in Tables 5.2, 5.3 and 5.4 demonstrate that Dijkstra's algorithm has the worst cost, with GRASP and A* methods having significantly lower costs than the first. In addition, when using the RRT Connect algorithm, Dijkstra's algorithm chooses to move to the starting point during the path due to the non-existence of a heuristic. While the second and third methods choose to never go to the starting point after the path starts.
In addition, the A* got a slightly lower cost than GRASP, independently of the motion plan algorithm. One of the reasons for this to happen is the fact that the A* chooses never to move to the starting point during the path, which does not happen when using GRASP with BKPIECE and PRM (Tables 5.3 and 5.4). No conclusions should be taken on the difference between the results of the GRASP and A* method due to the small difference in cost, the randomness of the first and the possibility of the A* method getting stuck in a local optimum.

When taking into account the results of the last three tables, the chart in Figure 5.8 was developed. It is, once again, possible to see the similarities between the results of the A* and GRASP

Algorithm	Final Path	Metric	K	N_{metric}	C_{final}
Dijkstra	0-4-8-0-3-7-0-1-5-0-2-6-9	Planning Time	1/5	2.007	3.498
		Movement Joints	1/5	1.498	
		EE-Distance	1/5	3.577	
		Security	1/5	4.601	
		Maximum Torque	1/5	5.806	
GRASP	0-4-8-1-5-0-3-7-2-6-9	Planning Time	1/5	2.010	2.968
		Movement Joints	1/5	1.557	
		EE-Distance	1/5	3.283	
		Security	1/5	3.170	
		Maximum Torque	1/5	4.820	
A*	0-4-8-3-7-2-6-1-5-9	Planning Time	1/5	1.948	2.763
		Movement Joints	1/5	1.200	
		EE-Distance	1/5	2.597	
		Security	1/5	2.789	
		Maximum Torque	1/5	5.281	

Table 5.3: Direct Comparison between the scheduling algorithms - using BKPIECE

Algorithm	Final Path	Metric	K	N_{metric}	C_{final}
Dijkstra	0-4-8-0-3-7-0-1-5-0-2-6-9	Planning Time	1/5	3.036	3.748
		Movement Joints	1/5	1.512	
		EE-Distance	1/5	3.666	
		Security	1/5	4.600	
		Maximum Torque	1/5	5.925	
GRASP	0-4-8-1-6-3-7-0-2-5-9	Planning Time	1/5	1.686	3.111
		Movement Joints	1/5	1.588	
		EE-Distance	1/5	3.039	
		Security	1/5	3.503	
		Maximum Torque	1/5	5.739	
A*	0-4-8-3-7-2-6-1-5-9	Planning Time	1/5	2.674	2.908
		Movement Joints	1/5	1.201	
		EE-Distance	1/5	2.597	
		Security	1/5	2.788	
		Maximum Torque	1/5	5.280	

Table 5.4: Direct Comparison between the scheduling algorithms - using PRM

algorithm when the second is executed with 1000 iterations. In addition, it is possible to conclude that Dijkstra's algorithm has the worst results, independently of the motion planning algorithm used.

5.3.2 GRASP and number of iterations

Since the results of the GRASP algorithm rely on the number of iterations, it is interesting to study the final cost outputted by the method when using different numbers of iterations. The study was performed with one of the motion planning algorithms (RRT Connect), with the same weight for

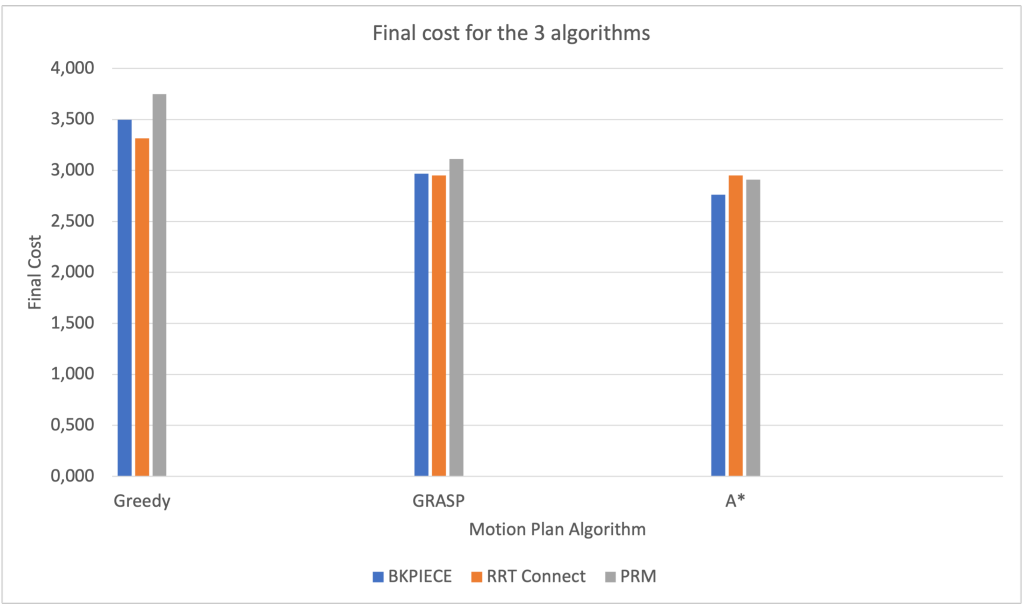

Figure 5.8: Final costs of the 3 algorithms

every metric and with a variable number of iterations: from 0 to 3000 iterations, with increments of 50. The results can be seen in the graph from Figure 5.9.

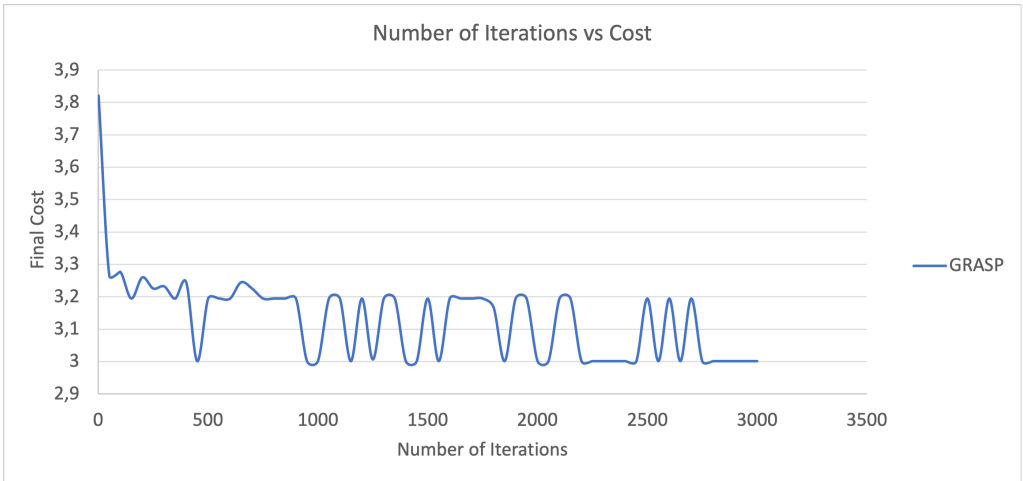

Figure 5.9: GRASP: Iterations vs Cost

In addition, looking at the graph in Figure 5.10, it is possible to see that Dijkstra's algorithm has a similar cost to the GRASP with 1 iteration. Furthermore, the GRASP algorithm stabilises around the A* cost after around 1000 iterations, which means that in most cases, it is not necessary to perform more than 1000 iterations of the GRASP method.

As seen in Figure 5.9, the GRASP method has a higher cost when working with a low number of iterations (< 500), between 3.2 and 3.8. This is expected since fewer iterations lead to a smaller exploration for the best solution possible. In addition, when the number of iterations passes the

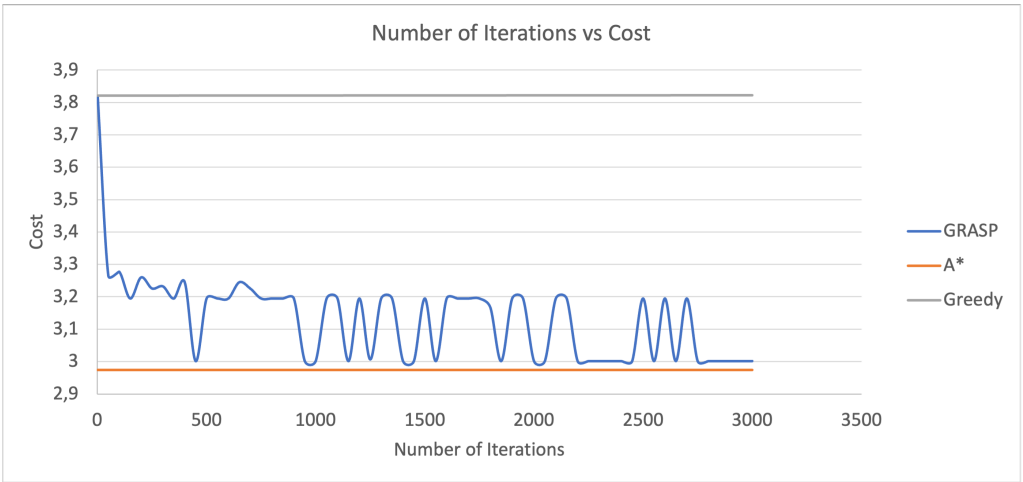

Figure 5.10: Three Algorithms: Iterations vs Cost

around 600 mark, the cost starts to stabilise between the 3.0 and 3.2.
The execution time of the algorithms is also a significant characteristic since it shows how fast and how much computational resources the method needs to reach the result. The execution time will be studied later in this chapter, but for now, we will be looking at the way the execution time of the GRASP algorithm varies with the number of iterations. As seen in the graph from Figure 5.11, the execution time, in ms, increases with the number of iterations in almost a linear way. Nevertheless, it is not a perfectly linear approximation because the construction phase has a random component.

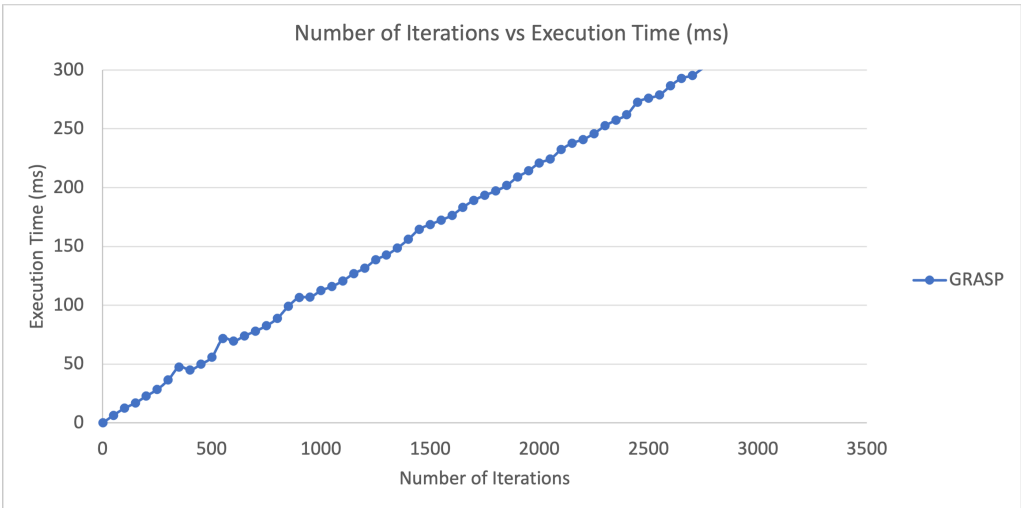

Figure 5.11: GRASP: Iterations vs Execution Time (ms)

5.3.3 Execution Time

The execution time of the algorithms shows how fast and how much computational resources the method needs to reach the end result. We already saw that the execution time of the GRASP algorithm increases with the number of iterations. Still, we wanted to assess how much this value represents compared to Dijkstra's and A* methods. For this, Table 5.5 shows the results of the three algorithms with the same conditions shown before: 1000 iterations for GRASP and the same weight for every metric, but now with the execution times represented in ms. As seen in the table, Dijkstra's method has the lowest execution time of 0.0121ms, while the A* comes after with 0.9435ms. The first is justified by the simplicity of the greedy algorithm and the fact that it only goes through the path once, while the A* algorithm calculates the cost 155 times, which is the number of nodes that ended up in the closed list.

Algorithm	Final Path	Metric	K	N_{metric}	C_{final}	Exec Time (ms)
Dijkstra	0-4-8-0-3-7-0-1-5-0-2-6-9	Planning Time	1/5	3.647	3.821	0.0121
		Movement Joints	1/5	1.504		
		EE-Distance	1/5	3.574		
		Security	1/5	4.603		
		Maximum Torque	1/5	5.779		
GRASP	0-4-8-3-7-1-6-2-5-9	Planning Time	1/5	1.991	3.001	110.228
		Movement Joints	1/5	1.750		
		EE-Distance	1/5	3.094		
		Security	1/5	2.875		
		Maximum Torque	1/5	5.298		
A*	0-4-8-3-7-2-6-1-5-9	Planning Time	1/5	3.001	2.974	0.9435
		Movement Joints	1/5	1.200		
		EE-Distance	1/5	2.598		
		Security	1/5	2.791		
		Maximum Torque	1/5	5.280		

Table 5.5: Scheduling algorithms with execution time - using RRT Connect

Since we are comparing Dijkstra's and A* algorithm with a GRASP method with a variable number of iterations, it is interesting to study and compare the execution time of the first two with the curve of the execution time of the GRASP method. This comparison can be done with the graph in Figure 5.12. As expected, the execution time of the GRASP algorithm is very high compared to the other two.

5.3.4 Sensibility test - Weights

In this sub-section, a sensibility test is performed to verify the sensibility of the algorithms to changes in the weights of the multiple criteria when calculating the final cost. In order to assess the sensibility, the tests were performed by giving the total cost to one metric, each at a time. This was tested with the RRT Connect motion plan algorithm and 1000 iterations for GRASP. As seen in Tables A.1, A.2, A.3, A.4 and A.5, the final paths alter when the weights of the metrics are changed, as expected. For this case, no resulting paths are the same when using different weights.

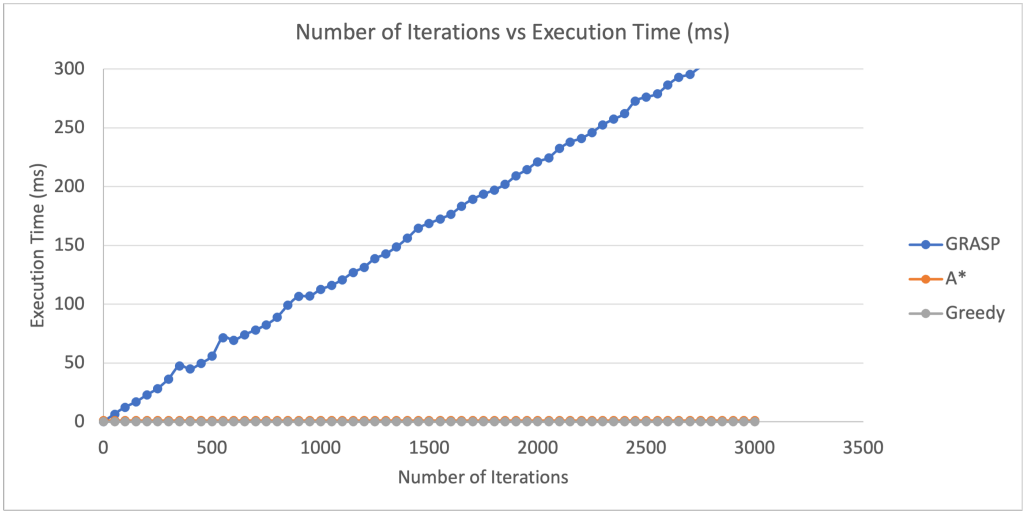

Figure 5.12: Three Algorithms: Iterations vs Execution Time (ms)

In addition, the final costs reflect only the normalised value of the metric, whose weight is 1. It is also possible to see that the normalised values also change since the final path is different.
With this, we can conclude that the algorithms are sensitive to the weight of the metrics.
It is also possible to assess that, just like before, Dijkstra's method gets the worst results, while GRASP and A* have similar results. In addition, with 1000 iterations, the GRASP method reached solutions with lower or equal costs than A* for the five sets of weights. This difference may be justified by the possibility of the A* method getting stuck in a local optimum. In contrast, the GRASP method may have found a better solution in one of the 1000 iterations.

5.3.5 Specific Scenarios

In this sub-section, specific scenarios with particular requirements and constraints that require certain values for the weights of the metrics are studied. With the objective of verifying if the results are consistent with the defined scenario, the results are compared with the ones presented in Table 5.5, since the scenario that resulted in Table 5.5 was defined with the same conditions as the ones in this sub-section, but with the initial weights for the metrics. The study will focus on the GRASP and A* algorithms since they achieved the best results, making them more suitable for comparison. On the contrary, Dijkstra's algorithm may not reach the desired path, meaning that the final normalised values should not be compared.
For the first scenario, we considered a situation where the main focus is the movements' speed and efficiency. For this, the total movement of the joints and end-effector distance will be the main metrics that will affect the cost, both weighting 35/100. The results of this configuration using the motion plan algorithm RTT Connect and the GRASP method with 1000 iterations can be seen in Table 5.6. By comparing the values of $N_{movementJoints}$ and $N_{EE-distance}$ with the ones in Table 5.5, we obtained a decrease of 31.43% and 12.58% in the GRASP and A* algorithm, respectively, for the total movement of the joints, seen in equations 5.1 and 5.2. In addition, we obtained a

decrease of 16.03% and an increase of 2.35% in the GRASP and A* algorithm, respectively, for the end-effector distance metric, seen in equations 5.3 and 5.4.

$$diff_{GRASP}^{movementJoints}(\%) = \frac{N_{scenario} - N_{reference}}{N_{reference}} = \frac{1.200 - 1.750}{1.750} * 100 = -31.43\% \tag{5.1}$$

$$diff_{A*}^{movementJoints}(\%) = \frac{N_{scenario} - N_{reference}}{N_{reference}} = \frac{1.049 - 1.200}{1.200} * 100 = -12.58\% \tag{5.2}$$

$$diff_{GRASP}^{EE-distance}(\%) = \frac{N_{scenario} - N_{reference}}{N_{reference}} = \frac{2.598 - 3.094}{3.094} * 100 = -16.03\% \tag{5.3}$$

$$diff_{A*}^{EE-distance}(\%) = \frac{N_{scenario} - N_{reference}}{N_{reference}} = \frac{2.659 - 2.598}{2.598} * 100 = 2.35\% \tag{5.4}$$

It is essential to mention that the algorithm minimises the sum of the costs and not the individual values of each metric, which can lead to a high decrease in one important metric, but a slight increase in another primary metric, as happened in this first scenario.

Algorithm	Final Path	Metric	K	N_{metric}	C_{final}	Exec Time (ms)
Dijkstra	0-4-8-0-3-7-0-2-5-0-1-6-9	Planning Time	10/100	2.594	3.396	0.0222
		Movement Joints	35/100	2.041		
		EE-Distance	35/100	3.966		
		Security	10/100	4.602		
		Maximum Torque	10/100	5.741		
GRASP	0-4-8-3-7-2-6-1-5-9	Planning Time	10/100	3.001	2.437	119.336
		Movement Joints	35/100	1.200		
		EE-Distance	35/100	2.598		
		Security	10/100	2.791		
		Maximum Torque	10/100	5.280		
A*	0-4-8-3-7-0-2-6-1-5-9	Planning Time	10/100	3.149	2.541	1.0443
		Movement Joints	35/100	1.049		
		EE-Distance	35/100	2.659		
		Security	10/100	3.502		
		Maximum Torque	10/100	5.777		

Table 5.6: Scheduling Algorithms - First Scenario

Secondly, we considered a scenario where the security metric is crucial because we want to make sure the manipulator and grabbed object are not too close to collision objects. For this, the security metric will weigh 0.6 while the others will all have a weight of 0.1. As seen by the results in Table 5.7, security values are low in every algorithm. By comparing the values of $N_{security}$ with the ones in Table 5.5, we get a decrease of 11.14% in the case of the A* algorithm, as seen in equation 5.6, and a decrease of 19.27% when using the GRASP method, seen in equation 5.5.

$$diff_{GRASP}^{security}(\%) = \frac{N_{scenario} - N_{reference}}{N_{reference}} = \frac{2.321 - 2.875}{2.791} * 100 = -19.27\% \tag{5.5}$$

$$diff_{A*}^{security}(\%) = \frac{N_{scenario} - N_{reference}}{N_{reference}} = \frac{2.480 - 2.791}{2.791} * 100 = -11.14\% \tag{5.6}$$

Algorithm	Final Path	Metric	K	N_{metric}	C_{final}	Exec Time (ms)
		Planning Time	10/100	4.122		
		Movement Joints	10/100	1.652		
Dijkstra	0-3-7-1-5-4-8-2-6-9	EE-Distance	10/100	3.678	2.885	0.0164
		Security	60/100	2.344		
		Maximum Torque	10/100	5.335		
		Planning Time	10/100	2.981		
		Movement Joints	10/100	2.127		
GRASP	0-4-8-2-5-3-7-1-6-9	EE-Distance	10/100	3.823	2.811	92.5959
		Security	60/100	2.321		
		Maximum Torque	10/100	5.251		
		Planning Time	10/100	4.004		
		Movement Joints	10/100	1.331		
A*	0-3-7-4-8-2-6-1-5-9	EE-Distance	10/100	2.994	2.855	0.6935
		Security	60/100	2.480		
		Maximum Torque	10/100	5.334		

Table 5.7: Scheduling Algorithms - Second Scenario

For the third scenario, we considered a situation where the main focus is the security of the trajectory along with minimising the maximum possible torque acting on the manipulator. Therefore, the first three metrics are set to a low weight, while the last two have high importance. As shown in Table 5.8, the values of security and maximum torque metrics are the lowest in the GRASP and A* algorithms, as expected. If we compare the values of the A* and GRASP methods with the ones in Table 5.5 (reference table), we see that we increased the $N_{security}$ by 4.91% but decreased the $N_{maxTorque}$ by 9.79% for the A* method, as seen in equations 5.7 and 5.8. Later, the $N_{security}$ increased by 1.84% while the $N_{maxTorque}$ decreased 10.10% in the case of the GRASP method, as seen in equations 5.9 and 5.10.

$$diff_{A*}^{security}(\%) = \frac{N_{scenario} - N_{reference}}{N_{reference}} = \frac{2.928 - 2.791}{2.791} * 100 = 4.91\% \tag{5.7}$$

$$diff_{A*}^{maxTorque}(\%) == \frac{N_{scenario} - N_{reference}}{N_{reference}} = \frac{4.763 - 5.280}{5.280} * 100 = -9.79\% \tag{5.8}$$

$$diff_{GRASP}^{security}(\%) = \frac{N_{scenario} - N_{reference}}{N_{reference}} = \frac{2.928 - 2.875}{2.875} * 100 = 1.84\% \tag{5.9}$$

$$diff_{GRASP}^{maxTorque}(\%) = \frac{N_{scenario} - N_{reference}}{N_{reference}} = \frac{4.763 - 5.298}{5.298} * 100 = -10.10\% \tag{5.10}$$

Algorithm	Final Path	Metric	K	N_{metric}	C_{final}	Exec Time (ms)
Dijkstra	0-4-8-0-3-7-0-1-6-0-2-5-9	Planning Time	10/100	2.594	4.596	0.0167
		Movement Joints	10/100	2.041		
		EE-Distance	10/100	3.956		
		Security	35/100	4.602		
		Maximum Torque	35/100	5.741		
GRASP	0-4-8-2-5-0-3-7-1-6-9	Planning Time	10/100	3.145	3.479	115.813
		Movement Joints	10/100	2.169		
		EE-Distance	10/100	3.853		
		Security	35/100	2.928		
		Maximum Torque	35/100	4.763		
A*	0-4-8-2-5-0-3-7-1-6-9	Planning Time	10/100	3.145	3.479	1.1353
		Movement Joints	10/100	2.169		
		EE-Distance	10/100	3.853		
		Security	35/100	2.928		
		Maximum Torque	35/100	4.763		

Table 5.8: Scheduling Algorithms - Third Scenario

As a final scenario, we considered a situation that focuses primarily on the planning time. For this, the first metric has a weight of 0.6 while the other weights are set to 0.1, giving the most importance to the planning time, as seen in Table 5.9. If we compare the values of the A* and GRASP methods with the ones in Table 5.5 (reference table), we see a decrease in the $N_{planningTime}$ of 33.66% when using the A* method, seen in equation 5.11, while $N_{planningTime}$ maintained its value when using the GRASP method, as seen in equation 5.12.

$$diff_{A*}^{planningTime}(\%) = \frac{N_{scenario} - N_{reference}}{N_{reference}} = \frac{1.991 - 3.001}{3.001} * 100 = -33.66\% \tag{5.11}$$

$$diff_{GRASP}^{planningTime}(\%) = \frac{N_{scenario} - N_{reference}}{N_{reference}} = \frac{1.991 - 1.991}{1.991} * 100 = 0\% \tag{5.12}$$

It is essential to mention that in some of the results for specific scenarios, the final cost when using GRASP was lower than when using A*. This may be due to the fact that the A* method may get stuck in a local optimum, while the GRASP may have found the final solution by improving its initial solution. Although this happens in the first and second scenarios, the difference between the method's final costs is low.

5.4 Testing the system

After testing the developed work in a simulated environment, the next step was testing the movements, control and planning in the real Bravo 7. To do this, the work was divided into simpler and more general steps, starting with the *dry tests* described in appendix B.

After making sure that the implementation worked in the *dry scenario*, by making small adjustments in the positioning of the objects and respective bases, the tests were then implemented in

Algorithm	Final Path	Metric	K	N_{metric}	C_{final}	Exec Time (ms)
Dijkstra	0 4 8 3 7 0 2 5 0 1 6 9	Planning Time	60/100	2.356	2.946	0.0132
		Movement Joints	10/100	1.881		
		EE-Distance	10/100	3.738		
		Security	10/100	3.972		
		Maximum Torque	10/100	5.740		
GRASP	0 4 8 3 7 1 6 2 5 9	Planning Time	60/100	1.991	2.496	139.078
		Movement Joints	10/100	1.750		
		EE-Distance	10/100	3.093		
		Security	10/100	2.875		
		Maximum Torque	10/100	5.298		
A*	0 4 8 3 7 1 6 2 5 9	Planning Time	60/100	1.991	2.496	1.6740
		Movement Joints	10/100	1.750		
		EE-Distance	10/100	3.093		
		Security	10/100	2.875		
		Maximum Torque	10/100	5.298		

Table 5.9: Scheduling Algorithms - Fourth Scenario

an underwater scenario using the Bravo 7 manipulator owned by the Center for Robotics and Autonomous Systems from INESC TEC (Institute of Systems and Computer Engineering, Technology and Science). The experiments were performed in a pool at FEUP (Faculdade de Engenharia da Universidade do Porto), along with the use of an underwater mounted motion capture system Qualisys. In order to be detected by Qualisys, Bravo 7's frame and end-effector were marked. In addition, the system was calibrated before the tests. The underwater scenario can be see in Figure 5.13.

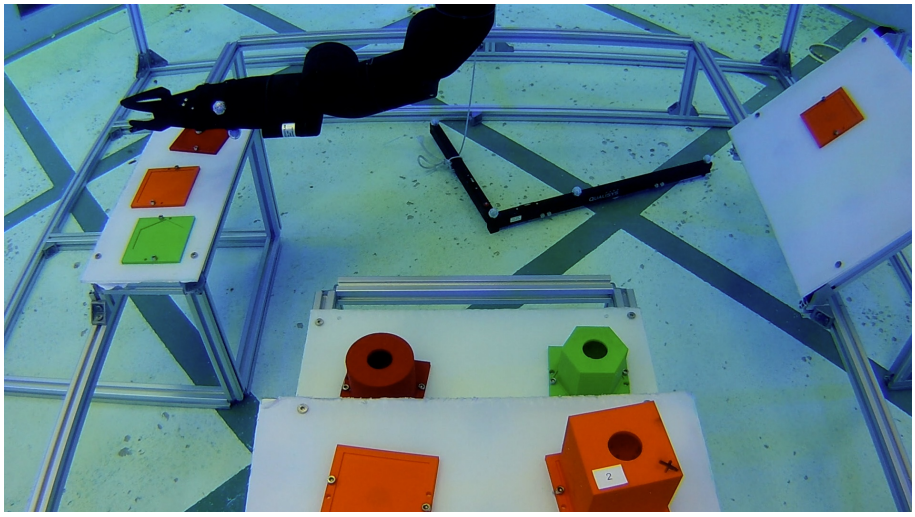

Figure 5.13: Underwater Scenario - During Tests

During this test, all sequences were tested underwater and were concluded successfully. In addition, the system recorded the data generated by Qualisys, allowing the comparison between the planned and executed trajectories. This comparison can be seen in Figure 5.14. The blue dots

represent the points generated by BKPIECE, whereas the green dots represent the end-effector position during the Moveit simulation. In order to obtain the simulation points, we considered the kinematics of the Bravo 7 manipulator. Moreover, the red dots show the position of the end effector detected by the Qualisys system underwater, which has a precision around 2 millimetres. Due to mechanical restrictions, the markers were slightly offset from the real position of the end-effector. A transformation was applied to compensate for this.

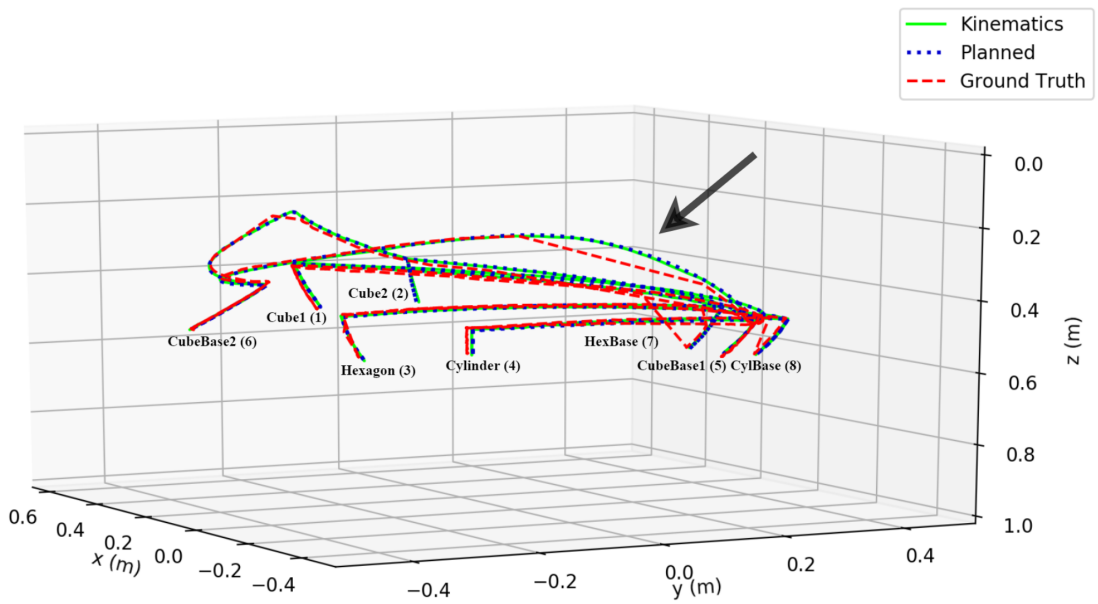

Figure 5.14: Bravo 7 end-effector trajectory comparison

Figure 5.14 shows the positioning of the different objects and bases along the Bravo 7's path. Using the Qualisys data as ground truth, we confirmed that the simulated trajectories were executed as expected in the real world. As observable, the errors between the planned and the executed trajectories are minimal. Despite this, because the environment created inside the Bravo 7 frame was highly congested, the Qualisys cameras detected a high number of reflections. Thus, although the majority of the occlusions were masked before the real tests, some are almost impossible to remove. As a result, the Qualisys system stopped detecting the markers on the Bravo 7 manipulator in some regions, which is evident in the zone marked by the black arrow in Figure 5.14, where no points were detected.

Chapter 6

Conclusions and Future Work

The purpose of this chapter is to summarise the conclusions reached throughout the project that led to the creation of this thesis. Besides illustrating the contributions and assessing whether the initially proposed objectives were met, some suggestions for future work are included.

6.1 Conclusions

The main objective of this thesis was to develop a system for scheduling and performing underwater pick & place missions, along with the development of the scheduling algorithms that consider relevant parameters to underwater conditions such as the distance of the end-effector, the total movement of the joints, the security, the total planning time and the maximum torque of the joints. The developed work possesses a ROS framework to simulate and schedule the missions, allowing their validation before performing them in the real scenario. The system and the algorithms, once completed, were validated through a series of tests in a highly congested pick & place scenario in dry and underwater conditions.

The movements were planned considering the motion planning algorithms RRT Connect, BKPIECE and PRM. These originate a series of points that correspond to the Bravo 7 joints positions over time. The planned trajectories are used to run the scheduling algorithms: Dijkstra's greedy method, GRASP and A*. These will return the best path found, according to the cost that was calculated with the metrics and corresponding weights.

The results proved that the Dijkstra's algorithm obtained the worst results in terms of cost. However, it has the lowest execution time of all algorithms, which means that it can be useful in real-time scenarios where the execution time is more important than the trajectory itself. In addition, it was possible to conclude that the GRASP algorithm can reach similar or even better results than the A* method. However, the quality of the results highly depend on the number of iterations that the algorithm presents. Furthermore, the execution time of the algorithm increases almost linearly with the number of iterations, which means that in order to have similar or better results than the A* method, the method needs a very high execution time.

It was also possible to conclude that the A* method presents the best results of all algorithms in

almost all scenarios. Even though it is possible for the method to get stuck in a local optimum, the results showed that the method has the best results overall. In addition, A* presents a low execution time compared to the GRASP method while getting low costs and a higher execution time than Dijkstra's algorithm but with a better result.
Furthermore, it was possible to observe that the developed architecture adapted to the desired scenarios by finding the best path according to the designated weights and metrics. This was validated with multiple scenarios with different characteristics and focus points.
Finally, it was possible to validate the developed work in a real underwater scenario using the built structure and objects, along with the use of the motion capture system Qualisys as a ground truth. The results proved that the system worked and that its error was very low.

6.2 Future Work

As part of a more comprehensive ongoing project, future work is foreseen to implement the image processing module to allow the estimation of the positions of the objects and respective bases.
With the objective of increasing the level of the state of the art in the inspection and maintenance of offshore wind farms, one of the primary next steps is to combine the current architecture with the control and planning of an AUV. With this, the system has to consider the AUV's movements and characteristics. In addition, the AUV may possess additional sensors that can be used for object detection and pose estimation.
This work studied three main algorithms used for path planning: Dijkstra's greedy, GRASP and A*. As a way of increasing usability, efficiency, and cost-effectiveness, more scheduling and path planning algorithms, such as genetic algorithms, can be studied and implemented, allowing a more significant number of methods to be evaluated. Other metrics, such as the execution time on the real manipulator, may also be studied to evaluate the trajectories, increasing the number of elements when calculating the costs.

Appendix A

Sensibility Test Results

The results for the sensibility tests are represented in Tables A.1 to A.5. The results were obtained by using the motion plan algorithm RRT Connect and 1000 iterations of the GRASP method. The used algorithm, final path, metric and corresponding weights, normalised value and the final cost, are shown in each table.

A.1 Sensibility Test Result Tables

Algorithm	Final Path	Metric	K	N_{metric}	C_{final}
Dijkstra	0-4-8-3-7-2-5-0-1-6-9	Planning Time	1	2.207	2.207
		Movement Joints	0	2.032	
		EE-Distance	0	3.676	
		Security	0	3.261	
		Maximum Torque	0	5.244	
GRASP	0-4-8-3-7-1-6-2-5-9	Planning Time	1	1.991	1.991
		Movement Joints	0	1.750	
		EE-Distance	0	3.094	
		Security	0	2.875	
		Maximum Torque	0	5.928	
A*	0-2-5-1-6-0-4-8-3-7-9	Planning Time	1	2.164	2.164
		Movement Joints	0	1.907	
		EE-Distance	0	3.719	
		Security	0	3.343	
		Maximum Torque	0	5.738	

Table A.1: Sensibility test - First set of of weights

Algorithm	Final Path	Metric	K	N_{metric}	C_{final}
Dijkstra	0-2-5-0-1-6-3-7-0-4-8-9	Planning Time	0	2.411	1.679
		Movement Joints	1	1.679	
		EE-Distance	0	3.273	
		Security	0	4.224	
		Maximum Torque	0	6.229	
GRASP	0-1-5-0-2-6-4-8-3-7-9	Planning Time	0	3.294	1.043
		Movement Joints	1	1.043	
		EE-Distance	0	2.895	
		Security	0	3.551	
		Maximum Torque	0	6.312	
A*	0-4-8-0-1-5-0-2-6-3-7-9	Planning Time	0	3.464	1.141
		Movement Joints	1	1.141	
		EE-Distance	0	2.881	
		Security	0	4.225	
		Maximum Torque	0	6.267	

Table A.2: Sensibility test - Second set of of weights

Algorithm	Final Path	Metric	K	N_{metric}	C_{final}
Dijkstra	0-4-8-0-2-5-0-3-7-0-1-6-9	Planning Time	0	2.594	3.966
		Movement Joints	0	2.041	
		EE-Distance	1	3.966	
		Security	0	4.602	
		Maximum Torque	0	5.741	
GRASP	0-4-8-3-7-2-6-1-5-9	Planning Time	0	3.000	2.598
		Movement Joints	0	1.200	
		EE-Distance	1	2.598	
		Security	0	2.791	
		Maximum Torque	0	5.280	
A*	0-4-8-3-7-0-2-6-1-5-9	Planning Time	0	3.149	2.659
		Movement Joints	0	1.049	
		EE-Distance	1	2.659	
		Security	0	3.503	
		Maximum Torque	0	5.777	

Table A.3: Sensibility test - Third set of of weights

Algorithm	Final Path	Metric	K	N_{metric}	C_{final}
Dijkstra	0-3-7-1-6-4-8-2-5-9	Planning Time	0	3.031	2.506
		Movement Joints	0	1.869	
		EE-Distance	0	3.402	
		Security	1	2.506	
		Maximum Torque	0	5.296	
GRASP	0-3-7-4-8-2-5-1-6-9	Planning Time	0	3.019	2.321
		Movement Joints	0	2.190	
		EE-Distance	0	4.053	
		Security	1	2.321	
		Maximum Torque	0	5.296	
A*	0-3-7-4-8-2-5-1-6-9	Planning Time	0	3.019	2.321
		Movement Joints	0	2.190	
		EE-Distance	0	4.053	
		Security	1	2.321	
		Maximum Torque	0	5.296	

Table A.4: Sensibility test - Fourth set of of weights

Algorithm	Final Path	Metric	K	N_{metric}	C_{final}
Dijkstra	0 4 8 0 3 7 0 2 5 0 1 6 9	Planning Time	0	2.594	5.741
		Movement Joints	0	2.041	
		EE-Distance	0	3.966	
		Security	0	4.602	
		Maximum Torque	1	5.741	
GRASP	0 4 8 1 6 0 3 7 2 5 9	Planning Time	0	2.464	4.755
		Movement Joints	0	2.100	
		EE-Distance	0	3.671	
		Security	0	3.171	
		Maximum Torque	1	4.755	
A*	0 4 8 0 3 7 2 5 1 6 9	Planning Time	0	2.254	5.242
		Movement Joints	0	2.219	
		EE-Distance	0	3.885	
		Security	0	3.261	
		Maximum Torque	1	5.243	

Table A.5: Sensibility test - Fifth set of weights

Appendix B

Testing the system

After testing the developed work in a simulated environment, the next step was to test the movements, control and planning in the real Bravo 7. In order to perform the tests, these were divided into *dry tests* and *underwater tests*.

B.1 Dry Tests

In order to perform the *dry tests*, the work was divided into simpler and more general steps, starting without the developed architecture:

1. Communicating with the Bravo 7 manipulator through Ethernet;
2. Moving the manipulator with a pre-defined set of joint values;
3. Moving the manipulator to a defined pose, containing position and orientation;
4. Moving the gripper of the Bravo;
5. Grabbing an object with the gripper;
6. Moving the Bravo while grabbing an object;
7. Performing pick & place tasks with three different objects, each with its final pre-defined position.

Only after testing the scenarios with only one node the final architecture was developed and tested in a *dry scenario*:

1. Performing pick & place tasks with three different objects, each one with its final destination;
2. Performing pick & place tasks with the final four objects, where the two cubes can go to either final destination.

When these tests were performed, the *dry tests* were considered concluded.

References

[1] Paul Denholm and Trieu Mai. Timescales of energy storage needed for reducing renewable energy curtailment. *Renewable Energy*, 130:388–399, 2019.

[2] A. Martinez and G. Iglesias. Multi-parameter analysis and mapping of the levelised cost of energy from floating offshore wind in the mediterranean sea. *Energy Conversion and Management*, 243:114416, 2021.

[3] Zhiyu Jiang. Installation of offshore wind turbines: A technical review. *Renewable and Sustainable Energy Reviews*, 139:110576, 2021.

[4] Daniel Campos, Aníbal Matos, and Andry Pinto. Multi-domain inspection of offshore wind farms using an autonomous surface vehicle. *SN Applied Sciences*, 3, 2021.

[5] A. M. Pinto, J. V. A. Marques, D. F. Campos, N. Abreu, A. Matos, M. Jussi, R. Berglund, J. Halme, P. Tikka, J. Formiga, C. Verrecchia, S. Langiano, C. Santos, Sá N, J. J. Stoker, F. Calderoni, S. Govindaraj, A. But, L. Gale, D. Ribas, N. Hurtós, E. Vidal, P. Ridao, P. Chieslak, N. Palomeras, S. Barberis, and L. Aceto. Atlantis - the atlantic testing platform for maritime robotics. In *OCEANS 2021: San Diego – Porto*, pages 1–5.

[6] Ying-Hao Yu and Ya-Tang Zhang. Collision avoidance and path planning for industrial manipulator using slice-based heuristic fast marching tree. *Robotics and Computer-Integrated Manufacturing*, 75:102289, 2022.

[7] Z. Arkouli, P. Aivaliotis, and S. Makris. Towards accurate robot modelling of flexible robotic manipulators. *Procedia CIRP*, 97:497–501, 2021.

[8] Satja Sivčev, Matija Rossi, Joseph Coleman, Gerard Dooly, Edin Omerdić, and Daniel Toal. Fully automatic visual servoing control for work-class marine intervention rovs. *Control Engineering Practice*, 74:153–167, 2018.

[9] Andry Maykol Pinto and Anibal C. Matos. Maresye: A hybrid imaging system for underwater robotic applications. *Information Fusion*, 55:16–29, 2020.

[10] Andry Pinto, A. Moreira, Miguel Correia, and Paulo Costa. A flow-based motion perception technique for an autonomous robot system. *Journal of Intelligent and Robotic Systems*, 75, 2013.

[11] Satja Sivčev, Joseph Coleman, Edin Omerdić, Gerard Dooly, and Daniel Toal. Underwater manipulators: A review. *Ocean Engineering*, 163:431–450, 2018.

[12] Gayathri Rajendran, Uma V, and Bettina O'Brien. Unified robot task and motion planning with extended planner using ros simulator. *Journal of King Saud University - Computer and Information Sciences*, 2021.

[13] Andry Maykol Pinto, Miguel V. Correia, A. Paulo Moreira, and Paulo G. Costa. Unsupervised flow-based motion analysis for an autonomous moving system. *Image and Vision Computing*, 32(6):391–404, 2014.

[14] Mostafa Sharifi, XiaoQi Chen, Christopher Pretty, Don Clucas, and Erwan Cabon-Lunel. Modelling and simulation of a non-holonomic omnidirectional mobile robot for offline programming and system performance analysis. *Simulation Modelling Practice and Theory*, 87:155–169, 2018.

[15] Ajisha Mathias, Samiappan Dhanalakshmi, R. Kumar, and R. Narayanamoorthi. Underwater object detection based on bi-dimensional empirical mode decomposition and gaussian mixture model approach. *Ecological Informatics*, 66:101469, 2021.

[16] P. N. Leite and A. M. Pinto. Exploiting motion perception in depth estimation through a lightweight convolutional neural network. *IEEE Access*, 9:76056–76068, 2021.

[17] A. P. O. Afonso and A. M. Pinto. Underwater object recognition: A domain-adaption methodology of machine learning classifiers. In *OCEANS 2019 MTS/IEEE SEATTLE*, pages 1–6.

[18] Anton Umek and Anton Kos. Validation of mems accelerometer for rapid hand movement measurement. *Procedia Computer Science*, 187:530–537, 2021.

[19] G. K. Adam. Hybrid neural controller of a stepper motor for a manipulator arm. In *Proceedings of the Fourth International Workshop on Robot Motion and Control (IEEE Cat. No.04EX891)*, pages 321–326.

[20] Richard Crowder. *8 - Stepper motors*, pages 209–226. Butterworth-Heinemann, 2020.

[21] Y. Chen, D. Huang, W. Jin, X. Gao, and H. Zhang. Precise positioning of power lithium battery assembly manipulator control based on stepper motor. In *2013 IEEE International Conference on Mechatronics and Automation*, pages 716–720.

[22] The difference between: Absolute and incremental encoders in application. *Machine Design*, 90(11):78–80, 2018.

[23] Goran S. Miljkovic and Dragan B. Denic. Redundant and flexible pseudorandom optical rotary encoder. *Elektronika ir Elektrotechnika*, 26(6):10–16, 2020.

[24] Xiaohua Shi, Yu Guo, Xuechan Chen, Ziming Chen, and Zhiwei Yang. Kinematics and singularity analysis of a 7-dof redundant manipulator. *Sensors (14248220)*, 21(21):7257, 2021.

[25] Adelhard Beni Rehiara. Kinematics of adeptthree robot arm. In Satoru Goto, editor, *Robot Arms*, chapter 2. IntechOpen, Rijeka, 2011.

[26] S. M. Lavalle. Motion planning: Part i: The essentials. *IEEE Robotics and Automation Magazine*, 18(1):79–89, 2011. Cited By :74 Export Date: 19 February 2022.

[27] J. Evans, P. Redmond, C. Plakas, K. Hamilton, and D. Lane. Autonomous docking for intervention-auvs using sonar and video-based real-time 3d pose estimation. In *Oceans 2003. Celebrating the Past ... Teaming Toward the Future (IEEE Cat. No.03CH37492)*, volume 4, pages 2201–2210 Vol.4.

[28] Y. Wang, S. Wang, Q. Wei, M. Tan, C. Zhou, and J. Yu. Development of an underwater manipulator and its free-floating autonomous operation. *IEEE/ASME Transactions on Mechatronics*, 21(2):815–824, 2016.

[29] D. Youakim, P. Ridao, N. Palomeras, F. Spadafora, D. Ribas, and M. Muzzupappa. Moveit!: Autonomous underwater free-floating manipulation. *IEEE Robotics Automation Magazine*, 24(3):41–51, 2017.

[30] N. Palomeras, A. Peñalver, M. Massot-Campos, G. Vallicrosa, P. L. Negre, J. J. Fernández, P. Ridao, P. J. Sanz, G. Oliver-Codina, and A. Palomer. I-auv docking and intervention in a subsea panel. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2279–2285.

[31] S. Hernandez-Mendez, C. Maldonado-Mendez, A. Marin-Hernandez, H. V. Rios-Figueroa, H. Vazquez-Leal, and E. R. Palacios-Hernandez. Design and implementation of a robotic arm using ros and moveit! In *2017 IEEE International Autumn Meeting on Power, Electronics and Computing (ROPEC)*, pages 1–6.

[32] A. Sharp, K. Kruusamäe, B. Ebersole, and M. Pryor. Semiautonomous dual-arm mobile manipulator system with intuitive supervisory user interfaces. In *2017 IEEE Workshop on Advanced Robotics and its Social Impacts (ARSO)*, pages 1–6.

[33] Y. J. Kim, J. H. Wang, S. Y. Park, J. Y. Lee, J. J. Kim, and J. J. Lee. A rrt-based collision-free and occlusion-free path planning method for a 7dof manipulator. In *2014 IEEE International Conference on Mechatronics and Automation*, pages 1017–1021.

[34] H. Kim, H. Seo, J. Kim, and H. J. Kim. Sampling-based motion planning for aerial pick-and-place. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7402–7408.

[35] Y. Huang and K. Gupta. Collision-probability constrained prm for a manipulator with base pose uncertainty. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1426–1432.

[36] F. Yan, Y. Zhuang, and J. Xiao. 3d prm based real-time path planning for uav in complex environment. In *2012 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 1135–1140.

[37] R. Bohlin and L. E. Kavraki. Path planning using lazy prm. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, volume 1, pages 521–528 vol.1.

[38] I. A. Sucan and L. E. Kavraki. A sampling-based tree planner for systems with complex dynamics. *IEEE Transactions on Robotics*, 28(1):116–131, 2012.

[39] S. Thakar, P. Rajendran, A. M. Kabir, and S. K. Gupta. Manipulator motion planning for part pickup and transport operations from a moving base. *IEEE Transactions on Automation Science and Engineering*, 19(1):191–206, 2022.

[40] Eurico Pedrosa, Nuno Lau, Artur Pereira, and Bernardo Cunha. A skill-based architecture for pick and place manipulation tasks. In Francisco Pereira, Penousal Machado, Ernesto Costa, and Amílcar Cardoso, editors, *Progress in Artificial Intelligence*, pages 457–468. Springer International Publishing.

[41] André Leite, Andry Pinto, and Aníbal Matos. A safety monitoring model for a faulty mobile robot. *Robotics*, 7:32, 2018.

[42] Hanfu Wang and Weidong Chen. Task scheduling for transport and pick robots in logistics: a comparative study on constructive heuristics. *Autonomous Intelligent Systems*, 1(1):17, 2021.

[43] Ellur Anand and Ramasamy Panneerselvam. Literature review of open shop scheduling problems. *Intelligent Information Management*, 07:33–52, 2015.

[44] Ilya Chernykh, Alexander Kononov, and Sergey Sevastyanov. Efficient approximation algorithms for the routing open shop problem. *Computers Operations Research*, 40(3):841–847, 2013.

[45] Heidemarie Bräsel and Holger Hennes. On the open-shop problem with preemption and minimizing the average completion time. *European Journal of Operational Research*, 157(3):607–619, 2004.

[46] Igor Averbakh, Oded Berman, and Ilya Chernykh. The routing open-shop problem on a network: Complexity and approximation. *European Journal of Operational Research*, 173(2):531–539, 2006.

[47] Chinyao Low and Yuling Yeh. Genetic algorithm-based heuristics for an open shop scheduling problem with setup, processing, and removal times separated. *Robotics and Computer-Integrated Manufacturing*, 25(2):314–322, 2009.

[48] S. D. Han, S. W. Feng, and J. Yu. Toward fast and optimal robotic pick-and-place on a moving conveyor. *IEEE Robotics and Automation Letters*, 5(2):446–453, 2020.

[49] R. Mattone, M. Divona, and A. Wolf. Sorting of items on a moving conveyor belt. part 2: performance evaluation and optimization of pick-and-place operations. *Robotics and Computer-Integrated Manufacturing*, 16(2):81–90, 2000.

[50] V. Roshanaei, M. M. Seyyed Esfehani, and M. Zandieh. Integrating non-preemptive open shops scheduling with sequence-dependent setup times using advanced metaheuristics. *Expert Systems with Applications*, 37(1):259–266, 2010.

[51] Levi R. Abreu, Jesus O. Cunha, Bruno A. Prata, and Jose M. Framinan. A genetic algorithm for scheduling open shops with sequence-dependent setup times. *Computers Operations Research*, 113:104793, 2020.

[52] J. Li, Y. Liu, H. Li, Z. Yuan, C. Fu, J. Yue, X. Feng, C. J. Xue, J. Hu, and H. Yang. Path: Performance-aware task scheduling for energy-harvesting nonvolatile processors. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 26(9):1671–1684, 2018.

[53] Alvaro Neuenfeldt Júnior, Lucas Rebouças Guimarães, Maria Federal University of Santa, and Education Federal Institute of. A greedy randomized adaptive search procedure application to solve the travelling salesman problem. *International Journal of Industrial Engineering and Management*, 10(3):238–242, 2019.

[54] Ke Zhang, Chun-Ming Yuan, Xiao-Shan Gao, and Hongbo Li. A greedy algorithm for feedrate planning of cnc machines along curved tool paths with confined jerk. *Robotics and Computer-Integrated Manufacturing*, 28(4):472–483, 2012.

[55] Liangsheng Qu and Ruixiang Sun. A synergetic approach to genetic algorithms for solving traveling salesman problem. *Information Sciences*, 117(3):267–283, 1999.

[56] Y. Wei, Y. Hu, and K. Gu. Parallel search strategies for tsps using a greedy genetic algorithm. In *Third International Conference on Natural Computation (ICNC 2007)*, volume 3, pages 786–790.

[57] Gang Tang, Congqiang Tang, Christophe Claramunt, Xiong Hu, and Peipei Zhou. Geometric a-star algorithm: An improved a-star algorithm for agv path planning in a port environment. *IEEE access*, 9:59196–59210, 2021.

[58] Elizabeth F. Gouvêa Goldbarg, Marco C. Goldbarg, and Joã P. F. Farias. *Grasp with Path-Relinking for the Tsp*, pages 137–152. Springer US, Boston, MA.

[59] R. Ragel, I. Maza, F. Caballero, and A. Ollero. Comparison of motion planning techniques for a multi-rotor uas equipped with a multi-joint manipulator arm. In *2015 Workshop on Research, Education and Development of Unmanned Aerial Systems (RED-UAS)*, pages 133–141.

[60] Alejandra Casado, Sergio Pérez-Peló, Jesús Sánchez-Oro, and Abraham Duarte. A grasp algorithm with tabu search improvement for solving the maximum intersection of k-subsets problem. *Journal of Heuristics*, 28(1):121–146, 2022.

[61] A. Cowley, B. Cohen, W. Marshall, C. J. Taylor, and M. Likhachev. Perception and motion planning for pick-and-place of dynamic objects. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 816–823.

[62] Jing Zhang, Jun Wu, Xiao Shen, and Yunsong Li. Autonomous land vehicle path planning algorithm based on improved heuristic function of a-star. *International journal of advanced robotic systems*, 18(5):172988142110427, 2021.

[63] Renato Silva, Aníbal Matos, and Andry Pinto. Multi-criteria metric to evaluate motion planners for underwater intervention. *Autonomous Robots*, pages 1–13, 2022.

[64] Nadav Eichler, Hagit Hel-Or, Ilan Shimshoni, Dorit Itah, Bella Gross, and Shmuel Raz. 3d motion capture system for assessing patient motion during fugl-meyer stroke rehabilitation testing. *IET computer vision*, 12(7):963–975, 2018.

[65] Martin Magdin. Simple mocap system for home usage. *International journal of interactive multimedia and artificial intelligence*, 4(4):80, 2017.

[66] N. T. Kostov, S. M. Yordanova, and Y. D. Kalchev. Mocap - the advantages of accelerometers and accuracy improvement. 9:60–64, 2015.