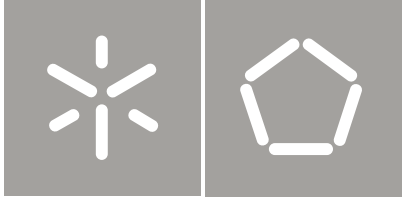
Universidade do Minho
Escola de Engenharia

Pedro Gonçalves da Costa

Motion Planning in Cartesian Space for the Collaborative Redundant Robot Sawyer

abril de 2019

Motion Planning in Cartesian Space for the Collaborative Redundant Robot Sawyer
Pedro Gonçalves da Costa
UMinho | 2019



Universidade do Minho
Escola de Engenharia

Pedro Gonçalves da Costa

Motion Planning in Cartesian Space for the Collaborative Redundant Robot Sawyer

Dissertação de Mestrado
Ciclo de Estudos Integrados Conducentes ao Grau de Mestre em Engenharia Eletrónica Industrial e Computadores

Trabalho efectuado sob a orientação do
Professor Doutor Sérgio Paulo Carvalho Monteiro

abril de 2019

Agradecimentos

Em primeiro lugar, gostaria de agradecer ao Professor Doutor Sérgio Monteiro pela oportunidade de desenvolver este projeto de dissertação, por todo o apoio técnico-cientifico , motivação e disponibilidade ao longo de todo o projeto.

Especialmente, queria agradecer aos meus colegas de laboratório Sara Sá e Gianpaolo Gulleta, pelos conhecimentos que me transmitiram, e pelo tempo disponibilizado para me ajudar na resolução de diversos problemas que surgiram ao longo deste projeto.

A todos os meus restantes colegas do Laboratório de Robótica Móvel e Antropomórfica da Universidade do Minho, com quem convivi, gostaria de deixar a minha gratidão pelo bom ambiente e espírito de entreajuda. Muito obrigado Carlos, Paulo, Luís, Flora, Tiago e Weronika!

Por último, mas não menos importante, um profundo reconhecimento à minha família, em especial aos meus pais e irmãos, pela dedicação e apoio incondicional em toda a minha vida, e especial incentivo e compreensão ao longo da realização desta tese. Quero agradecer em particular à minha namorada e melhor amiga Cristiana, que esteve sempre presente, e que de forma excecional e paciente me compreendeu, incentivou e ajudou ao longo deste percurso, demonstrando um apoio e dedicação incondicional.

A todos um "Muito Obrigado" por terem acreditado em mim e no meu trabalho, mesmo quando eu próprio não acreditava. Espero que esta etapa, que agora termino possa, de algum modo, compensar e retribuir todo o apoio e carinho que me deram.

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

Resumo

Planeamento de Movimento no Espaço Cartesiano para o Robô Colaborativo Redundante Sawyer

Ao longo dos anos, a investigação no ramo da robótica industrial tem vindo a desenvolver soluções para otimizar os processos de fabrico. Neste contexto, surge este projeto de dissertação cujo objetivo é desenvolver um planeador Cartesiano sujeito a um conjunto de restrições especificadas pelo utilizador para uma determinada tarefa, que será desempenhada pelo robô colaborativo Sawyer.

O objetivo desta dissertação é estudar o problema do planeamento de caminhos sujeito a restrições, essencialmente associadas ao raio minímo de curvatura. O caso de estudo consiste na deposição de um filamento de fibra ótica num PCB. O objetivo é utilizar a fibra ótica como um sensor no controlo da qualidade dos PCBs produzidos. Para tal, este projeto de dissertação divide-se em duas subtarefas, nomeadamente, a geração de caminhos em PCB's e a geração da trajetória para execução pelo robô. Quanto ao planeamento de caminho, este é sujeito a um conjunto de restrições, tais como: (i) vários pontos alvo no mesmo caminho; (ii) segmentos de reta em cada um dos pontos alvo e (iii) o caminho final tem que validar os raios de curvatura mínimos definidos. Relativamente à geração da trajetória para o robô Sawyer, esta deve acompanhar o caminho gerado. Para tal, a trajetória final deve ter um movimento suave, evitando singularidades e colisões com os obstáculos presentes no cenário.

A validação do sistema foi realizada em ambiente virtual e laboratorial, através da execução da tarefa em que o robô simula o depósito de um filamento sobre uma placa de circuito impresso. Os resultados mostraram que o método de planeamento de caminhos desenvolvido é capaz de encontrar uma solução que respeite todas as restrições impostas. Além disso, a trajetória criada para o sistema robótico Sawyer, permitiu acompanhar o caminho desejado.

Palavras-chave: planeamento de caminho, espaço cartesiano, planeamento de movimento, robô Sawyer.

Abstract

Motion Planning in Cartesian Space for the Collaborative Redundant Robot Sawyer

Over the past few decades, the research in the field of industrial robotics has been developing solutions to optimize manufacturing processes. In this context, this project of dissertation arises with the aim of developing a Cartesian planner subject to a set of restrictions specified by the user for a certain task, that will be performed by the collaborative robot Sawyer.

The aim of this dissertation is to study the problem of the path planning subject to restrictions, essentially associated with the minimum radius of curvature. The case of study consists on the deposition of an optic fibre filament in a PCB. The objective is to use the optic fibre as a sensor in the quality control of the produced PCBs. For this purpose, this dissertation project is divided in two subtasks, namely, the path generation in the PCB and the robot trajectory generation. As for the path planning, it is subject to a set of constraints, such as: (i) several target points in the same path; (ii) straight linear segments at each target point, and (iii) the final path must validate the defined minimum radius of curvature. Regarding the trajectory generation for the Sawyer robot, it should follow the generated path. For this, the final trajectory must have a smooth movement, avoiding singularities and collisions with the obstacles present in the scenario.

The validation of the system was performed in a virtual and laboratory environment, through the execution of the task in which the robot simulates the placement of a filament on a PCB. The results showed that the developed path planning method is able to find a solution that accommodate all the imposed constraints. Furthermore, the trajectory created for the robotic system Sawyer, allowed to follow the desired path.

Keywords: path planning, cartesian space, motion planning, Sawyer robot.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Over the last 60 years, the area of robotics has been focused on the development of new solutions that meet to the humans technical needs and, consequently, improve their life quality. Robotics applications can be found in almost every area of modern life. Robotic systems can be divided into two main categories, namely, industrial and service [1]. Industrial robots are designed to be used in manufacturing processes, such as industrial assembly lines [2]. On the other hand, service robots are designed to assist the needs established by the humans in their task. There are a wide range of applications for the service robots, such as: (i) medical surgeries [3]; rehabilitation and heath care [4]; (ii) exploration, such as space exploration [5] and underwater exploration [6], and rescue [7], (iii) hazourdous [8] and disasters environments [9]; (iv) agriculture and forestry [10], (v) construction [11], and (vi) domestic applications [12]. Figure 1.1 illustrates some of the robotics applications.

The majority of the robots used in the mentioned applications are robotic manipulators or mobile robots. Regardless the application, robots may perform their tasks in structured and known environments or have to adapt to the surrounding environment. One example of the first case is the use of robotic manipulators installed in the facilities as they are constantly executing the tasks in the station of the industrial assembly line. In other applications, such as the exploration with mobile robots, the environment is unknown *a priori*, and thus, the robot constructs a representation of the environment model with the information provided by its sensors, while heading to a target position and avoiding possible collisions with obstacles that may arise. For both scenarios, robot follows a set of target points, which represents the path for the trajectory. Therefore, the task can be divided into path and motion planning, where the first

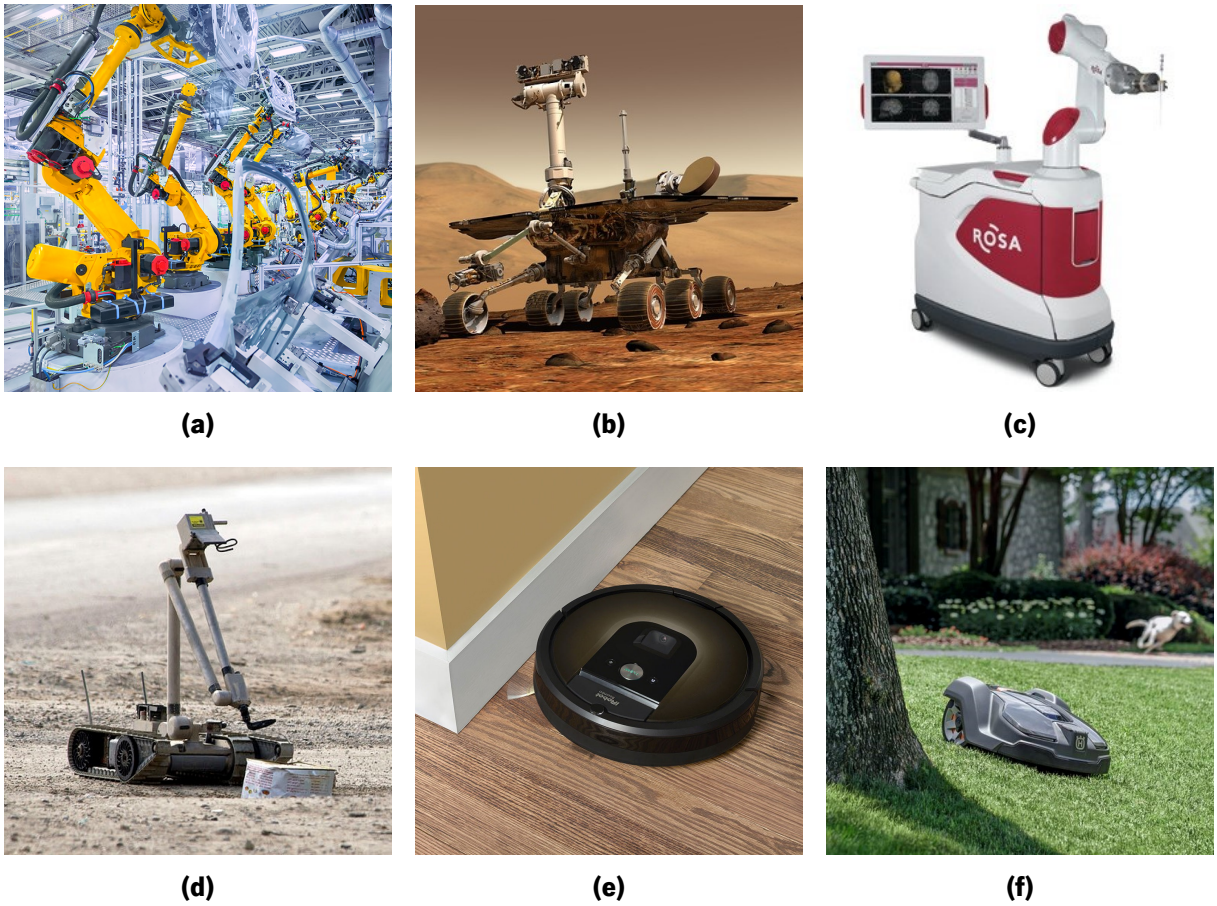


Figure 1.1: Examples of robotics applications: (a) industrial robots in a assembly line; (b) space exploration – Opportunity robot; (c) medical surgeries – ROSA robot; (d) hazardous environments – iRobot Packbot ; (e) and (f) domestic robots – vacuum cleaner and lawn mower, respectively.

one represents the planner that generates the path according to specified target positions and to another constraints imposed by the object, the robot or the human, whereas the motion planning corresponds to the generated trajectory to execute the task.

This dissertation project focuses particularly on the development of a path planner restricted to constraints, that should be performed by the robot. The following section will contextualize the path and motion problem, regarding to industrial robots. Besides that, it describes the evolution of these robots in the industrial sector. Then, Section 1.1 will present the motivations and the main goal associated to the development of the present dissertation project. Lastly, Section 1.3 presents the outline of the present dissertation.

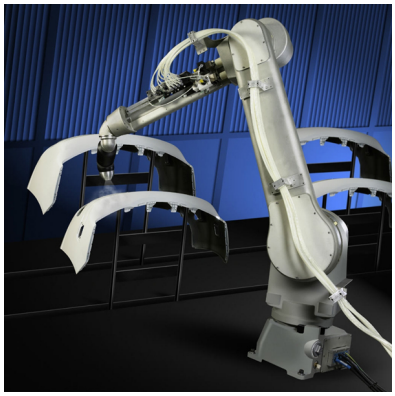
1.1 Motivations and Goal

Nowadays, most industrial activities have been searching for robotic solutions to automate their processes and/or tasks, through the settlement of robotic systems in their infrastructures [13]. According to the International Federation of Robotics (IFR), it is expected that the sale of industrial robots will continue to grow steadily by 2021, and it is predicted to reach 630 000 units in the same year. Between 2018 and 2021, it is estimated that almost 2.1 million of new industrial robots will be installed in factories around the world [14].

Over the years, the research in the industrial robotics field has been developing solutions to optimize the manufacturing processes, in particular, using collaborative robotics [15, 16]. This solution resides in the combination of the human operator and the robot's abilities, obtaining solutions of production that stand out by: (i) the increase of productivity; (ii) the improvement of the products' quality; (iii) the reduction of fabrication costs; and (iv) the replacement of heavy and exhausting tasks for the human operator.

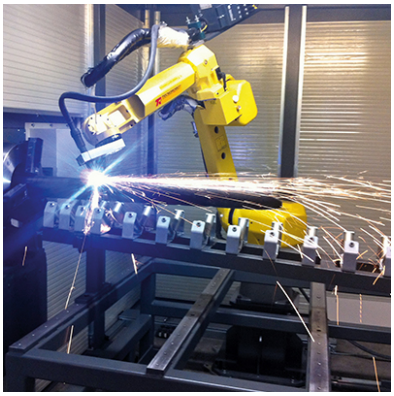
As already mentioned, industries have been increasingly investing in the automation of their processes in order to improve the production rate, the products' quality and the production costs, as well as decreasing the number of exhaustive and heavy tasks performed by the operators. The main industrial branch that has been investing in this approach, improving its assembly lines is the automotive industry, in order to face the current market [14]. This industry is a good example due to the variety of processes and tasks that are performed in its assembly lines, such as material handling (Figure 1.2b), welding (Figure 1.2c), assembly of components (Figure 1.2d), spray painting (Figure 1.2a) and others [17]. Each of these processes is well characterized and structured, and the robot must be programmed according to the requirements and constraints of the task. One of the main problems is the definition of the path that should be performed by the robot and depends on the application requirements. The welding procedure, for example, includes two components, the path where the robot must weld the component and the trajectory that the robot must perform to follow the specified path [18, 19]. Another example is the spray painting, and just as in the previous application, the path must be previously planned depending on the type of material and surface that have to be painted or other requirements of the task. Some researchers, such as in [20–22], have been working in this problem and developing strategies for the path planning problem of spray painting.

Figure 1.2 illustrates the most common tasks performed on cars assembly lines.



(a) Painting

(b) Handling



(c) Welding

(d) Assembly

(e) Collaborative Robots

Figure 1.2: Typical industrial robot applications.

The case of study under development in this dissertation project consists on the placement of an optical fibre filament in a Printed Circuit Board (PCB). The purpose is to use the optical fibre as a sensor in the quality control of the produced PCBs (vibration or temperature). This task is divided into subtasks, namely the geometric path generation and the robot trajectory generation. The Path Planner generates the path according to the list of rules and constraints and the Motion Planner creates the trajectory that has to follow the generated path for the placement of the optical fibre filament.

Even though the planner is developed for this specific application, it can also be used for solving other problems. There are others applications where this planner could be applied, such as mobile robots, or other applications where the path is subject to constraints. The following section explains in detail the problem of this dissertation work.

1.2 Problem Definition

The task assigned in this research project, introduced in Section 1.1, portrays a path planning problem, where the main objective is to find an optimal path that covers all constraints and rules established by the operator, for the placement of an optical fibre filament.

In order to develop a solution, it is required to take into account the mechanical constraints of the material that will be used in the task. In this case, it will be used an optical fibre filament as the placement material in the PCB. The characteristics such as the thickness and the flexibility of the material must be considered. Despite its flexibility, it has some limitations regarding the radius of curvature, thus, it must be guaranteed that the curvatures radius does not go under the minimum bend radius specified by the manufacturer. This constraint ensures that the optical fibre maintains the internal reflectivity required for the travelling optical flow.

Besides the mechanical constraints of the material, the operator must specify other constraints and/or rules for the path planner. The rules that must be defined are the: (i) position of each waypoint in 2D space, represented by the coordinates (x, y); (ii) waypoints sequence, as the order that the path has to follow; and (iii) length of the straight line segment, all the waypoints have a straight line segment centred on the waypoint origin.

An overview of the inputs for the path planning algorithm and the expected outputs of the planner are summarized in the following topics:

Inputs for the Path Planner

- The PCB mask, which is later converted into configuration space;
- The waypoints coordinates in 2D space (x, y) and their sequence;
- The size and maximum bend radius of curvature of the placement material;
- The length of the straight line segment, which is centred in each waypoint.

Outputs from the Path Planner

- A continuous and smooth path that crosses all the defined waypoints following their sequence, and bounded by constraints on the curvature;
- The built path should avoid collisions with the components present in the PCB.

The path generated by the Path Planner should satisfy the mechanical constraints induced by the material used for the placement in the PCB. Further, it should be continuous and bounded in terms of the curvature, thus guaranteeing smooth path. Besides that, the path should avoid collisions with the obstacles present in the configuration space.

This task will be performed by the robotic platform Sawyer, which will replay the trajectories generated by the Motion Planner. An overview of the inputs for the Motion Planner and the expected outputs are summarized in the following topics:

Inputs for the Motion Planner

- The information about the PCB, such as its dimensions and pose in the environment;
- The generated path by the Path Planner.

Outputs from the Motion Planner

- A smooth and feasible trajectory that avoid the collision with the obstacles represented in the scenario.

The final solution should be generated offline before performing any task. Firstly, the operator should validate the solution in simulation, and then, if it is a valid solution, it outputs the trajectory to be executed by the robot Sawyer.

1.3 Dissertation Outline

The present dissertation is organized in six chapters and each one will be briefly described as follow. Chapter 1 describes the motivation and the goal of the dissertation project, and it also explains the problem definition. Additionally, it presents a contextualization of the problem in the scope of industrial robotics.

Chapter 2 exposes a review of the state of the art in path planning algorithms most used for various applications.

Chapter 3 presents an overview of the system, describing the main modules – Path Planner and Motion Planner. Additionally, it exposes the main features of the robotic platform Sawyer and it also briefly describes the tools and frameworks used.

1.3. Dissertation Outline

Chapter 4 details the developed solution integrated in the Path Planner and characterizes it according to the tests performed.

Chapter 5 refers to the implementation and validation of the Motion Planner. It describes how the trajectories are generated for the Sawyer robot and it shows the results performed to validate the system in a virtual and real environment.

Lastly, Chapter 6 presents an overview of the developed solution, discusses the main conclusions of the project and suggests some topics to be developed as future work.

Chapter 2

Path Planning Methods

This chapter presents a literature review of motion planning algorithms. Section 2.1 introduces the concepts related to motion planning problem, and their classification. The following sections describe a couple of the most used algorithms nowadays. Lastly, the section 2.6 discusses the improvements of the algorithms and the more suitable for this project of path planning.

2.1 Introduction

The robot motion planning corresponds to the process of computing collision free path between the start and target configuration for the robot among obstacles. In the scope of the motion planning, there are two important concepts: the path planning and the trajectory planning. The path planning deals with the designing of only geometric (kinematical) specifications of the positions and orientation of the robot, whereas, the trajectory planning also includes the designing of the linear and angular velocities [23–25].

According to Hwang and Ahuja in [26], the motion planning algorithms can be classified, as global or local. Global methods take into account all the information in the environment, and they plan a motion from the start to the goal configuration. The algorithms of roadmap, cell decomposition and sample based are examples of global methods.

In contrast, the local methods, such as the potential field, are tailored to avoid obstacles in the vicinity of the robot. Thereby, these algorithms rely on the information about the nearly obstacles only. Local methods can be used as a component in a global planner, or as a safety feature to avoid unexpected

obstacles not present in the configuration space, but detected by the sensors during the motion execution [26].

The primary task in any motion planning problem is defining the configuration space (C-space) for a robot [23, 24]. A configuration is known to describe the pose of the robot, and the C-space is the set of all possible configurations. An arbitrary representation of C-space for a motion planning problem is shown in Figure 2.1.

The basic task of any motion planning problem is to find a path from q_I to q_G in C_{free}, which corresponds to the set of configurations that avoids collision with obstacles. C_{obs} defines the obstacle space and the entire blob (Figure 2.1), represented by $C = C_{free} \cup C_{obs}$ [24].

The problem of motion planning is conceptually illustrated in Figure 2.1. The main difficulty is related to the fact it is neither straightforward nor efficient to construct an explicit boundary or solid representation of either C_{free} or C_{obs} [24].

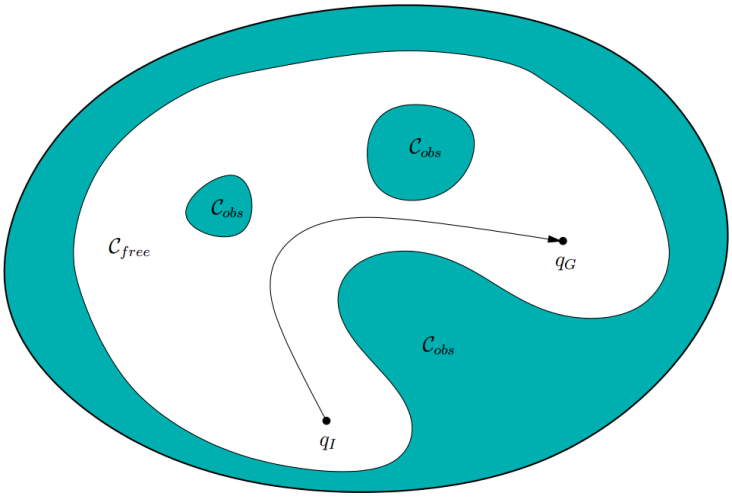


Figure 2.1: The basic motion planning problem is conceptually simple using C-space ideas. The task is to find a path from q_I to q_G in C_{free}. The entire blob represents $C_{space} = C_{free} \cup C_{obs}$. Reprinted from [24].

Several motion planning algorithms exist in the literature, which are categorized into two main branches: the classical path planning methods and the sampling based path planning methods. The mentioned methods are discussed in the following sections.

2.2 Deterministic Methods

The deterministic methods, also called by classic or combinatorial methods, are known for the defining an explicit representation of the obstacles, in the configuration space. This section describes the most known combinatorial methods, which are subdivided into the following categories: cell decomposition, potential fields and roadmaps.

2.2.1 Cell Decomposition

The methods of cell decomposition consist of breaking down the robot's free space in simply-shaped regions, called cells. The adjacency relations between the cells are saved in a connectivity graph, where a node corresponds to a cell extracted from the free space and two nodes are connected by a link if, and only, the two corresponding cells are adjacent; it means that they share a common boundary [23–25, 27].

The problem of the path planning is solved by following two steps: firstly, the planner identifies the cells that contain the start and the goal configurations, and then, it searches for a path within the adjacency graph [23]. This last step is turned in a graph searching problem, and therefore it can be solved by using graph-searching techniques [28].

There are two main techniques of cell decomposition: exact and approximate, which are illustrated in Figure 2.2. When the union of the cells represents exactly the free space, the method is based on the exact cell decomposition (Figure 2.2a). However, in some cases, it is not possible or convenient to compute the free space with the previous method, thus, the approximate cell decomposition must be employed (Figure 2.2b).

2.2.2 Potential Fields

According to Khatib [29], the potential field method is based on two functions, which possess attractive (Figure 2.3a) or repulsive potential (Figure 2.3b), in a proximity of a target or an obstacle, respectively, to construct a collisions free path.

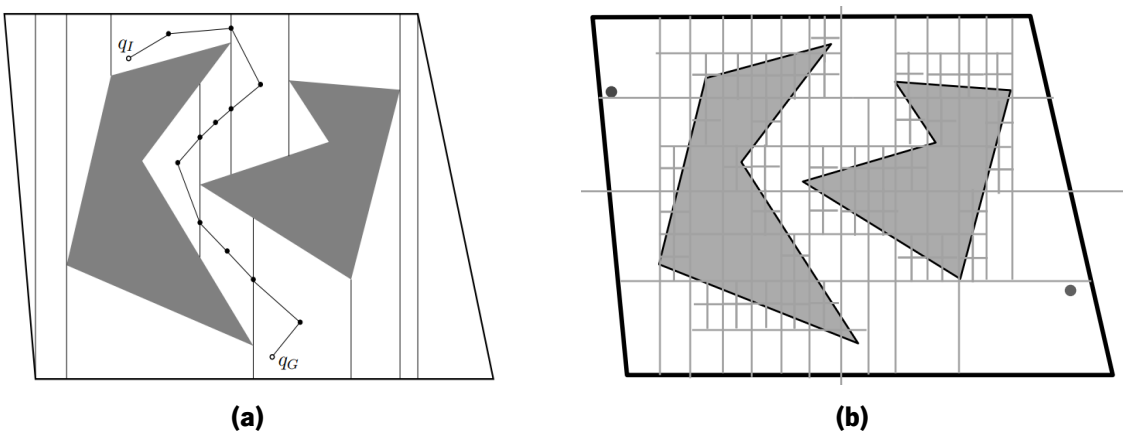


Figure 2.2: Cell decomposition: (a) represents the exact cell decomposition method (trapezoidal decomposition), and (b) presents the approximate cell decomposition method (quadtrees decomposition). The grey areas corresponds to obstacles. Adapted from [24, 28].

The combination of both potentials, attractive force of the target and repulsive force of the obstacles, represents the total potential field (Figure 2.3c). Through this function, it is possible to control the movement of the robot from the start to the goal location, while avoiding the obstacles arranged in the robot workspace [23].

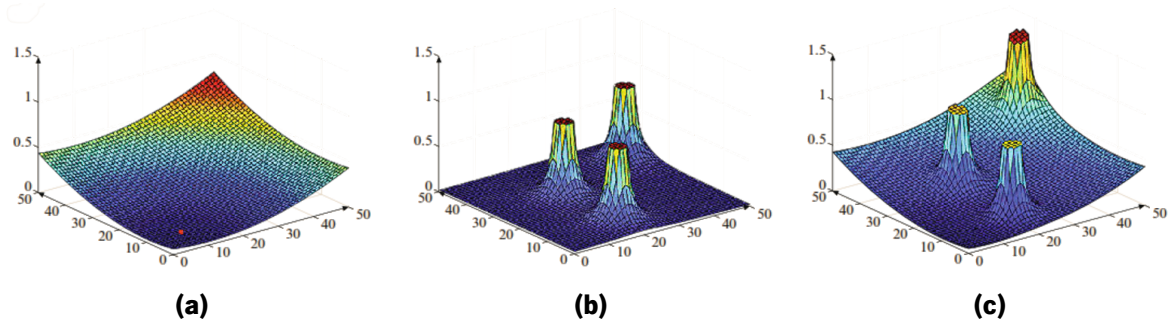


Figure 2.3: Potential field composed by an attractive potential and a repulsive potential. (a) represents an attractive potential of the target; (b) presents a repulsive potential of the obstacles arranged in the workspace; and (c) reports the total potential field as a sum of both attractive and repulsive fields. Reprinted from [30].

This method is widely applied due to its simplicity and speed to find a solution. However, there are some drawbacks; one of the cases is the local minimum shown in Figure 2.4. This problem emerges when both potentials have the same potential, and the robot does not known the trajectory it must follow. To solve this problem, one of the solutions could be the use of a navigation function, which it will generate

random paths for the robot to follow; thus, the potential is not zero, and it can overcome the local minimum problem [31].

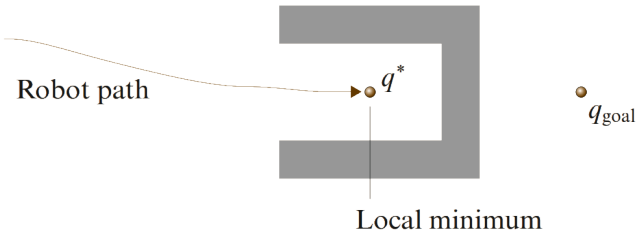


Figure 2.4: Example of the local minimum problem. Reprinted from [30].

2.2.3 Roadmap

The roadmap methods consist of capturing the connectivity of the robot's free space in a network of straight lines segments, or curves, or both. In this scope, there are various proposed methods based on this general idea, which compute different roadmaps [27]. Two of the most known roadmap examples are the visibility graph and the Voronoi diagrams, which are illustrated in Figure 2.5.

Hart *et al.* introduced the visibility graphs in [32] and it is represented in Figure 2.5a. This method consists of a non-directed graph whose nodes are all the vertices of the obstacles. The links in the graph are all straight line segments connecting two nodes that do not intersect the interior of the C_{obs} region. Then, add the nodes of the initial and goal configurations and connect them to the roadmap. The path searches and connects the initial and goal configurations through the roadmap links created, if it exists. The way that this graph is constructed in some areas the path is very closely to the obstacles.

The Voronoi diagram, such as the previous method, creates a network, yet this one has the particularity of having equidistant points of the obstacles, which are bonded by straight line segments and curves, it can be seen in Figure 2.5b. Through this roadmap it is possible to maximize the free space between the obstacles. The Voronoi diagrams allows a path further away from the obstacles, comparatively to the visibility graphs. However, the built path is not shorter than the visibility graphs.

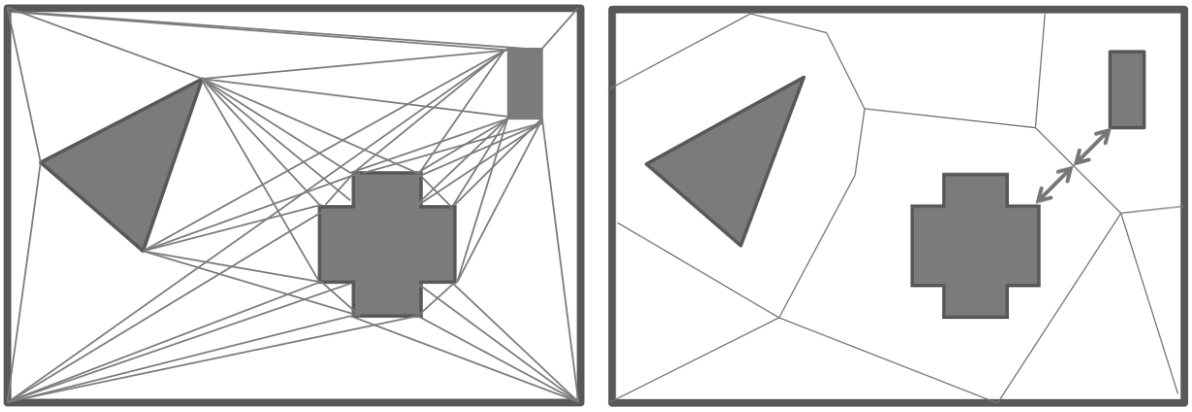


Figure 2.5: Examples of roadmap methods: (a) an example of a visibility graph and (b) an example of a Voronoi diagram. Adapted from [33].

2.3 Sample Based Methods

The sample based methods, also known as probabilistic methods, resort to diverse strategies for generating samples, and for connecting the samples with paths, in order to obtain solutions for path planning problems [23]. The Rapidly Exploring Random Tree (RRT) and Probabilistic Roadmap (PRM) algorithms are the most known and highly used in path planning problems, which are explained in the following sections.

2.3.1 Rapidly Exploring Random Tree - RRT

The Rapidly Exploring Random Tree (RRT) is a probabilist method, where the path is constructed by selecting and joining random points of the map, until a solution is reached, or until expiring the planning time. The RRT algorithm is represented by a tree creation, in which random branches are generated, starting in an initial point to a final point (Figure 2.6) [34].

Initially, the method begins to initialize the tree, knowing beforehand: the environment, the initial point, the final point and the number of samples/ maximum iterations for the construction [23, 35].

As said before, the branches construction is performed randomly. The algorithm generates samples until the maximum number is reached, or until the path in the free space is found, between the initial

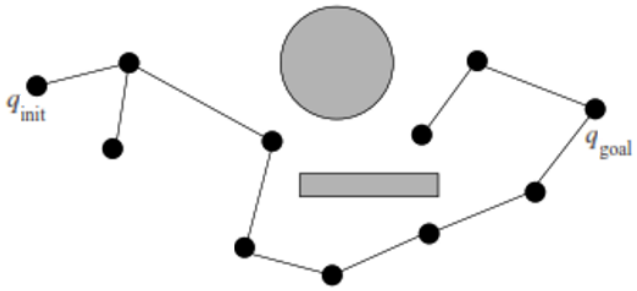


Figure 2.6: Example of a planned path through RRT algorithm. The planned path has as an initial point q_{init} and a q_{goal} as final point. Gray areas correspond to existing obstacles in the workspace, black circles represent some of the randomly created points, and lines to branches. Adapted from [23].

point and the final point. However, in some situations the algorithm can not find a solution; then, the only way to solve this specific problem is to rerun the algorithm with a greater number of samples.

2.3.2 Probabilistic Roadmap - PRM

According to Kavraki *et al.* [36], the Probabilistic Roadmap (PRM) is divided in two phases: the learning phase and the query phase. The first phase consists of building a probabilistic roadmap, inside the configuration space free of obstacles (C_{free}) (represented in Figure 2.7a).

In the search phase, the algorithm PRM adds an initial and final configuration, q_{init} and q_{goal}, respectively, to the roadmap and connects the closer nodes of each configuration. Then, it is determined the shortest path between the configurations, through a path-finding algorithm (Figure 2.7b) [23].

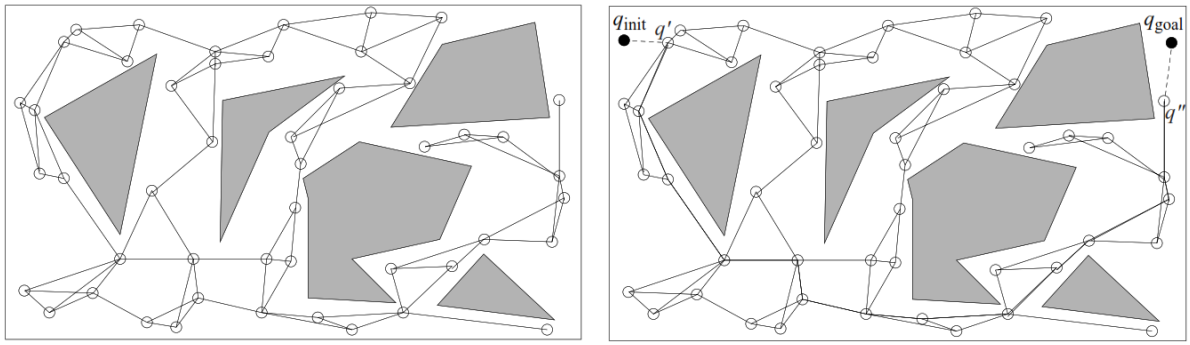


Figure 2.7: Phases of algorithm PRM: (a) learning phase, where a roadmap is built; (b) search phase, where the configurations, q_{init} and q_{goal}, are connected to the roadmap. Thereafter, it is calculated the shortest path between the configurations using a pathfinding algorithm (for example: Dijkstra, A* and D*). The grey areas represent the obstacles, the circles and the segments correspond, respectively, to the nodes and the vertices of the roadmap. Reprinted from [23].

2.4 Pathfinding Algorithms

Some of the previous algorithms, such as the roadmap, the cell decomposition and the PRM methods, build a structure to represent all the possible configurations (inside C_{free}), to search and construct for a possible path. The representation shape is not unique, it can be presented as a roadmap, a graph, a diagram or a grid. Nevertheless, in terms of content meaning is the same, all of them represent the free space through nodes. The intended solution is a set of nodes that connect the initial to the goal configuration. Therefore, for searching the best solution, the pathfinding algorithms are applied.

These methods have several techniques to solve the problem, ones are oriented to performance, others are directed to an oriented search, for example. The majority of the algorithms have a characteristic that saves time during the searching phase, that is, save the travelled nodes and mark them as *visited*. This feature reduce the computational time, because the algorithm can remember where it travelled. Regarding to an oriented search, a couple of algorithms use cost functions that estimates the travel cost. There are two cost functions, the one that calculates the travel cost between the start to the current node, and the one that estimates the cost of the current node towards the goal node. The last function is know as heuristic cost, and it is detailed in Section 2.4.2.

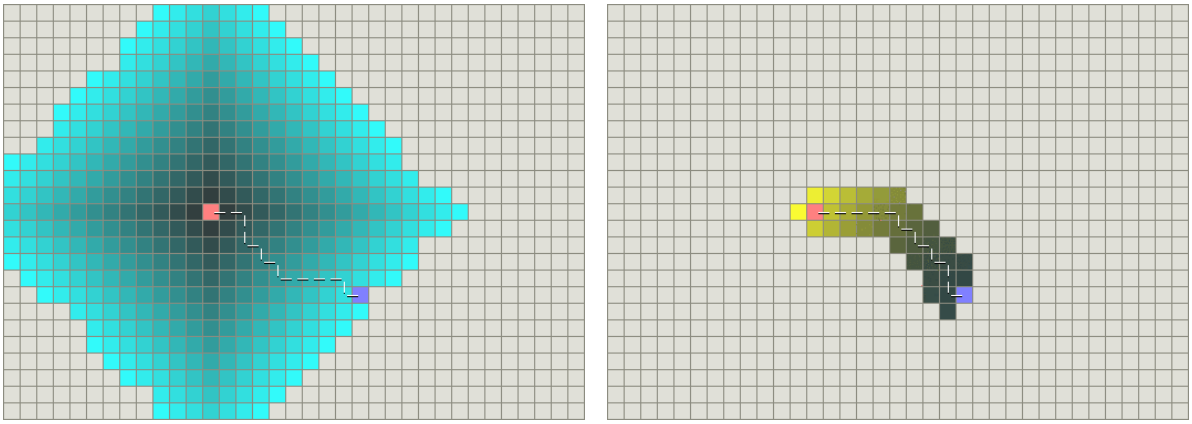
Two well-known pathfinding algorithms are the Dijkstra and the A* (A Star), which is addressed in Section 2.4.1 and 2.4.2. These methods were developed based on older ones, such as the Greedy Best-First (GBFS). The GBFS [37] is an algorithm where the search is only oriented to goal position (Figure 2.8b). It estimates the cost between the current to the goal node through an heuristic function. The node to be expanded is the one that has the lowest cost. The nodes will expand until the goal node is reached. In case of not finding the goal node, the algorithm goes back until it reaches an unvisited node.

2.4.1 Dijkstra Algorithm

Dijikstra introduced his algorithm in [38], where one of his goals was to compute the shortest path between two nodes. This method remains the choice for many planing problems, when the heuristics are not known or not possible.

The algorithm's flow is based on expanding a frontier of equal distance from a source node, expanding outwards from it, step by step, until all nodes are processed or until a goal node is reached. This method is based on a cost function, which represents the distance from the source node to the adjacent one. While the algorithm searches for a solution, it always selects the lowest cost node of the ones that are closer to source node, until the goal node is reached.

Figure 2.8 shows the performance of Dijkstra and Greedy Best-First algorithms. The Dijkstra algorithm presented in Figure 2.8a shows much more expanded nodes than the Greedy Best-First, but it guarantees the shortest path. The Greedy Best-First algorithm illustrated in Figure 2.8b presents a lot less expanded nodes, which proves that the search is oriented to the goal, unlike the Dijkstra, where the search is performed evenly. Taking into account the number of expanded nodes of both algorithms, it concludes that the Greedy Best-First is faster than Dijkstra algorithm. The searching time is directly related to the number of explored nodes; the greater they are, the longer the time.



(a) (b)

Figure 2.8: Comparison between (a) Dijkstra (b) Greedy Best-First algorithms, shows that Greedy Best-First can find paths very quickly compared to Dijkstra algorithm. In (a) the lightest teal areas are those farthest from the starting poind, and thus form the "frontier" of exploration. In (b) the yellow represents those nodes with a high heuristic value (high cost to get to the goal) and black represents nodes with a low heuristic value (low cost to get to the goal). The pink square represents the starting point and the blue square the goal [1].

2.4.2 A* Algorithm

About ten years later the Dijkstra's algorithm [38], a great contribution was published by Hart *et al.* in [32]. It is the A* algorithm which was used for many applications in that time, and is still nowadays one of the most used algorithms.

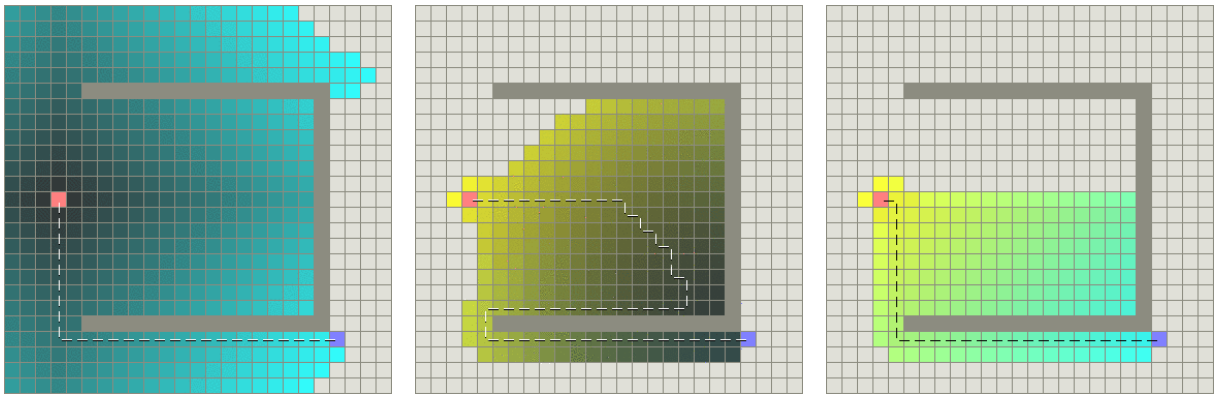
This method is used for pathfinding, mostly because of its flexibility and performance. This algorithm finds a least cost path from a start node to a goal node in a graph, guided by a heuristic evaluation function, $f(n) = g(n) + h(n)$; where $g(n)$ is the cost of the best path found from the start node to node n, and $h(n)$ estimates the cost of the best path from node n to the goal.

In other words, this method combines the best features of the Dijkstra and Greedy Best-First algorithms. It can behave like Dijkstra or Greedy Best-First, when $h(n) = 0$ or $g(n) = 0$, respectively.

Figure 2.9 illustrates the performance of the Dijkstra, Greedy Best-First and A* algorithms with a concave obstacle. As predicted, the Dijkstra in Figure 2.9a gets the shortest path, but it searches too many nodes. Figure 2.9b represents the Greedy Best-First algorithm where it shows that the focus is only the goal node, not the shortest path but to arrive as soon as possible to the goal position. Last one,

[1] Images from https://www.redblobgames.com/.

Figure 2.9c shows the A* performance, as you can see, the path is very similar to Dijkstra and the number of expanded nodes are a lot less than Dijkstra and less than Greedy Best-First, which is a very good solution for this problem.



(a) **(b)** **(c)**

Figure 2.9: Comparison between (a) Dijkstra, (b) Greedy Best-First and (c) A* algorithms. The pink square represents the starting point and the blue square the goal [1].

A great performance of A* method rely on a well-defined heuristic function. The most known heuristic cost functions, $h(n)$, which are used to estimate the distance between the source node and the goal node, are the Manhattan distance, the diagonal distance or the Euclidean distance.

The Manhattan distance is the simplest way to compute de distance between the current node and the goal node. This function only allow four directions (horizontal and vertical movements), according to the next equation:

$$h(n) = \Delta x + \Delta y \tag{2.1}$$

The diagonal distance, also known as Chebyshev distance, can be used when it is possible to move in eight directions, which includes diagonal movements:

$$h(n) = max(\Delta x, \Delta y) \tag{2.2}$$

The last function is the euclidean distance, this one permits movements in any direction. It is a segment between the actual node and the goal node:

$$h(n) = \sqrt{(\Delta x)^2 + (\Delta y)^2} \tag{2.3}$$

[1] Images from https://www.redblobgames.com/.

2.5 Towards Smooth and Bounded Curvature Algorithms

Path Planning for a nonholonomic robot is a difficult problem, since the planner must check nonholonomic constraints and obstacle avoidance simultaneously. This sections details some of the approaches for path planning of nonholonomic robots, in particular, those capable of constraining the curvature, i.e. the steering angle and giving continuous and smooth paths. The term nonholonomic comes from mechanics and refers to differential constraints that cannot be fully integrated to remove time derivatives of the state variables [24].

The following sections will explain some of the approaches most used, in order to adapt for the path planning constraints of the case of study of the project.

2.5.1 Dubins Curves

The first results on motion curvature constraints were presented by Dubins and shown in [39]. This method shows that between any two configurations, the shortest path for a car-like robots, can always be expressed as a sequence of straight line segments and circular arcs [24]. This method is one of the simplest methods for solving path planning for a car-like robot made use of a sequence of Dubins curves. The Dubins paths do not allow backward movements, it always moves forward. The task is to find the shortest path between two configurations (position and orientation), according to the curvature constraints of the robot, namely the minimum radius of curvature.

The Dubins path is expressed as a combination of no more than three motion primitives, which have an associated symbol for each primitive S, L and R. The S primitive drives the car straight ahead. The L and R primitives turn as sharply as possible to the left and right, respectively. The Dubins path is no more than a combination of these symbols, which sequence corresponds to the order in which the three primitives are applied. This sequence is called by *word*. There are six words that Dubins considered as a optimal possibly [24]:

$$\{LRL; RLR; LSL; LSR; RSL; RSR\} \tag{2.4}$$

The shortest path between any two configurations can always be characterized by one of these words. These words are called the *Dubins curves*. To be more precise, the duration of each primitive should also

be specified. Using such subscripts, the Dubins curves can be more precisely characterized as [24]:

$$\{L_\alpha R_\beta L_\gamma, R_\alpha L_\beta R_\gamma, L_\alpha S_d L_\gamma, L_\alpha S_d R_\gamma, R_\alpha S_d L_\gamma, R_\alpha S_d R_\gamma\} \tag{2.5}$$

in which $\alpha, \gamma \in [0, 2\pi]$, $\beta \in [\pi, 2\pi]$, and $d \geq 0$. Figure 2.10 illustrates two cases for different words [24].

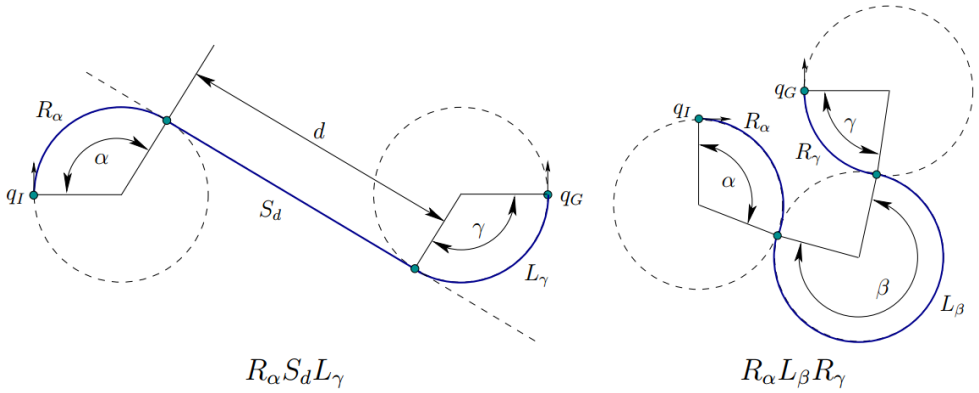


Figure 2.10: Dubins Curves example. Reprinted from [24].

Several researchers have developed solutions based on Dubins curves, such as Guang Yang and Kapila [40] and Anderson *et al.* [41], who built their solution upon Dubins' ideas to generate feasible trajectories for unmanned air vehicles (UAVs). Parlangeli and Indiveri [42] present an approach based on the Dubins shortest paths, to address the issue of planning smooth 2D paths with bounded curvature and curvature derivative connecting two assigned poses (positions and orientations). The developed solution allows to link two fixed (initial and final) poses, while passing through the position of each given intermediate waypoint.

2.5.2 RRT with Post Path Smoothing Algorithm

Yang and Sukkarieh [43] proposed an efficient and analytical continuous-curvature path-smoothing algorithm, that fits an ordered sequence of waypoints generated by an obstacle-avoidance path planner. The algorithm generates a cubic Bezier curve that satisfies both the curvature continuity and maximum curvature constraints. The authors mentioned above used this type of approach to generate smooth paths of UAVs in 3D environment [44, 45]. The approach only requires the value of the maximum curvature and

implements two algorithms, G1 and G2 Continuous Cubic Bezier Spiral (G1CBS and G2CBS, respectively) for path smoothing. The difference between the two algorithms is that the curvature of the G1CBS path is discontinuous at the junction points, while the curvature of the G2CBS path is continuous over the whole path.

The approach first make use of the basic RRT planner [46] to find the path between the initial and final position. Then, a path pruning algorithm is applied to remove most extraneous nodes from the path, simplifying the path with only the strictly necessary nodes. Finally, a G1CBS or G2CBS path smoothing is applied, which makes use of cubic Bezier curve [47], an iterative tuning to get bounded continuous curvature path. Figure 2.11 illustrates each performed phase and the smooth path obtained through the G2CBS path-smoothing algorithm. Although these guarantee the maximum curvature constraint over the whole path, they present a relatively high computational complexity due to the path pruning algorithm and the path smoothing algorithms.

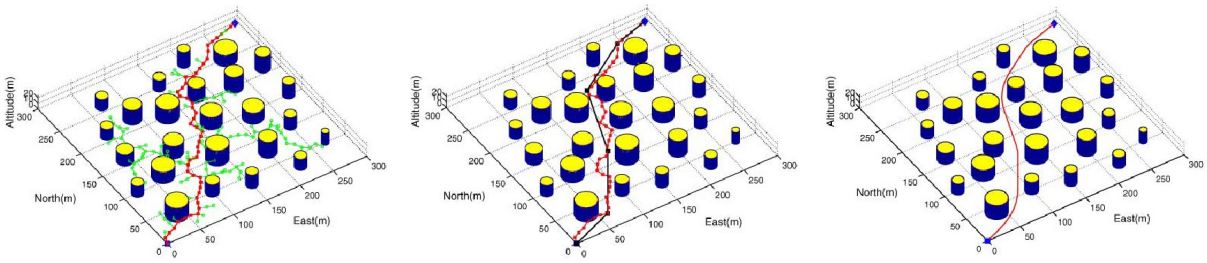


Figure 2.11: Path planning using RRT algorithm and path smoothing using G2CBS algorithm. (a) RRT path planner generates a 3-D collision-free path among obstacles; (b) Most extraneous nodes are eliminated by the path-pruning algorithm; and (c) G2CBS path-smoothing algorithm generates a smooth path satisfying the maximum curvature constraint. Reprinted from [43].

2.6 Discussion

According to Hwang and Ahuja in [26], there are two possible classifications for motion planning, depending on the mode in which the obstacle information is available, which could be static or dynamic. In a static problem, all the information about the obstacles are know *a priori*, and the motion of the robot is designed from the given information. On the other hand, in a dynamic planning only partial informations

is available about the obstacles. The robot plans the path based on all the on the available information, which is collected/updated as the robot follows the path.

The environmental model of the problem under consideration is classified as static, because all the information about obstacles is known in advance by the PCB mask. Therefore, an important step is the definition of the configuration space. There are two methods described in this chapter that are tailored to interpret the environment model and to define the free and obstacles configuration space, which are the roadmap and the cell decomposition. One important aspect to take in mind is that the obstacles could have various and different shapes. Between the two algorithms, the cell decomposition method is the most suitable, because it is independent of the shape of the obstacles, unlike the roadmap methods. The approximate cell decomposition with a regular cell size is the simplest method to implement and much faster to compute than the roadmap. The main disadvantage is that the representation of obstacles is usually less precise than in other methods. According to the advantages mentioned above, the approximate cell decomposition method was selected to represent the environment in all performed tests.

There are several search algorithms to generate paths from this or other representations of the environment. The Dijkstra finds the shortest path between two configurations, however it consumes more processing time than other methods, since it is not oriented to the final cell. The A* algorithm uses heuristics costs to estimate the distance to the final cell, it is one of the most used algorithms due to its simplicity and efficiency. There are various approaches based on A* algorithm that aim to smooth the path, obtaining more linear paths and/ or limited angular variations. The Theta* [48] is one of the examples derived from A*, which is based on a new function called Line of Sight (LOS). This function creates straight lines between two cells if it does not pass through a blocked cell, resulting on a smoother path. Other approach introduced by Yakovlev *et al.* [49] is called LIAN, a combination of the provided features of A* and Theta* algorithms, and the addition of some restrictions, namely, restricting the search angle of the successors. These derivations of A* algorithm do not integrate the curvature constraints associated to the problem of the project. However, some approaches satisfies these constraints through post processing methods, which integrate them taking into account the generated path by the search algorithms.

Unlike the pathfinding algorithms, potential fields and sample based methods usually use the original map as a representation of the environment. However, in some cases, these two methods are used from the Voronoi diagrams. The main disadvantage of the potential fields is the possibility of having local

minimum where holds the robot position. The probabilistic methods can produce different solutions from the same initial conditions. These methods allow quick results to be obtained in complex situations and are widely used in situations where the solutions are almost required in real time. The main drawback is that the presented solutions are usually not optimal because of the randomness that exists during planning.

Section 2.5 presented two approaches which satisfy the curvature constraint of the path planning problem. The first is based on the Dubins curves and the second one is based on the Bezier curves. As mentioned, Bezier curves present a huge computing cost compared to the Dubins curves. However, the various phases presented in the second approach, namely, the search, prun and smooth revealed to be a good approach, taking into account that the path planner has to avoid collisions with the obstacles and at the same time guarantee the maximum radius of curvature.

Taking into account all the methods and algorithms described in this chapter, the approach that will be used to plan a smooth path that satisfies the maximum radius of curvature will be based on three distinct phases. In the first phase, a search algorithm will be implemented to find a valid path between the sequence of waypoints. The most suitable search algorithm will be evaluated by its performance in the environment model. The second phase will implement an algorithm which will simplify whenever possible the generated path. Finally, the last phase will accommodate the curvature constraints through the development of a method based on the Dubins curves.

Chapter 3

System Overview

This chapter presents a high-level overview of the whole project in order to visualize and to understand the role and function of each component of it. These components can be seen in the diagram depicted in Figure 3.1 along with their work flow. The PCBs masks and the user configurations are the first components, they represent the inputs for the path planning problem. Then, there is the Path Planner module, which is responsible for generating the smooth path according to the PCB and the configuration set. After that, the Motion Planner creates a trajectory to follow the smooth path. At the end, it is represented the Sawyer robot and the RViz, which corresponds to the possible scenarios where the trajectory can be performed, in a real or in a virtual scenario, respectively.

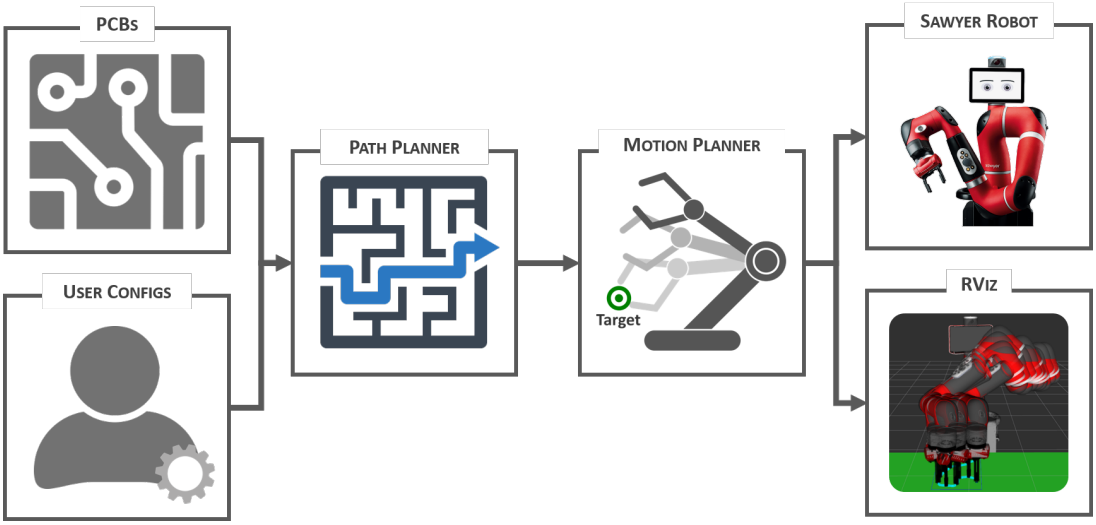


Figure 3.1: System overview.

In the following sections, each module presented in the diagram of Figure 3.1 will be briefly described, as well as the tools and the frameworks used during the development of the project.

3.1 Path Planner Module

The Path Planner module present in Figure 3.1 is the one responsible for building a smooth and bounded curvature path in a Cartesian plan of two dimensions (2D). This block receives and processes the input data, such as the PCB mask and the configurations specified by the user. Then, it should output a smooth path with all the specified constraints fulfilled. When it has a valid solution, the module sends it to the next one – the Motion Planner module.

A control application was developed, that can be seen in Figure A.1, and all of its features are described in Appendix A.1. The main goal of this graphical interface is to provide a simple and friendly environment for the user to set up configurations, and to change the parameters. It provided a fastest way to run and save several tests, that were used to compare the developed algorithm.

3.2 Motion Planner Module

The Motion Planner module presented in the system overview illustrated in Figure 3.1, is responsible for generating the motion for the Sawyer robot. This module generates the trajectory for the Sawyer robot, according to the desired path created by the Path Planner.

The Motion Planner is a ROS based module, which main purpose is to be able to communicate with the real robot. It uses some of the features provided by the MoveIt in order to plan and execute the trajectories, that are accessible through ROS. Additionally, the virtual scenario is built through a ROS tool called RViz (ROS Visualization), where the trajectories are visualized and validated before sent them to the real robot.

Similarly to the Path Planner module, the Motion Planner is integrated as well in a graphical interface, to deal with the communication between the frameworks (MoveIt and RViz), the Sawyer robot and the Path Planner module. In Figure A.3 it is illustrated the developed application for this module, and its

functionalities are listed in Appendix A.2. The following sections details the main features provided by ROS, MoveIt and RViz.

3.2.1 ROS, MoveIt! and RViz

The keywords ROS, MoveIt and RViz introduced in the previous section, will be briefly described to provide a background knowledge about each one.

ROS

ROS[1] stands for Robot Operating System and it is an open-source meta-operating system for robots. ROS provides a set of functionalities identical to those provided by an operating system, such as: (i) hardware abstraction, (ii) low-level device control, and (iii) message passing interface between inter-process communication. Additionally, ROS provides a collection of tools, algorithms and libraries, which are organized in packages and they aim to simplify the task of developing complex and robust software for a variety of robots. The majority of these packages are included in the ROS distributions, while others implemented by developers or by the robot manufacturers, providing and distributing the packages in repositories.

The ROS communication is based on four fundamental concepts: nodes, messages, topics and services [50, 51]. A ROS system is composed of a large number of independent programs running simultaneously – nodes – that are communicating with one another by exchanging messages. In order to ensure the communication between the nodes of the system, the ROS master, often called roscore, provides the connection information to nodes, so that they can transmit messages to one another.

The most common way for ROS nodes to communicate is to send messages through topics. A topic is a name for a stream of messages with a defined type. Nodes can simultaneously subscribe and publish in topics. The idea is a node that wants to share information - publisher - publish messages on one or more topics. A node that wishes to receive information - subscriber - must subscribe to the topics of particular interest. The ROS master is responsible for ensuring that publishers and subscribers know each other and interconnect. Topics are responsible for transmitting messages sent by publishers to all subscribers associated with them.

[1] For more information visit http://www.ros.org.

Another way to communicate between nodes is through services, this way is well suited for occasional data transmission and that takes a limited time to complete. Each node can provide or use one or more services, a ROS service is defined by its name and a pair of strictly typed messages: one for the request and one for the response. Whenever a node wants to use a service, it sends a data request – request – to the server node and waits for a response. After receiving a request, the server node performs some task (such as calculating something, configuring hardware or software, among others) and sends the requested data to the client node – response.

MoveIt!

MoveIt[2] is a motion planning library that offers various motion planning algorithms that have been used on a wide variety of robots. MoveIt runs on top of ROS and can be used with any ROS-supported robot.

Figure 3.2 shows the high-level system architecture of the primary node provided by MoveIt called "move_group". This node provides a set of ROS actions and services for users to use. There are three possible ways that the users can access the actions and services provided by the node "move_group": C++ (move_group_interface), Python (moveit_commander) and GUI (RViz Plugin). The approach used in the developed solution was the move_group_interface package, using the C++ language to interface the move_group node. In Section 5.1.1 are enumerated and explained all the move_group services used in the developed project.

RViz

RViz[4] stands for ROS Visualization, it is a 3D visualizer for displaying sensor data and state information from ROS. Using RViz, it is possible to visualize the robot's current configuration on a virtual model. It can also display live representations of sensor values coming over ROS topics including camera data, infra-red distance measurements, sonar data, and more.

This application allows the visualization of all the necessary information (such as the robotic model, the planned trajectory, among others) in order to inspect and analyse the developed solution, allowing to

[2]For more information visit https://moveit.ros.org/.

[3]Image from https://moveit.ros.org/documentation/concepts/.

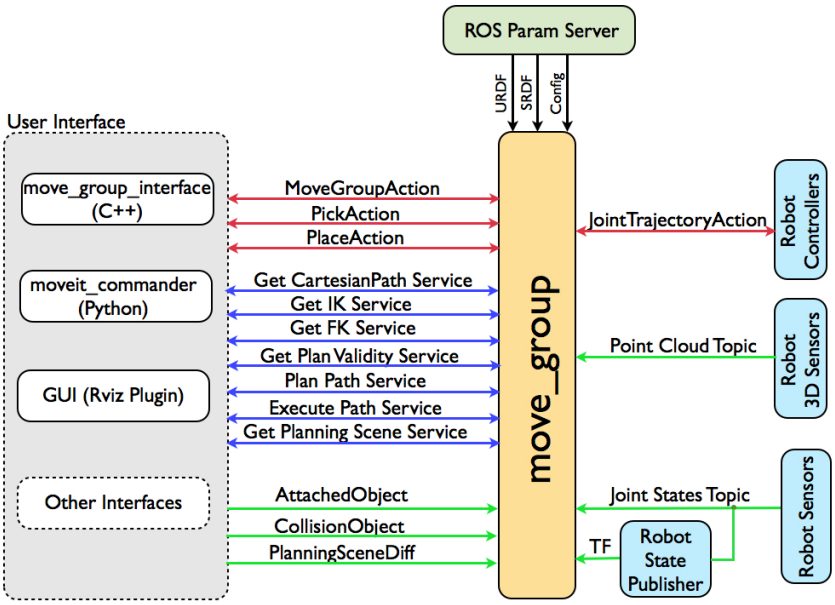


Figure 3.2: System architecture of the move_group node provided by MoveIt![3].

identify problems that may arise. The RViz is used to validate the solution in virtual environment and only after the validation, it is performed in the real robot.

3.3 Sawyer Robot

The robotic manipulator used during the implementation and validation of the dissertation project was the Sawyer robot. Sawyer is the latest collaborative robot produced by the company Rethink Robotics and it was unveiled on March 2015. The robotic system is composed by a mobile pedestal (Figure 3.3b) and by a collaborative robotic arm designed to provide industries high performance automation, essential for performing a wide range of tasks that require high flexibility and in shared workspaces with humans (Figure 3.3a).

The Sawyer's robotic arm is composed by seven rotational joints, whose disposition on the mechanical structure confers a total of seven degrees of freedom. The maximum payload that it can handle is 4 kg; it can reach a maximum of 1260 mm, and perform the tasks with a repeatability of $\pm$ 0.1 mm. In Table 3.1, it is summarized the main specifications of the Sawyer robot.

[4]For more information visit http://www.ros.org/core-components/.

(a) **(b)**

Figure 3.3: Sawyer robotic system: (a) Sawyer's robotic arm[5] and (b) its mobile pedestal[6].

Table 3.1: Basic specifications of robot Sawyer[7].

Max Reach	1260 mm
Task Repeatability	$\pm$ 0.1 mm
Degrees of Freedom	7
Operating Temperature	5°C-40°C, 80% Relative Humidity
Robot Weight	19 kg
Payload	4 kg
Power Requirements	100-240 VAC, 47-63 Hz, 4A Max
Communication	Modbus TCP, TCP/IP
Pedestal Dimensions	69x76x92 cm

Sawyer offers a set of features that distinguishes it from the majority of the industrial robots, highlighting its ability to immobilize in contact with obstacles and humans. This feature is possible due the high-resolution force sensors embedded in the actuators of each joint. The integration of these high-resolution sensors allow the robot to have a controlled motion, which can be managed by the torque or by the angular position of the joints.

Figure 3.4 presents the name of each link of the robotic manipulator, and the values of each link are presented in the Table 3.2. Note that the link L_0 is shown in Figure 3.5b.

[5]Figure from https://www.active8robots.com/robots/rethink-robotics/.
[6]Figure adapted from http://mfg.rethinkrobotics.com/intera/Sawyer_Hardware.
[7]Table adapted from http://www.rethinkrobotics.com/sawyer/tech-specs/.
[8]Adapted from http://mfg.rethinkrobotics.com/wiki/Robot_Hardware.

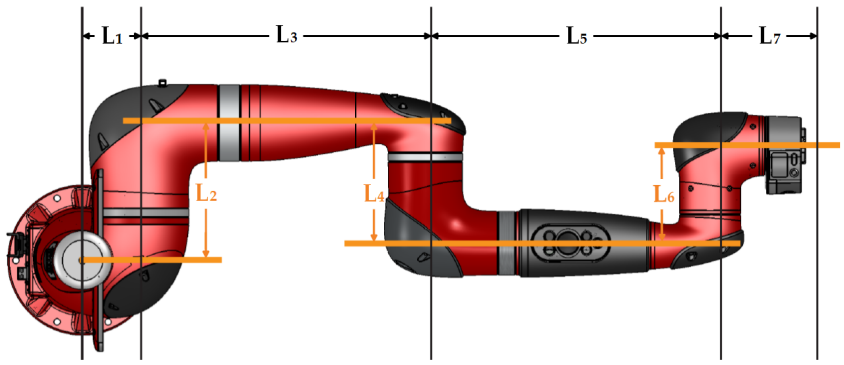


Figure 3.4: Names of Sawyer links[8].

Table 3.2: Lengths of Sawyer links[8].

Link	L_0	L_1	L_2	L_3	L_4	L_5	L_6	L_7
Length (mm)	317	81	192.5	400	168.5	400	136.3	133.75

Figure 3.5 gives an overview of Sawyer's rotational joints position. This pose have all the joints with the value zero. The angular range that each joint can execute are presented in Table 3.3.

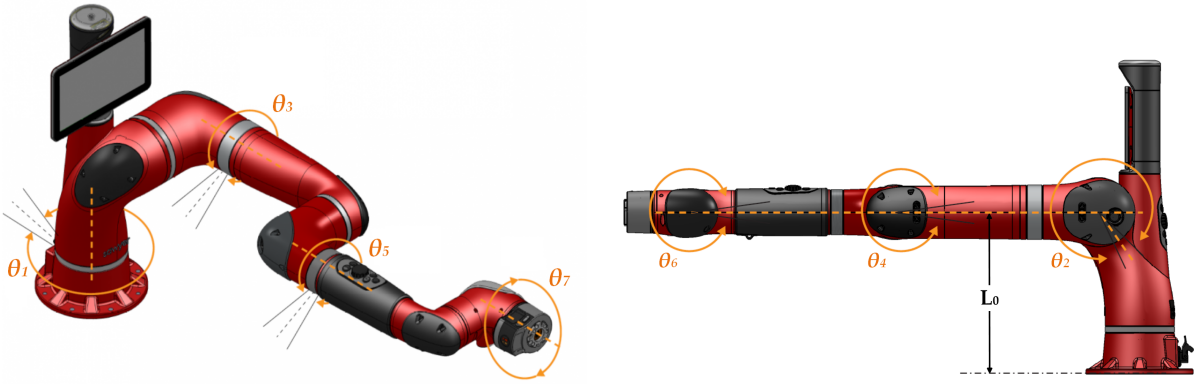


Figure 3.5: Disposition of the joints in the mechanical structure of the Sawyer's robotic arm: (a) roll joints and (b) pitch joints[8].

According to Rethink Robotics, Sawyer is suitable for various industrial applications, such as handling and manipulating objects, test and quality inspection, co-packing and end-of-line packing, and others[9].

[9]For more information about the Sawyer's applications https://www.rethinkrobotics.com/applications/.

Table 3.3: Range of each joint of robot Sawyer[8].

i	1	2	3	4	5	6	7
$\theta_{i\ max}(^\circ)$	174.75	130.75	174.95	174.95	171	171	269.9
$\theta_{i\ min}(^\circ)$	-174.75	-218.75	-174.75	-174.75	-171	-171	-269.9
$Range\ (^\circ)$	349.5	349.5	349.5	349.5	342	342	539.8

Chapter 4

Path Planning

This chapter details the implementation of all the adopted methods and strategies of the Path Planner Module. The problem of this path planning is subject to various constraints, which are listed in Section 1.2. In the following sections, some concepts are introduced about path searching in square grids, followed by the environment representation. Then, it is highlighted how the waypoints and the straight linear segments are configured and placed. Thereafter, it is described each phase of the developed path planning algorithm. The last section corresponds to the characterization of the path planning algorithm, where several tests were done to evaluate its performance and its behaviour in different situations.

An overview of this module is presented in Figure 4.1. There are two main blocks, the Inputs and the Path Planner. The first corresponds to the input data that the user should specify, such as (i) the PCB mask, (ii) the Safety Distance (SD) from the obstacles, (iii) the position of each waypoint in the PCB mask, (iv) the length of the straight linear segment in each waypoint, (v) the orientation of each segment and (vi) the maximum, preferential and minimum turning radius. The second block represents the sequence of algorithms and methods implemented in the Path Planner and their relation with input data. In the first place, the cell decomposition algorithm receives the PCB mask and the safety distance from the user and the size of the cells to construct a representation of the environment model. Then, the Map Weighting method loads the parameters Penalty Radius (PR) and Penalty Weight (PW) for weighting the cells near obstacles. After that, the Search Phase, which is the first phase to find a path according to the input data, namely the coordinates of the waypoints, the size of the straight linear segment and their orientation. In this phase, four algorithms were implemented , the A*, the Theta*, the A*3D and the Theta* LA (Limited

Angle), but in the final solution only the A* algorithm was used. The A* receives as input parameters the radius of the possible successors Δ and the heuristic costs functions and their respective weights (wG and wH). Next, the Prun Phase is responsible for the elimination of unnecessary nodes in the previous generated path. Finally, the Smooth Phase applies the Dubins Curves to smooth the prun path. In this case, it requires the specification of the radius of curvature (ρ_{max}, ρ_{pref} and ρ_{min}).

The following sections discuss the behaviour and influence of each parameter and algorithm, since the map representation to the smooth phase.

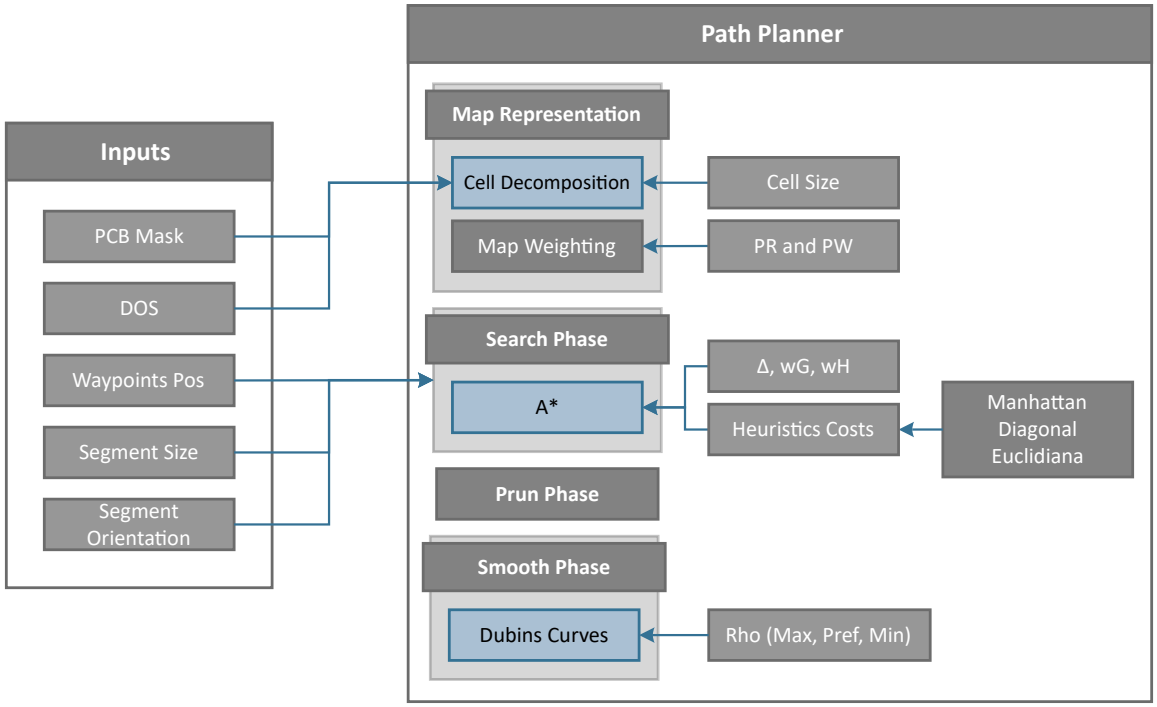


Figure 4.1: Overview of the Path Planner Module.

4.1 Grid Searching Concepts

This section addresses some concepts related to grid searching, that were used and implemented in the developed algorithm. The first concept refers to the square grid notation, where there are two square grid notations widespread nowadays, centre-based and corner-based. The centre-based representation assigns a node at the centre of a cell, while the corner-based notation assigns nodes at the four corners

of a cell. These notations are represented in Figure 4.2. In this project it was adopted the centre-based notation (Figure 4.2a).

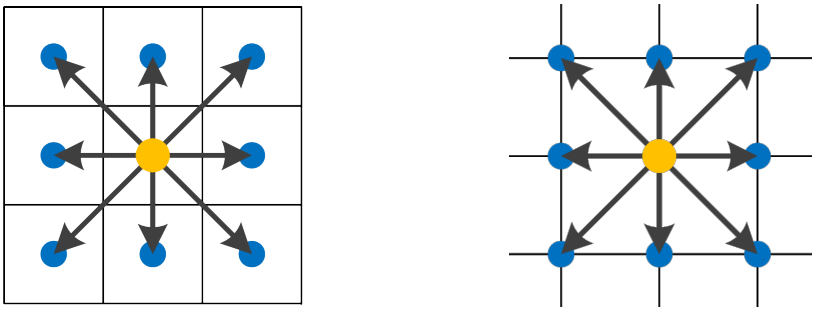


Figure 4.2: Square grids notation: (a) centre-based and (b) corner-based.

Figure 4.3 illustrates two algorithms, Midpoint and Bresenham algorithm. The Midpoint algorithm is a method well-known in computer graphics, which is responsible for drawing "discrete circumferences" [52]. This method is used to select the radius of the neighbouring cells. In the example, Figure 4.3a, the radius was set to 4 ($\Delta = 4$) and the neighbouring cells of the blue node are the yellow ones.

This approach has been used by researches, such as Yakovlev *et al.* in [49], as one of the strategies for path planning problem with angle constrained. Its main advantage regarding the angle constrained problem is the fact that how greater the radius (Δ) is, greater the angle resolution between the possible neighbours will be, which gives more flexibility during the searching. For example, when the radius is 4, the resolution between the neighbours is 15° (Figure 4.3a), while in Figure 4.2a the resolution is only 45°.

The other method is the Bresenham algorithm, introduced by Bresenham [52]. This method is used to detect the presence or absence of Line-Of-Sight (LOS) between two cells. The algorithm draws a "discrete line section" (Figure 4.3b) and if it contains only traversable cells/ free cells, then the algorithm returns true, otherwise it returns false. This method is used by several algorithms, such as Theta* and other algorithms for post processing.

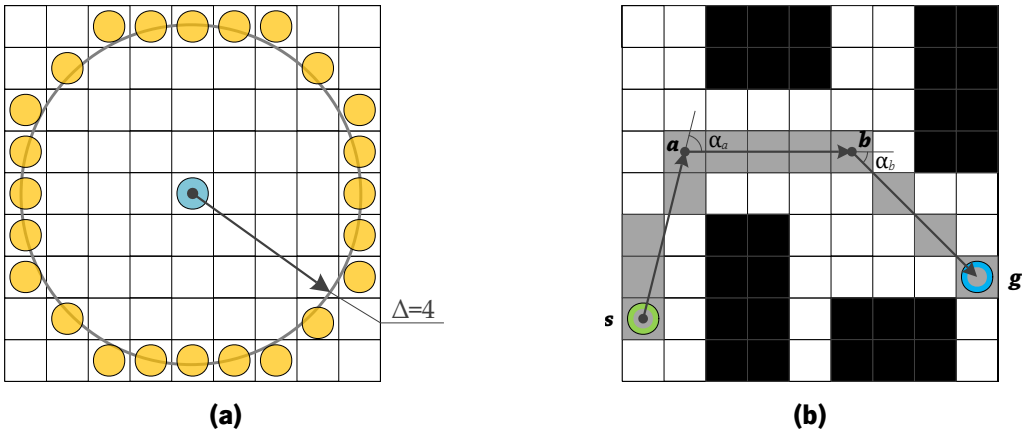


Figure 4.3: Midpoint and Bresenham algorithms: in (a) Midpoint algorithm and (b) Bresenham algorithm. Adapted from [49].

4.2 Map Representation

The representation of the environment model is based on the cells decomposition algorithm discussed in Section 2.2.1. The environment model, in this project, is a PCB mask selected by the user, which is divided into several cells of the same size (grid of square cells). The size of the cells may vary depending on the type of PCB mask and according to the PCB dimensions or the intended results.

In order to categorize each cell, obtained from the map decomposition, it was built a binary matrix (illustrated in Figure 4.4). Each square corresponds to a cell, which can be classified as a blocked cell or as a free cell. When an obstacle intersects a cell, the cell value is set to "1", otherwise its value is set to "0", which means that it is free of obstacles. This binary matrix is used by the algorithm during the search phase to find a couple of non blocked cells, between the start and goal position.

Additionally, there is a parameter called safety distance (SD) that represents the minimum distance that the path should have from obstacles along the path. As the generated path is punctual and has no dimension, in order to avoid collisions with the obstacles, this approach was implemented. This distance ensures that regardless the size of the material that should be placed along the path or the dimensions of the robot that should follow the path, depending on the problem in hands, it never collide with any obstacle. Moreover, it is possible to define a minimum distance that the final path has to maintain from the obstacles along the path. Thus, SD is represented by the size of the material/robot plus the minimum distance required from the obstacles. This method increases the obstacles size, as shown in Figure 4.5.

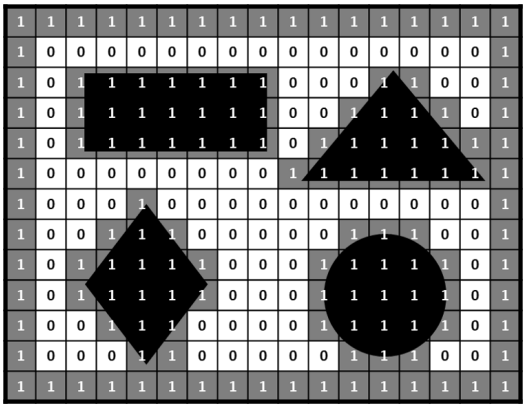


Figure 4.4: Environment representation through a binary matrix. The cells that intersect the obstacles are represented by the value "1", whereas the remaining cells are represented by the value "0". The black colour corresponds to the obstacles present in the environment, and the grey and white cells stand for blocked and free cells, respectively.

Herein, it is represented a PCB mask without a safety distance and another one with a safety distance, and it is possible to observe the growth of the obstacles size, represented by the grey colour.

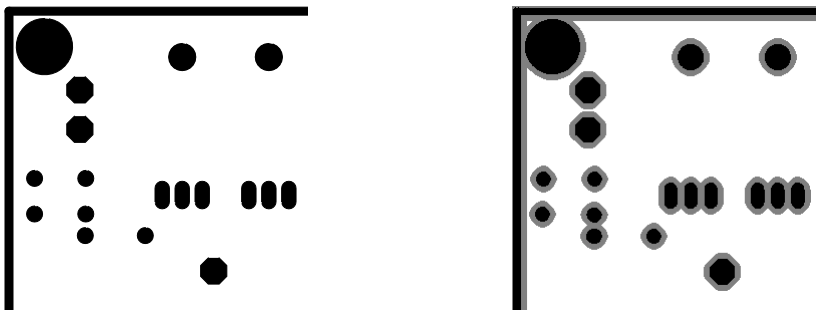


Figure 4.5: Safety distance from obstacles; (a) without safety distance and (b) with 1 mm of safety distance.

Moreover, it was implemented a method called Map Weighting, which is responsible for penalizing the cells that reside near to the obstacles; therefore, it makes these cells less attractive to be chosen. The main goal of this method is to improve the performance of the path planner and to maximize the distance between the path and the obstacles.

The implementation of this method is based on two parameters the Penalty Radius (PR) and the Penalty Weight (PW); the first corresponds to the radius of the discrete circumference centred in the blocked cells (by the Midpoint algorithm introduced in Section 4.1) and the second is the maximum penalty weight that

a cell can have. Figure 4.6 illustrates the weighted cells by the gradation of the green colour, the darker cells represent the highest weights, while the lighter ones have a smaller penalty weight.

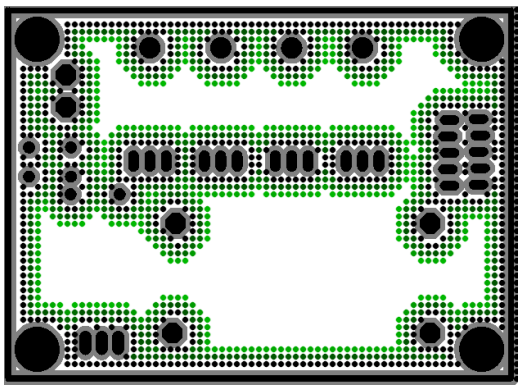


Figure 4.6: Weighted map with a penalty radius of 3 cells.

4.3 Waypoints and Straight Linear Segments

The path planning problem of this project is defined by a final destination and intermediate waypoints, each one is defined by their Cartesian coordinates $W = [x, y]$. Only the waypoints placed in the free space are valid, others ones are discarded.

All the valid waypoints have a straight linear segment with its dimension and orientation is specified by the user. For each waypoint, all the possible orientations are checked, and only the acceptable ones are allowed for selection. The segments that collide or are very close to the obstacles are discarded. Figure 4.7 portrays two examples; Figure 4.7a represents all the possible orientations for the waypoint segment, while Figure 4.7b shows a situation where there are only two acceptable orientations, because the others ones had intersected with blocked cells; thus, they were eliminated.

The angular resolution between the segments varies depending on the size of the cells; thus, the resolution increases when the size of the cells decrease.

Figure 4.7: Waypoint and segments representation: (a) illustrates all the possible orientations that a segment may have; (b) shows that for this waypoint there are only two valid orientations, because the remaining orientations had intersected the blocked cells and thus, they were discarded. The obstacles correspond to the black colour; the purple colour represents the blocked cells; the grey segments are the possible orientations, and the blue one is the selected.

4.4 Developed Path Planning Algorithm

This section details the development of the path planning algorithm, explaining the implementation details and the accommodation of all the constraints.

The developed path algorithm in this dissertation project is divided in three phases by a specific sequence of phases: (i) searching; (ii) pruning; and (iii) smoothing. The following sections describe each of these phases.

4.4.1 Searching Phase

The first phase of the path planning algorithm is the searching phase. This phase is based on the traditional algorithm A* with some improvements, namely, in the assignment of potential successors and the use of weighted heuristics. The pseudo code of A* modified algorithm (wA*+) is presented in Algorithm 4.1.

Likewise any other A* search algorithm, wA*+ maintains and stores in memory two lists of nodes: *open* and *closed*. The *open* list contains all the potential candidates to be processed, ordered by their priority; whereas, the *closed* list have all the nodes already visited and that have been processed. The priority of a node n, $f(n)$, is calculated as $f(n) = w_g \cdot g(n) + w_h \cdot h(n)$; where w_g is the respective weight factor of $g(n)$, which represents the exact cost of the path from the starting node to node n, and

Algorithm 4.1: Modified A* Algorithm – wA*+

```
1  Search( [start], [goal], Δ )
2      g([start]) := 0;
3      bp([start]) := [start];
4      open := ∅;
5      open.push([start]);
6      closed := ∅;
7      while open ≠ ∅ do
8          [a] := open.Pop();              /* Get the node with the lowest cost f([a]) */
9          if a = goal then
10             constructPath();
11             return "path found";
12         closed.push([a]);
13         SUCC = getCircleSuccessors([a], Δ);
14         if dist(a, goal) < Δ then
15             SUCC.push(goal);
16         foreach [a'] ∈ SUCC do
17             if [a'] is a blocked cell then
18                 continue;
19             if LineOfSight(a, a') = false then
20                 continue;
21             if a' ∉ open ∨ closed then
22                 bp([a']) := [a];
23                 [a'].g := [a].g + g(a, a');
24                 [a'].h := h([a']);
25                 [a'].f := w_g[a'].g + w_h[a'].h;
26                 open.push([a']);
27     return "no path found";
28 end
```

w_h the weight factor of $h(n)$ that corresponds to a heuristic estimated cost from node n to the goal node. The highest priority node is the one with lowest value of $f(n)$.

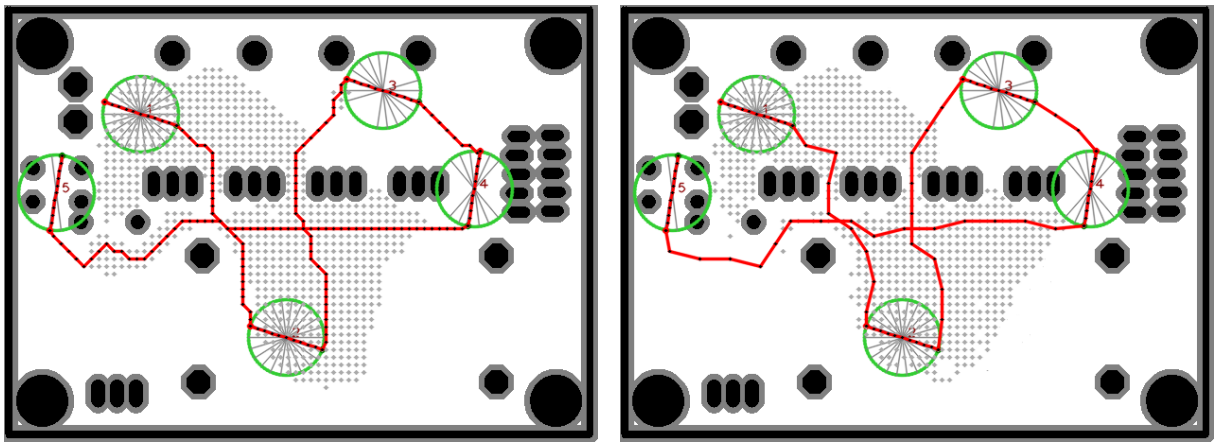
Initially, the algorithm starts with both lists empty and adds the start node ($[start]$) to the *open* list; the first node to be processed. Then, the algorithm will be repeating the procedure: (i) retrieves the highest priority node (the one with the lowest $f(n)$), (ii) checks if the node is the goal node, (iii) assigns the possible successors of the retrieved node, (iv) calculates all the costs for the successors, and (v) adds the retrieved node to the *closed* list and the successors to the *open* list. The algorithm stops when the goal

node is retrieved from *open* list, in this case the path will be constructed; or when the *open* list becomes empty, which means that it is not possible to find the goal node and the algorithm returns failure to found a path.

The possible successors are calculated through the method *getCircleSuccessors*$([a], \Delta)$, which implements the Midpoint algorithm described in Section 4.1. This method calculates all the successors of the node $[a]$, through a discrete circumference of radius Δ. It returns only the possible successors, which are the ones that are not in a blocked cell and that verify a direct line of visibility between the node cell and the successor cell. The possible candidates to add in the *open* list to be processed are those that are neither present in the *open* list nor in the *closed* list.

The exact cost of the transition from the node $[a]$ to the node $[b]$ ($g(a, b)$) is calculated as follows $len(a, b) = dist(a, b)(1 + avgW)$, where the $avgW$ is the average weight of the cells returned by the Bresenham algorithm and the $dist(a, b)$ corresponds to the euclidean distance between them.

Figure 4.8 represents two examples of the generated path by this phase. The results shown that for Δ=4, the number of expanded nodes is lesser than Δ=1; the time spent is almost the same but the resulted path is smother using Δ=4.



(a) **(b)**

Figure 4.8: Searching phase with different Δ values; (a) $\Delta = 1$ and (b) $\Delta = 4$. The resulted path is represented by the red colour, the expanded nodes are the grey circles and the black colour corresponds to the obstacles.

4.4.2 Pruning Phase

The intermediate phase of the path planner is the pruning phase. Its main goal is to simplify the previous path resulted from the searching phase, by eliminating unnecessary nodes. Therefore, it is implemented a method named Pruning Method based on the Line-Of-Sight (LOS), introduced in Section 4.1. The general idea is to incrementally look for a line of visibility between two nodes, n_i and n_{i+n}, and whenever possible, eliminate the intermediate nodes. This method is presented in Algorithm 4.2. The output from this phase is a path based on the previous one, but with fewer nodes, called by pruned path.

Algorithm 4.2: Pruning Method.

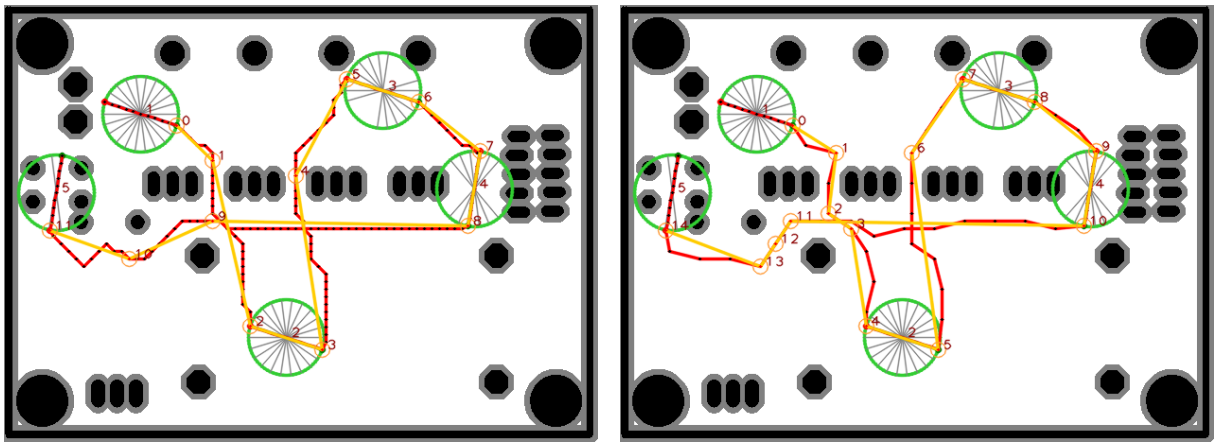
Input: Original Path $\{x_1, \ldots, x_N\}$
Output: prunedPath

```
1  prunedPath := ∅;
2  keepNodes := ∅;
3  n_attempts := 0;
4  j := N;
5  prunedPath.push(x_j);
6  foreach i ∈ [1, j − 1] do
7      if n_attempts ≥ n_max_attempts ∨ i = j − 1 then
8          n := getLowestNodeCost(keepNodes);
9          j := n;
10         prunedPath.push(x_j);
11         n_attempts := 0;
12     if LineOfSight(x_i, x_j) = true then
13         n_attempts + +;
14         keepNodes(i);
```

The pruning operation is carried out as follows, first it receives a set of nodes $\{x_1, \ldots, x_N\}$ from the previous path, where x_1 and x_N are the first and the last node of the path. Then, the node x_j (where $j = N$) is added to the empty set – *prunedPath*. After that, for each node x_i, where $i \in [0, j-1]$, it is verified the LOS between the nodes x_i and x_j. In case of LOS the variable $n\_attempts$ is incremented and the node x_i is added to the set *keepNodes*. The purpose of this set is to store a couple of nodes that have a direct line of visibility with node x_j, because the first node found may not be the most suitable, and so some possible nodes are stored in the set *keepNodes*. When the maximum limit of attempts is reached or $i = j - 1$, it is calculated which node present in the set *keepNodes* will be the most appropriate to be added to the pruned path. The criteria used to select the best candidate is the cost between each node

$n \in$ *keepNodes* and the node x_j; the one with lowest cost is returned by *getLowestNodeCost* (x_n). Then, x_n is added to the pruned path and the next x_j is updated by assigning $j = n$. The procedure will repeat until a complete path is generated.

Figure 4.9 illustrates the result of the application of the pruning method to the previous generated paths of Figure 4.8. In both figures, it is illustrated the pruned path in yellow and the previous path created by the searching phase in red. There were a lot of nodes that were deleted, resulting on a simplified version of the path.



(a) **(b)**

Figure 4.9: Pruning phase of the generated path in Figure 4.8a and Figure 4.8b, by the searching phase. In both figures, the red colour represents the search path and the yellow corresponds to the pruned path.

4.4.3 Smoothing Phase

The majority of the pathfinding algorithms such as A*, Theta*, Phi*, Dijkstra and others, only requires the start and final position and the problem is solved when the planner reaches the final position. However, in this path planning problem, the orientation of the start and goal position are also considered, which means that the path has to begin in the start position with the respective orientation, and achieve the goal position with the assigned orientation. Additionally, the path has to respect the minimum turning radius specified in the problem constraint.

The last phase called smoothing phase is the one responsible for integrating all these constraints. In order to accommodate the stipulated turning radius and the target orientations, it was implemented a

method based on the Dubins curves introduced in Section 2.5.1. This method calculates all the possible Dubins curves that can be created between a pair of nodes and returns the shortest one. It receives a pair of nodes denoted by its Cartesian coordinates and by its orientation, as follows $n = [x, y, \theta]^T$, where $\theta \in [0, 2\pi]$ and also the value of the turning radius – ρ.

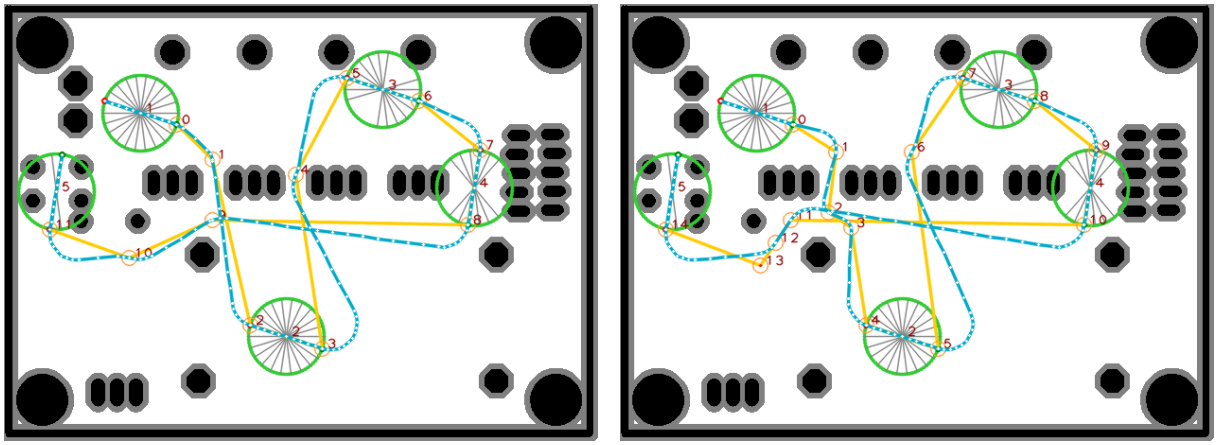
The pseudo code present in Algorithm 4.3 details the approach used to smooth the previous path (pruned path) and integrate all of the above constraints. There are three values of turning radius, ρ, that are used to create different Dubins curves with different radius of curvature, these values are designed by ρ_{max}, ρ_{pref} and ρ_{min}. This phase starts by receiving the pruned path $\{x_1, \ldots, x_N\}$, where $x_1 = [x, y, \theta_1]^T$ is the first node and $x_N = [x, y, \theta_N]^T$ is the last one. Then, the smoothing operation begin by selecting a pair of nodes and compute a possible Dubins curve between them. The returned curve is free of collisions and can be created by any of the three ρ, depending on the distance between the nodes and the obstacles around them. In case of successful creation of the Dubins curve, it is added to the *smoothPath*, otherwise it will be selected the next node of the prunned path and the procedure will repeat until computing a possible Dubins curve.

Algorithm 4.3: Smoothing phase.

Input: Pruned Path $\{x_1, \ldots, x_N\}$
Output: smoothPath

```
1  smoothPath := ∅;
2  i := 1;
3  j := i + 1;
4  while j ≤ N do
5      if createDubinsCurve(x_i, x_j) = true then
6          smoothPath.push(dubinsCurve);
7      else
8          while createDubinsCurve(x_i, x_j) = false do
9              j++;
10         smoothPath.push(dubinsCurve);
11         i := j;
12         j := i + 1;
```

Figure 4.10 shows the smooth path that was created from the pruned paths. The two examples validate the turning radius and it is possible to see the smooth path in blue.



(a) (b)

Figure 4.10: Smoothing phase of the generated path in Figure 4.9a and Figure 4.9b, by the pruning phase. In both figures, the yellow colour corresponds to the pruned path and the blue represent the smooth path.

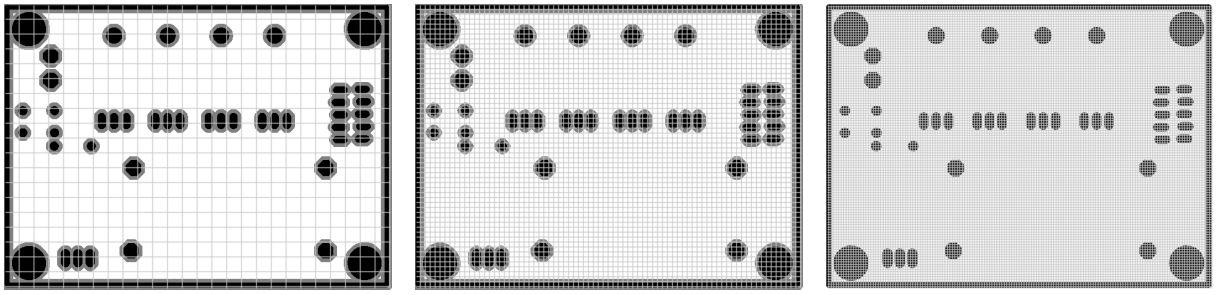
4.5 Path Planner Characterization

The following section presents the results of the planner's performance in different situations/ configurations. The results were compared, analysed and discussed, relatively to: (i) the different cells size for representing the map, (ii) the utilization of different heuristic methods to estimate the distance to the goal, (iii) the variation of the weights on the cost function during the searching phase, (iv) the influence of a weighting and the variation of the safety distance, and (v) the variation of the radius of curvature.

4.5.1 Map Resolution

Cell decomposition method was used for map representation. This method introduced in Section 2.2.1 relies on a map division into cells, therefore was fundamental to analyse the influence of the cells size, which changes the grid map size.

In order to evaluate the impact of the cells size, it will be used the example presented in Figure 4.11, which contains three different resolutions of the same environment. The dimensions of the PCB mask used in this example are 78.5 mm x 57.75 mm. The grid size for each resolution shown in Figure 4.11a, Figure 4.11b and Figure 4.11c, corresponds to the following grid sizes 26x19, 58x79 and 157x116 cells, respectively.



(a) (b) (c)

Figure 4.11: PCB decomposition in squared cells with three different resolutions: (a) 3 mm – grid size of 26x19 cells, (b) 1 mm – grid size of 58x79 cells and (c) 0.5 mm – grid size of 157x116 cells.

As previously explained in Section 4.2, the map is divided into cells and the cells that intersect the obstacles are considered blocked. Even if the intersection is very small, the cell is considered occupied and thus, increases the size of the obstacle, as shown in Figure 4.4 and Figure 4.12a. The size of the cells changes the time spent on the searching phase, decreasing as the cells size increases, since it has fewer cells to look for. However, if the size of the cells is considerable large, as represented in Figure 4.11a, not all the real free space can be reached. This usually leads to a significant increase of the obstacles occupancy, in this case it is about 37.45% whereas the other ones have around 28.50%.

The problems that may arise when using this cell decomposition method are presented in Figure 4.12. The first problem in Figure 4.12b shows two obstacles (a circle and a triangle) and the extra space they occupied. Figure 4.12a represents a situation where one zone of the map is mistakenly assumed to be occupied, and thus the planner has to re-plan a path around it.

The value of the cells size should be sufficient to maximize the speed of the search algorithms, while at the same time allowing all areas of the map to be reached.

4.5.2 Heuristic Cost Results

Previously, in Section 2.4.2 it was explained that the algorithm A* is guided by an heuristic cost function $h(n)$, and it was shown three possible heuristic functions, namely, the Manhattan, the diagonal and the Euclidean function. Regarding to these possibilities, this sections intends to ascertain which function is more efficient and more reliable, testing each one under the same conditions.

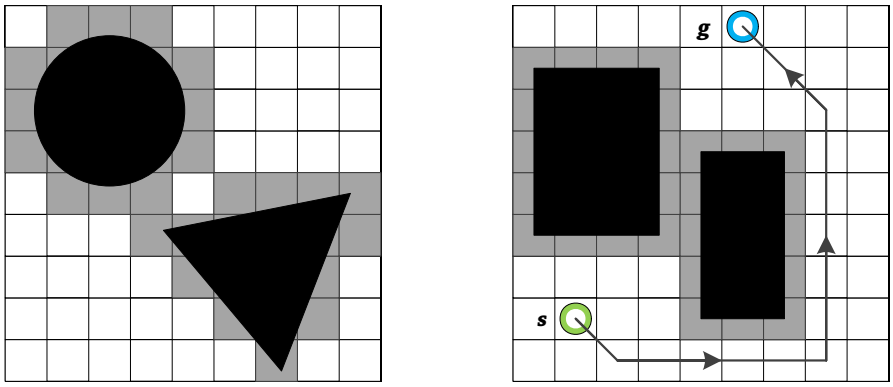


Figure 4.12: Obstacles representation in a square grid. The colour black represents the obstacles, and the grey cells correspond to the extra space set as obstacle. In (b) the segments represents the planned path between the start and goal point, s and g respectively.

The tests were conducted in two PCB masks with different layouts and dimensions. Each PCB mask was assigned five targets in different positions on the PCB mask. In addition to the specification of the position and orientation of each target, the parameters of the path planner were configured with the following values present in Table 4.1. The definition of each parameters is explained in Section 4.4.

Table 4.1: Parameters used for the test 1.

Parameter	Value
Cells Size	$1.0\ mm$
Segment Size (SS)	$10\ mm$
Penalty Radius (PR)	$2\ cells$
Penalty Weight (PW)	2
Safety Distance (SD)	$1.0\ mm$
Max Turning Radius (ρ_{max})	$3.5\ mm$
Pref Turning Radius (ρ_{pref})	$2.5\ mm$
Min Turning Radius (ρ_{min})	$1.5\ mm$

In order to analyse the performance of each heuristic method, the following indicators will be used to compare them:

- Exp. N – Expanded Nodes (in nodes) number of nodes visited during the searching phase;
- T – Time (in seconds) time spent by the algorithm to find a solution in the first phase;
- L – Length (in mm) length of the path constructed in the searching phase;
- PT – Pruned Time (in seconds) time used in the pruning phase;

- PL – Pruned Length (in mm) length of the resulted path in pruning phase;
- ST – Smooth Time (in seconds) time spent in the smoothing phase;
- SL – Smooth Length (in mm) length of the smooth path.

Initially, the tests were performed in the first PCB mask, called PCB 1, shown in Figure 4.13a. This PCB mask is composed by a grid of 79x58 cells, were each cell represents a square of 1 mm. The obstacles percentage is about 30.34%, the percentage of cells that are considered blocked and not traversable. The configuration of each waypoint used for the PCB 1 is present in Table 4.2.

Table 4.2: Waypoints configurations for the PCB 1.

w_i	$x\ (mm)$	$y\ (mm)$	$\theta\ (rad)$
1	18.00	43.43	5.970
2	37.26	13.82	5.970
3	50.00	16.50	0.0
4	62.07	33.56	4.556
5	6.83	32.944	1.414

Table 4.3 presents the results performed in the PCB 1 with the waypoints configurations presented in Table 4.2. The path created using each heuristic cost is represented in Figure 4.13b. According to the results shown in Table 4.3, the Euclidean distance is the one that explored fewer nodes during the search phase, then the Diagonal distance and with a further 68.0% the Manhattan distance. Regarding the path lengths, the difference is not significant, the maximum difference in the final path (smooth path) is only 0.78% more than the smallest. The Diagonal distance was the one that consumed less time (total time), the Euclidean distance spent 2.10% more and the Manhattan consumed about 13.55% more than the Diagonal distance. In terms of performance the Diagonal and Euclidean distance are very close, while Manhattan distance was the one with the shortest smooth path. The resulted paths from the tests are depicted in Figure 4.13b, and in Figure 4.13a it is represented the PCB mask 1 and the smooth path created by the Euclidean heuristic cost.

Another tests were executed in a different PCB mask, named PCB 2, illustrated in Figure 4.14a. The position and orientation of each waypoint for the PCB 2 is present in Table 4.4. As well as the previous one, the size of the cells is about 1 mm, which divide in a grid of 187x157 cells. The obstacles occupancy is around 26.67%, a little less than the PCB 1. However, PCB 2 presents more obstacles than the previous

Table 4.3: Tests of the heuristic costs in PCB 1.

Heuristic	Exp. N	T	L	PT	PL	ST	SL
Manhattan	1596 (+68.0%)	0.13746 (14.33%)	217.15 (+12.98%)	0.000843 (+51.08%)	144.96 (+0.60%)	0.002456 (−)	194.82 (−)
Diagonal	953 (0.32%)	0.120229 (−)	192.20 (−)	0.000577 (+3.41%)	144.10 (−)	0.003173 (+29.19%)	195.11 (+0.15%)
Euclidean	950 (−)	0.122701 (2.06%)	193.76 (+0.81%)	0.000558 (−)	144.67 (+0.40%)	0.003301 (+34.41%)	196.33 (+0.78%)

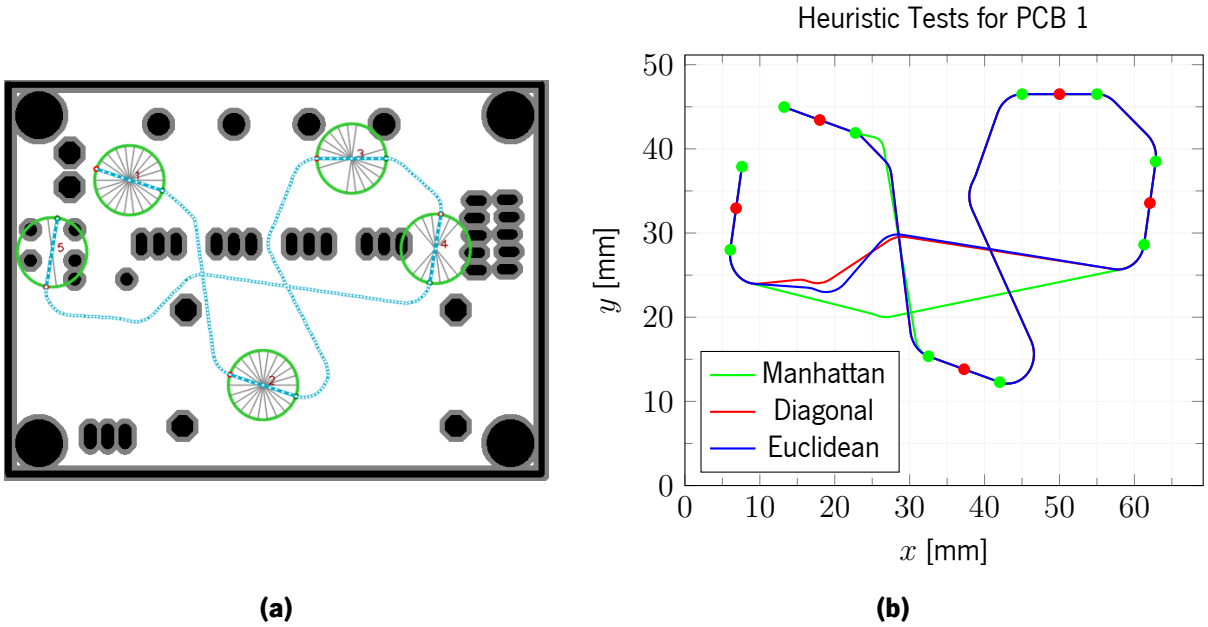


Figure 4.13: Smooth paths resulted from the heuristic distances tests in PCB 1. (a) represents the PCB 1 with the smooth path created through the Euclidean heuristic; (b) depicts the resulted paths of each heuristic method. The red circles correspond to the waypoints and the green circles to the beginning and the end of the straight linear segment in each waypoint.

one, they are smaller and distributed around the PCB, composing a layout more complex to find a solution than the previous.

Table 4.5 presents the values of the indicators resulted from the test performed in the second PCB mask, which is illustrated in Figure 4.14a. These values will be used to compare the different heuristic distances. Unlike the previous test, here the Manhattan distance was the one that had the lower number of expanded nodes in the searching phase, then the Euclidean and after with 61.98% more the Diagonal distance. The total time spent, the Diagonal loses for about 257% from the Manhattan, and the Euclidean

Table 4.4: Waypoints configurations for the PCB 2.

w_i	$x\ (mm)$	$y\ (mm)$	$\theta\ (rad)$
1	24.17	132.33	1.414
2	73.33	99.00	4.556
3	50.00	55.17	4.839
4	169.17	12.33	1.131
5	115.00	48.17	1.414

is between them, but more closely from the Manhattan (125.73%). Relatively to the developed path they are practically similar, the maximum different is just around 8 mm.

Table 4.5: Tests of the heuristic costs in PCB 2.

Heuristic	Exp. N	T	L	PT	PL	ST	SL
Manhattan	5962 (−)	1.824448 (−)	501.89 (+3.25%)	0.00164 (+5.67%)	402.48 (+0.57%)	0.003877 (+2.11%)	460.61 (+1.52%)
Diagonal	9657 (+61.98%)	6.527387 (+257.77%)	486.10 (−)	0.001552 (−)	400.19 (−)	0.003797 (−)	453.71 (−)
Euclidean	7534 (+26.37%)	4.125118 (+126.10%)	486.70 (+0.12%)	0.001714 (+10.44%)	401.97 (+0.44%)	0.003984 (+4.92%)	461.75 (+1.77%)

The paths associated to the results discussed before are shown in Figure 4.14; one path for each heuristic distance is illustrated in Figure 4.14b, and in the left there is the mask of the PCB 2 with the smooth path created by the Euclidean distance.

According to the results shown in this section, there is no solid conclusion as to which heuristic method is most suitable for the general problem. However, based on their performances, the Euclidean distance seems to be the one with better results, considering the average time spent and the length of the smooth path in the previous results, the Euclidean distance presents the best compromise. Thus, for the remaining of this document, the heuristic method that will be used is the Euclidean distance.

4.5.2.1 Variation of the Heuristics Weights

As previously mentioned, the search phase of the algorithm is based on a cost function $f(n) = g(n) + h(n)$, that represents the sum of the the total cost between the start to the current node ($g(n)$), and the

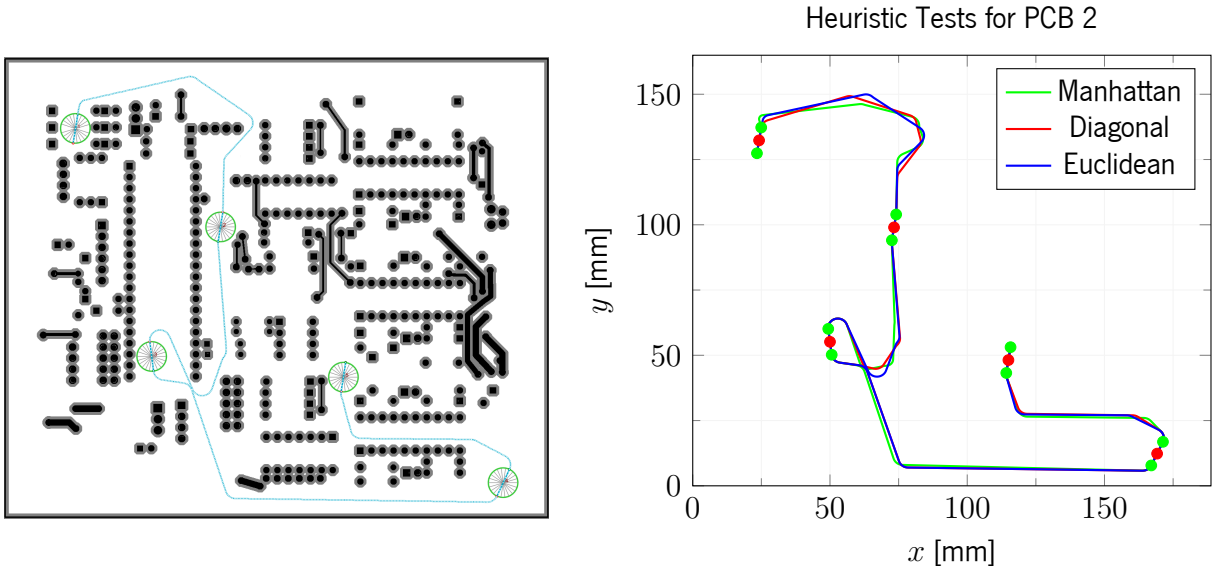


Figure 4.14: Smooth paths resulted from the heuristic distances tests in PCB 2. (a) represents the PCB 2 with the smooth path created through the euclidean heuristic; (b) depicts the resulted paths of each heuristic method. The red circles correspond to the waypoints and the green circles to the beginning and the end of the straight linear segment in each waypoint.

estimate cost between the current to the goal node ($h(n)$). Each cost has an adjustable parameter, wG and wH, which is responsible for scaling the magnitude of the respective weight cost.

In order to analyse the contribution of the each parameter, several combinations of wG and wH were tested. The results have shown that, when the difference between wG and wH increases, that is, when the value of wG decreases, the success rate of the planner also reduces. It may even reach 0%, if the value of wG is close to zero. Due to the planner be more concerned about reaching the goal, the contribution of $g(n)$ decreases and so the search path is constructed very close to the obstacles; thus it increases the chances of failure during the smooth phase. In this case, the consumed time is much lesser than with the contribution of $g(n)$, but the success rate is too small.

On the other hand, if the value of wH begins to decrease the consumed time will increase exponentially and it may raise about 1896.35% over the average time. However, the success rate of the planner is practically near 100%, the only problem is the consumed time which is too much.

In summary, the combination of these factors has to be good in terms of spent time as well as the success rate of the planner. A good approach is to set the value of wH slightly higher than the wG, that

speeds up the planner's performance and does not affect the success rate.

4.5.3 Safety Distance and Map Weighting Results

The Safety Distance (SD) is a variable that specifies the minimum distance between the final path and the obstacles present in the PCB mask. It is always guaranteed that the final path fulfils this SD distance from the obstacles. This parameter is very important, its main purpose is to ensure that whatever the width of the path or the dimensions of the robot that had to follow the generated path, never collide with the obstacles in the environment. As previously said, the generated path is punctual and has no dimension, so in order to avoid collisions with the obstacles the SD parameter can be adjusted according to the problem (dimensions of the material/ robot). Additionally, it is possible to define a minimum distance that the material or robot should have between the obstacles along the path, and thus, the SD parameter can be represented as SD = *size of the material or robot* + *distance from the obstacles*.

In this case, the material considered is an optical fibre cable that has to be placed along the generated path. The value of SD is set as SD = *ray of the optical fibre cable* + *distance from the obstacles*, since this parameter is adjustable, it can be adjusted to other materials/ problems. In this implementation the SD increases the obstacles size by its value and then the matrix of obstacles is updated. This variable affects directly the obstacles space and increases the obstacles percentage in the PCB.

Before the search phase of the algorithm, it is performed the methods that construct the representation of the map. At this point, the matrix of obstacles is constructed and updated if the SD's value is greater than zero, and then it is executed the method responsible for weighting the cells close to the obstacles, the method called Map Weighting. This method is based on two parameters: the Penalty Radius (PR) and the Penalty Weight (PW); the first one is the radius of cells centred in the blocked cells that will be penalized, and the second one is the maximum value that a penalized cell can have.

The main goal of the Map Weighting is to improve the planner's performance and to move away the path from the obstacles. It was performed two tests to see evolution of the planner's performance for different configurations of these two parameters, the PCB mask used was the PCB 2. The first one is shown in Figure 4.15, it represents the time and length variation for different values of PW. The results revealed that for values of PW $\in [0.25; 1.25]$ the success rate of the planner is below 50%, but when the

value is between $[1.25; 5.0]$ the success rate of the planner is almost 100% and the time tends to decrease with increasing PW. Relatively to the path length the conclusions that resulted was that for small values of PW the path is created near to the obstacles, and the path goes through zones of obstacles, whereas for bigger values such as PW $\in [2.0; 5.0]$ the path is constructed further away from the obstacles, since the weight penalization is bigger near the obstacles and so the path tends to be created with the smaller possible cost.

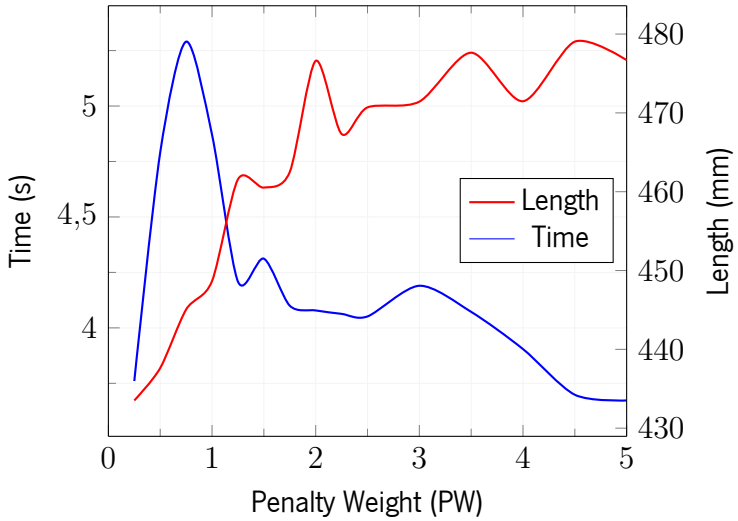
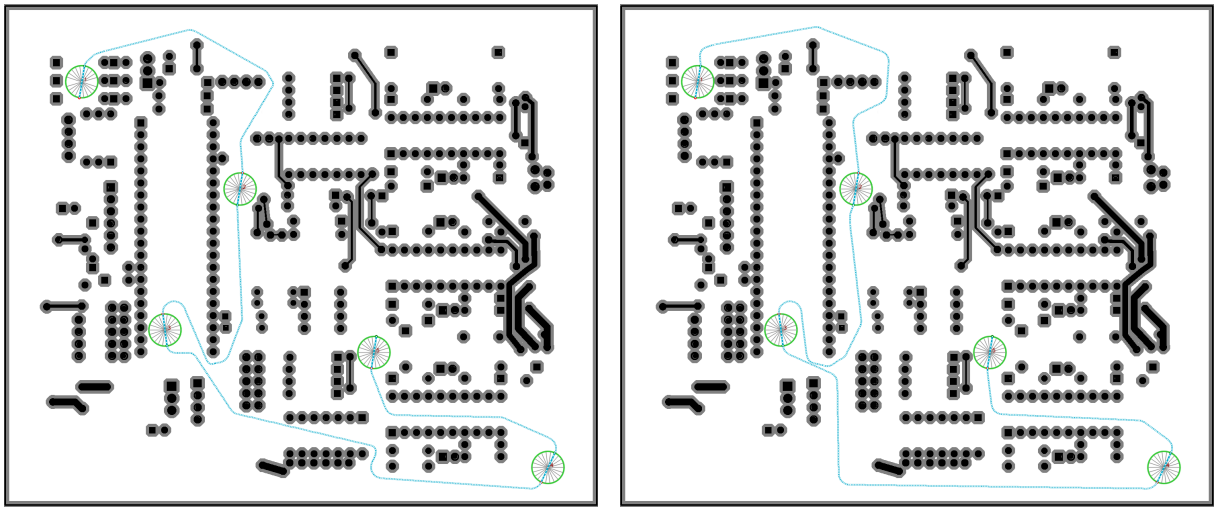


Figure 4.15: Time and length variation along Penalty Weight (PW). For this test the parameter's configuration used was the following PR=3 cells, wG=0.85, wH=1.0 and Δ=3.

The second performed test intends to evaluate the effect of the PR parameter. There is two examples in Figure 4.16 that shows what changed for different values of PR. Figure 4.16a represents the resulted path when it was used the configuration PR=1, it shows that the path in some places is close to the obstacles; on the contrary Figure 4.16b shows a path that is much more distanced from the obstacles with PR=5. Despite the path represent in Figure 4.16b seems to be a better solution, this one consumed more time than the Figure 4.16a; there are more weighted cells with PR=5 than PR=1, thus the planner has to explore more cells which increases the consumed time.

4.5.4 Curvature Radius Results

The purpose of this section is to present the results relatively to the curvature radius of the path, to demonstrate that the final solution is a combination of arcs and straight segments of different dimensions.



(a) **(b)**

Figure 4.16: Comparison of path created by different values of PR (Penalty Radius). The results shown in (a) uses PR=1 and in (b) the PR's value is 5. For both tests the search parameters were PW=2.5, wG=0.85, wH=1.0 and Δ=3.

As previously mentioned in Section 4.4.3, the Dubins curves can be created between two points according to their orientation and the curvature radius specified (ρ) . Since this implementation uses three possible values for the radius of curvature (ρ_{max}, ρ_{pref} and ρ_{min}), the final path will present Dubins curves created through different values of curvature radius. Moreover, it will show that these parameters are configurable and for different values the planner can find a solution.

In order to verify that in the final solution, the planner selects the most suitable Dubins curve in different sections of the path, according to the curvature radius set to ρ_{max}, ρ_{pref} and ρ_{min}, was calculated the value of the curvature (k) along the path. Since the construction of the smooth path is based on the Dubins curves, the resulted path is a combination of arcs and straight lines. The curvature of a straight line is defined by $k = 0$ and for a circle of radius R the curvature follows $k_{circle} = 1/R$.

The example presented in Figure 4.16b was used to verify that in this built path, it respects the curvature radius specified along the path, and also that the radius of curvature vary along the path. Figure 4.17a shows the curvature profile of the mentioned path, it is possible to see through the curvature profile that the path fulfil the constraints assigned, which in this case were ρ_{max}=3.5 mm, ρ_{pref}=2.5 mm and ρ_{min}=1.5 mm. In the left side is the heading profile of the path, the variation of the orientation along the length of the path.

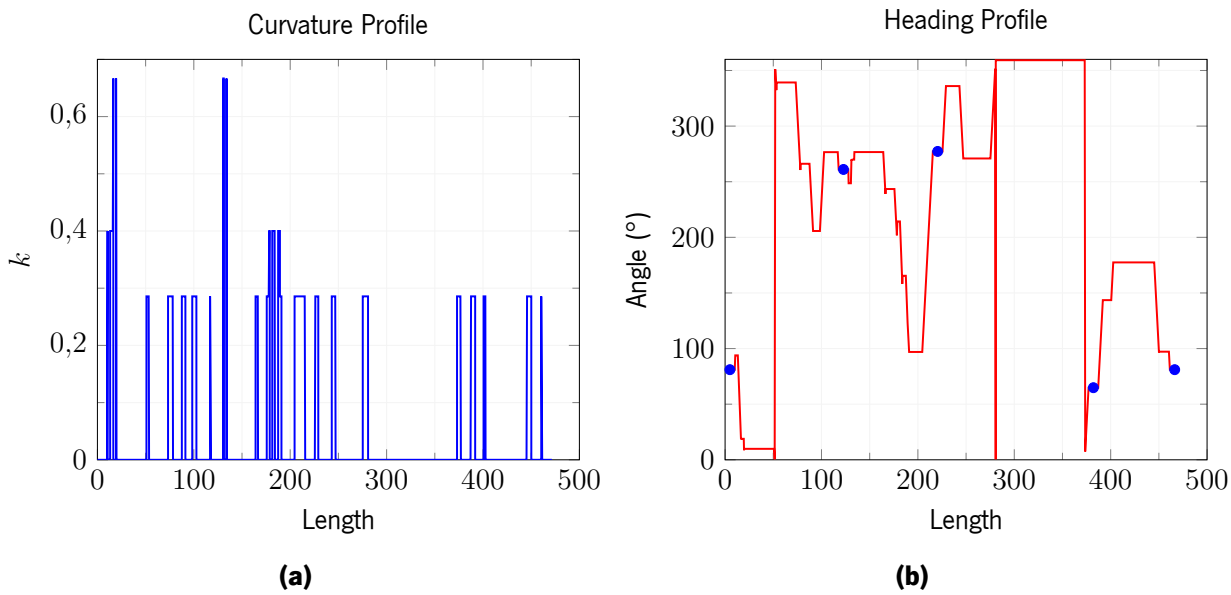


Figure 4.17: Curvature and heading profiles of smooth path depicted in Figure 4.16b. The radius of curvature used were ρ_{max}=3.5 mm, ρ_{pref}=2.5 mm and ρ_{min}=1.5 mm.

Another test was performed using the same PCB mask, but with an increase in the turning radius to ρ_{max}=8.0 mm, ρ_{pref}=6.0 mm and ρ_{min}=4.0 mm. The path created from this configuration is represented in Figure 4.18. The Figure 4.18b shows two paths, the red one is the previous path created with smaller turning radius (ρ_{max}=3.5 mm, ρ_{pref}=2.5 mm and ρ_{min}=1.5 mm) and the blue one is the path created in this test. The new configuration with higher values of turning radius turns the path more smooth (blue path present in Figure 4.18b).

As well as the previous path, it is shown the results of the curvature and heading profile of the path presented in Figure 4.18, that was built with higher values of turning radius comparatively to the one presented in Figure 4.16b. The results of this new configuration are present in Figure 4.19, it validates the variation of the turning radius along the path, according to the values specified.

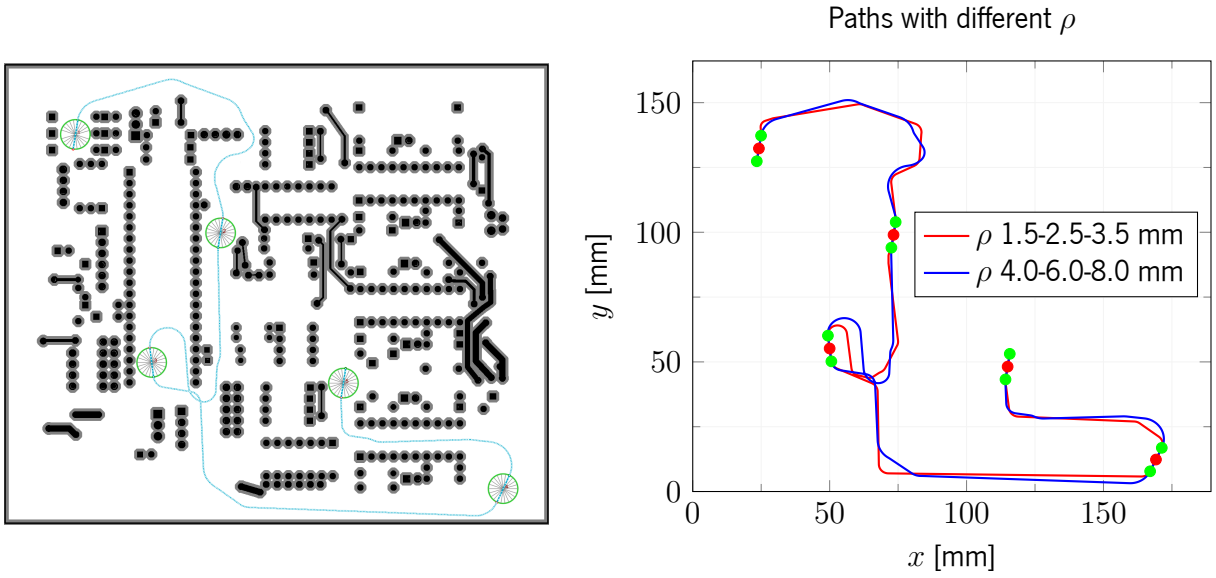


Figure 4.18: Smooth path for higher values of turning radius. (a) represents the path created with the radius of curvature set to ρ_{max}=8.0 mm, ρ_{pref}=6.0 mm and ρ_{min}=4.0 mm; (b) depicts the previous path presented in Figure 4.16b in red and the path of the Figure 4.18a in blued. The red circles correspond to the waypoints and the green circles to the beginning and the end of the straight linear segment in each waypoint.

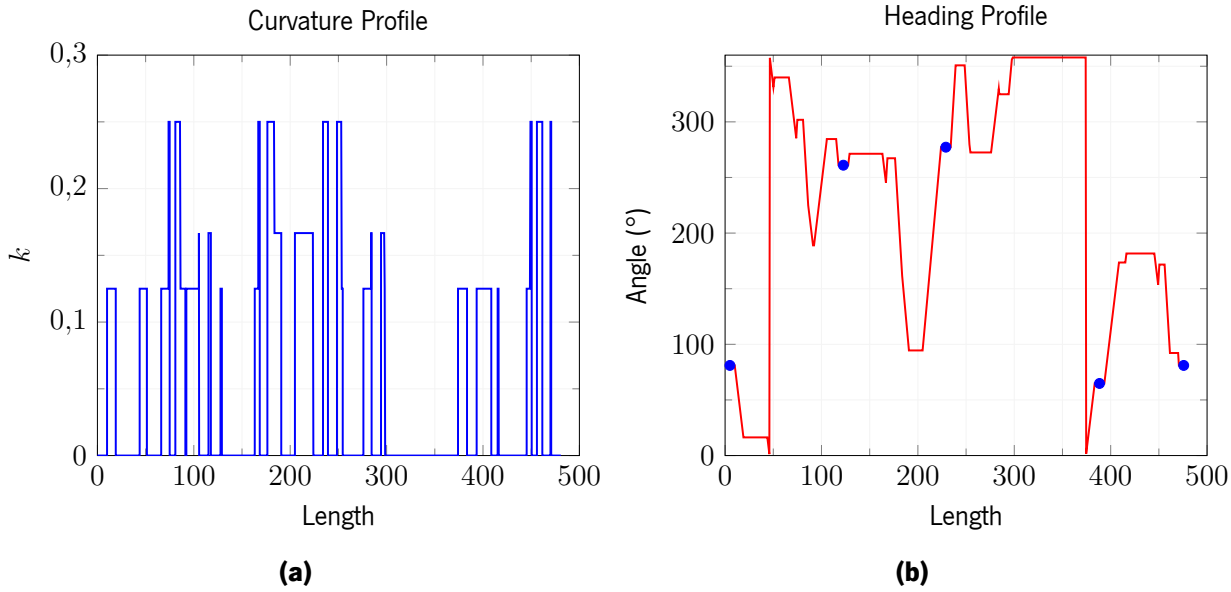


Figure 4.19: Curvature and heading profiles of smooth path in the Figure 4.18a. The radius of curvature used were ρ_{max}=8.0 mm, ρ_{pref}=6.0 mm and ρ_{min}=4.0 mm.

4.6 Results Discussion

Overall, the results presented in this chapter have shown the features of the path planner, an highly tunable module within a wide range of parameters. The previous sections provided an explanation about the influence of each parameter and configuration in this module. The goal of all these tests was to understand the behaviour of each configuration, in order to adjust it to the best configuration for the problem.

Initially, it was noticed that the size of the cells should be chosen according to the dimensions of the obstacles, in order to obtain a better representation of the environment, namely the free space and the obstacles space. At the same time, the value of the size of the cells should be as large as possible to maximize the speed of the planner.

Then, it was presented the results for the heuristic costs, where the Euclidean distance revealed to be the most suitable for the purposed problems; thus, it was used for the following tests. The fact that the planner has two parameters that can scale the magnitude of the contribution of each cost ($h(n)$ and $g(n)$) during the search phase, it showed that the speed can be increased when the contribution of $h(n)$ is greater than $g(n)$. However, if the difference is too high it is more likely the planner will fail.

Next, the safety distance and the map weighting method were discussed. The first shown that is responsible for ensuring a minimum distance between the path and the obstacles, and also to guarantee that whatever the with of the path, never collide with the obstacles in the environment. Relatively to the method of map weighting, it revealed to be a great addition to the planner, since it decreases the computational time needed to find a solution and also it can increase the distance of the path from the obstacles, only by adjusting the parameters PR and PW (Penalty Radius and Penalty Weight).

The last tests presented in Section 4.5.4 were performed to validate that the developed planner is customizable in terms of turning radius. It is not restricted to a set of values, the results have shown that it was able to find a solution when the values of the radius of curvature were changed, in this case were increased. The developed solution implements three possible values of turning radius (ρ_{max}, ρ_{pref} and ρ_{min}), however it could also add more values or even a range of possible values, which would increase the flexibility of the generated path.

4.6. Results Discussion

Overall, the planner shown to be highly configurable and possible to adjust to various problems, setting the parameters to the values more suitable for the problem.

Chapter 5

Motion Planning

The present chapter details the implementation and validation of the module Motion Planner introduced in Section 3.2. The validation of the system will be performed in simulation and in real environment, using the robotic manipulator Sawyer. The following sections are organized and divided as follows: first, it describes how the trajectories are generated for the simulated robot and also for the real robotic arm. Then, it shows the motion sequence performed by the robot in this specific application and its respective results. The trajectory performed by the real robot is compared with the planned trajectory, regarding to (i) the angular position of each joints and (ii) the position and velocity of the end-effector along the trajectory. The last section summarizes and discusses all the results presented.

The Motion Planner receives the desired path created by the Path Planner, and then it generates the trajectory for the robot Sawyer and displays it in the RViz, as can be seen in the overview of the module represented in Figure 5.1. The RViz displays all the objects present on the scenario, the loaded path and also the generated trajectory. After the validation of the trajectory, the module sends it to the real robot and receives the feedback.

5.1 Trajectory Planning

The Motion Planner module runs on top of ROS (Robot Operating System), introduced in Section 3.2.1. The main motivation to develop this module based on a ROS distribution is due to three factors: (i) MoveIt, (ii) RViz and (iii) Sawyer robot. The first is a motion planning library that offers various implementations

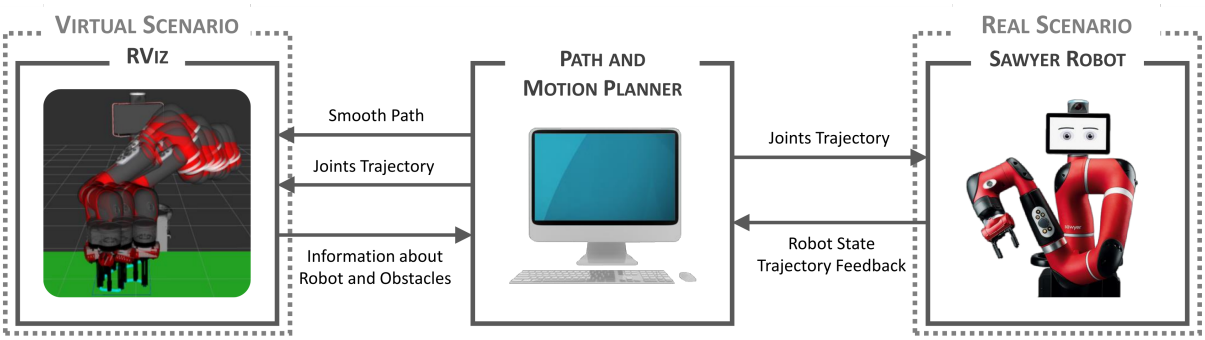


Figure 5.1: Overview of the Path and Motion Planner Modules.

of planning algorithms and can be integrated in a ROS environment (introduced in Section 3.2). RViz is a 3D visualizer to display sensor data and state information from ROS. The last one refers to the software of the robotic platform Sawyer, which was developed based on ROS.

5.1.1 Virtual Scenario

A virtual scenario that replicates the real environment in which the robot will be placed was created, in order to test the developed solution. So, that way it makes possible to analyse the behaviour of the robot, and to study the viability of its use in the context of the task. Moreover, the Motion Planner manages and controls the movements of the robot, which are displayed through RViz, resulting in a safer and practical way to test the generated trajectory for the robot, and to find and solve problems that may arise. The scenario is composed by: the model of the robot with its pedestal, the end-effector that is the basic electrical parallel gripper of Rethink Robotics, and the work table where the PCB will be placed to perform the task.

This simulation environment is displayed by the RViz, where the control application loads the model of the robot and the objects present in the scenario. The model of the robot is supplied by the company through a package, which contains the CAD model of all the components of Sawyer and, also contains the configurations of all joints, from the mechanical limits to the maximum velocity and acceleration of each joint of the robot, for example. These information is used by the motion planning algorithm while it searches for a feasible solution. A valid and feasible solution, if there is any possible one, respects the joints limits, avoid collisions with the obstacles present in the scenario and also, with the robot's own

mechanical structure. Besides that, the time parametrization of the trajectory is calculated according to the parameters set in the robot package.

As previously mentioned, the Motion Planner module uses some of the functionalities provided by the framework MoveIt during the generation of the trajectories. These features are accessible through a ROS node designated by *move_group*, which has services that implement algorithms for calculating forward and inverse kinematics of the robot. Furthermore, it has ROS topics where, for example, specific messages are published with information about the position of each joint and the generated trajectory. The next topics present a briefly explanation about these services and topics:

- **IK Service** - responsible for computing the representation in joint space of a representation described in Cartesian space;
- **FK Service** - responsible for calculating the representation of the end-effector in Cartesian space through a representation in joint space;
- **Compute Cartesian Path Service** - computes to a set of poses (position and orientation) the set of configurations of the robot in joint space;
- **Plan Service** - creates a trajectory between the current state of the robot to a desired configuration;
- **Execute Service** - responsible for send a request to the real robot, in order to perform the generated trajectory;
- **Joint States Topic** - ROS topic where information about the position of each joint is published.

The main trajectory is generated through a request message via the Compute Cartesian Path ROS Action. This request message is a goal message type, which is composed by a set of desired poses for a specific link of the manipulator. As well as all ROS services, it returns a feedback message of the requested message, this message includes the configuration of each joint in each desired pose sent by the request message. Besides that, the Compute Cartesian ROS Action sends a result message which notify if the trajectory was planned with success or not.

In this task, the main goal is to generate a trajectory for the collaborative robot Sawyer that should follow the previous path (created by the Path Planner). This path should be followed by the extremity of the end-effector, and thus, the *right_hand_tip* was assigned as the desired link. Figure 5.7 illustrates the end-effector used, which is the basic electrical parallel gripper manufactured by Rethink Robotics and, it also shows the axis at the end of the end-effector (*right_hand_tip*).

As previously said, this service computes the kinematic motion according to a set of desired poses. Firstly, it computes the kinematic solutions for the desired poses and, tries to minimize the cost by merging the solutions with the lowest joint displacement. After that, an algorithm called Iterative Parabolic Time Parametrization is executed, that is responsible for assigning the velocity and acceleration for each joint in each step of the trajectory. Besides that, it calculates the time step in each configuration of the trajectory.

5.1.2 Real Scenario

The robotic platform Sawyer provides an action server named "Joint Trajectory Action Server" that allows the execution of trajectories through a request sent by the control application (Motion Planner). The action server can operate in three different modes, such as (i) velocity, (ii) position and (iii) dynamics feed forward position. After testing each of the operating modes, the dynamics feed forward position was the one that stood out, as the performed trajectory was smoother, faster and precise. This mode accepts trajectories with position, velocity, and acceleration supplied for each joint, thus it is highly recommended by the Rethink Robotics to use this mode when using motion planners from MoveIt.

The Motion Planner establishes the communication with the action server and requests the execution of trajectories already validated in virtual scenario. In order to prepare the request, a goal message is constructed, that contains the sequence of steps of the desired trajectory, including the position, velocity and acceleration of each joint. The request message is sent to the action server and then, the robot calculates the control signals to the motor actuators, in order to perform the requested trajectory, respecting the temporal requirements.

During the execution of the trajectory, the action server provides feedback messages, which are constantly being published in the topic called "/robot/limb/right/follow_joint_trajectory/feedback". These messages contains information about the desired and actual position of each joint and also provide the error associated with the position of the joints. When the trajectory execution finishes, a result message is sent, informing that the trajectory has been completed.

Figure 5.2 gives an overview of the action interface between the action client (Motion Planner) and the action server (Joint Trajectory – Sawyer). The action protocol relies on ROS topics and services. These ROS topics are in a specific namespace, which in this case is "/robot/limb/right/follow_joint_trajectory/".

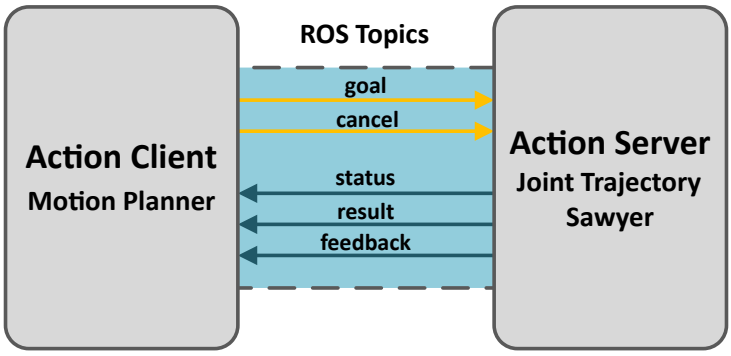


Figure 5.2: Action interface between the client and the server. The namespace of the ROS topics is "/robot/limb/right/follow_joint_trajectory/". The yellow arrows are messages sent by the client, whereas the blue arrows are sent by the server.

In the following topics is a brief description of each ROS topic presented in Figure 5.2:

- **goal** - used to send new goals to the action server;
- **cancel** - used to send cancel requests to the action server;
- **status** - used to notify the action client on the current state of every goal in the system;
- **feedback** - used to send to the action client periodic auxiliary information for a goal;
- **result** - used to send to the action client one-time auxiliary information upon completion of a goal.

5.2 Motion Sequence and Results

The following sections describe and analyse the motions performed by the robotic platform during the executing of the specified task. The motion sequence is generated through the Motion Planner module, where the obtained results will be analysed, in particular: (i) the position of each joint and (ii) the position and linear velocity of the end-effector. The simulation results presented were generated by a Processor Intel® Core™ i7-6700HQ @ 2.60 GHz, with 16 GB of RAM, and a graphical card NVIDIA GeForce GTX 960M, running the operating system Ubuntu 14.04 LTS of 64 bits.

5.2.1 Calibration of the PCB Position

Initially, a routine is always performed after the selection of the PCB, which outputs the exact position in which the PCB must be placed. The purpose of this routine is to ensure that PCB is placed in the right

position, preventing possible changes in the position of the working table. Therefore, placing the PCB in the specified area guarantees that the planned path will be performed in the same conditions than in the virtual scenario. This routine is based on a sequence of movements to four specific positions. These positions represent the four corners of the selected PCB, they are calculated based on the information extracted from the PCB. One of these points can be configured, ${}^{B}P_o$, which represents the origin of the PCB in relation to the referential of the base of the robot. Through the origin of the PCB, the other corners are calculated as follows:

$$ {}^{B}P_o = P_o = [x_o, y_o, z_o]^T \tag{5.1} $$

$$ \begin{aligned} P_{LS} &= P_o, & P_{RS} &= [0, w, 0]^T + P_o \\ P_{LI} &= [l, 0, 0]^T + P_o, & P_{RI} &= [l, w, 0]^T + P_o \end{aligned} \tag{5.2} $$

where L and R corresponds to left and right side, S and I corresponds to superior and inferior corner, and l and w represents the length and width of the PCB, respectively.

Figure 5.3 illustrates the sequence of motions performed by the robot in this routine, it moves the tip of the end-effector to the position of each corner of the PCB, in order to define the exact location where it should be placed. Figure 5.4 shows the exact same motions but with the real robot Sawyer.

All the motion sequences share the same initial and final state of the robot. This sate is called by Home Pose and it is illustrated in Figure 5.3a and Figure 5.4a. This posture is depicted in Figure 5.5 in different angles to show that this posture was selected to ensure that the robotic platform occupies a small area of the workspace and keeps the arms away from the working table. Table 5.1 presents the values of each joint of this configuration.

Table 5.1: Home pose for Sawyer robot.

Joint (i)	0	1	2	3	4	5	6
$\theta_i(^\circ)$	0	-85	-90	85	10	90	14

[1]Complete videos of the execution of the simulation trajectory - "Calibration of the PCB Position (Virtual Scenario)" and real environment - "Calibration of the PCB Position (Sawyer Robot)" available in: https://www.youtube.com/playlist?list=PLTiZ9ch2XF1Dw8zWhBw--MWfUa7YFs-9U.

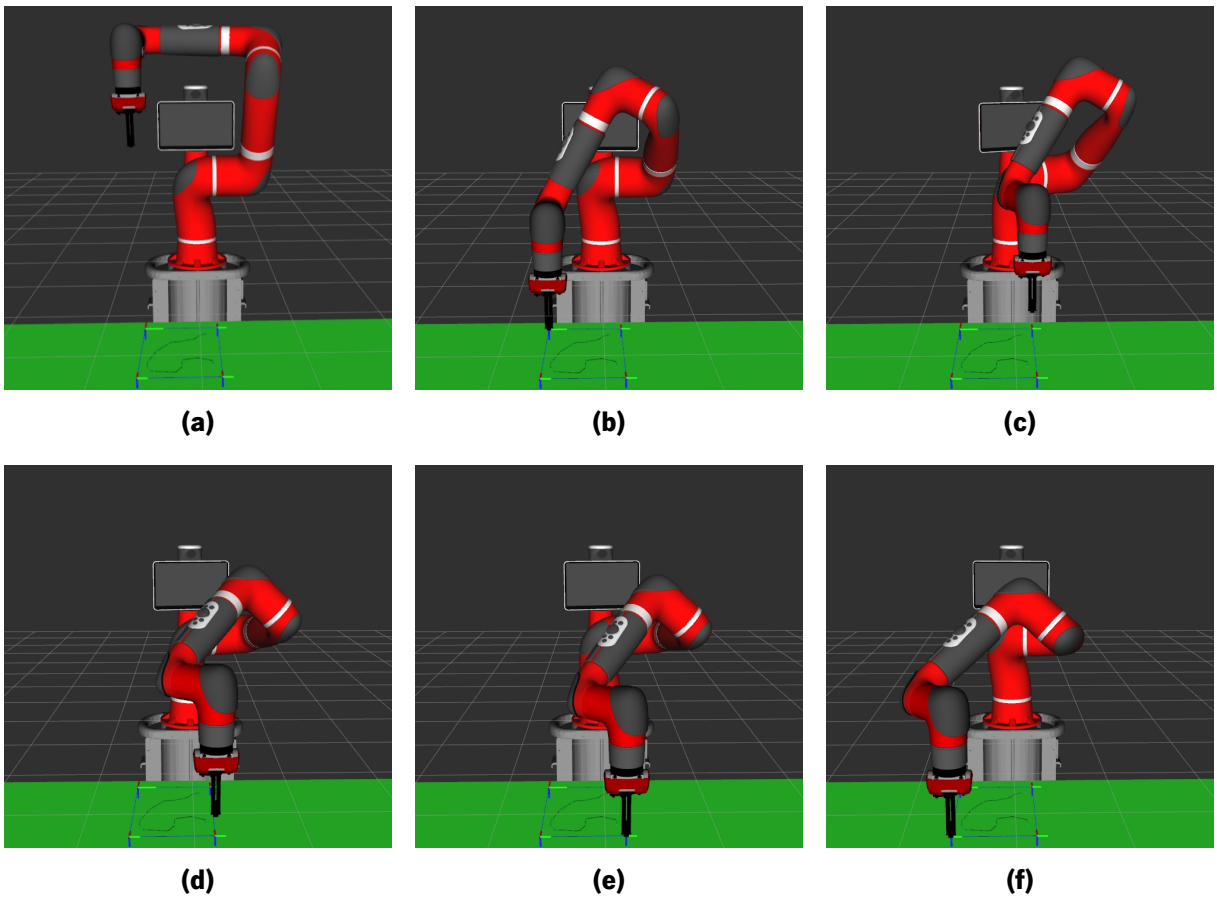
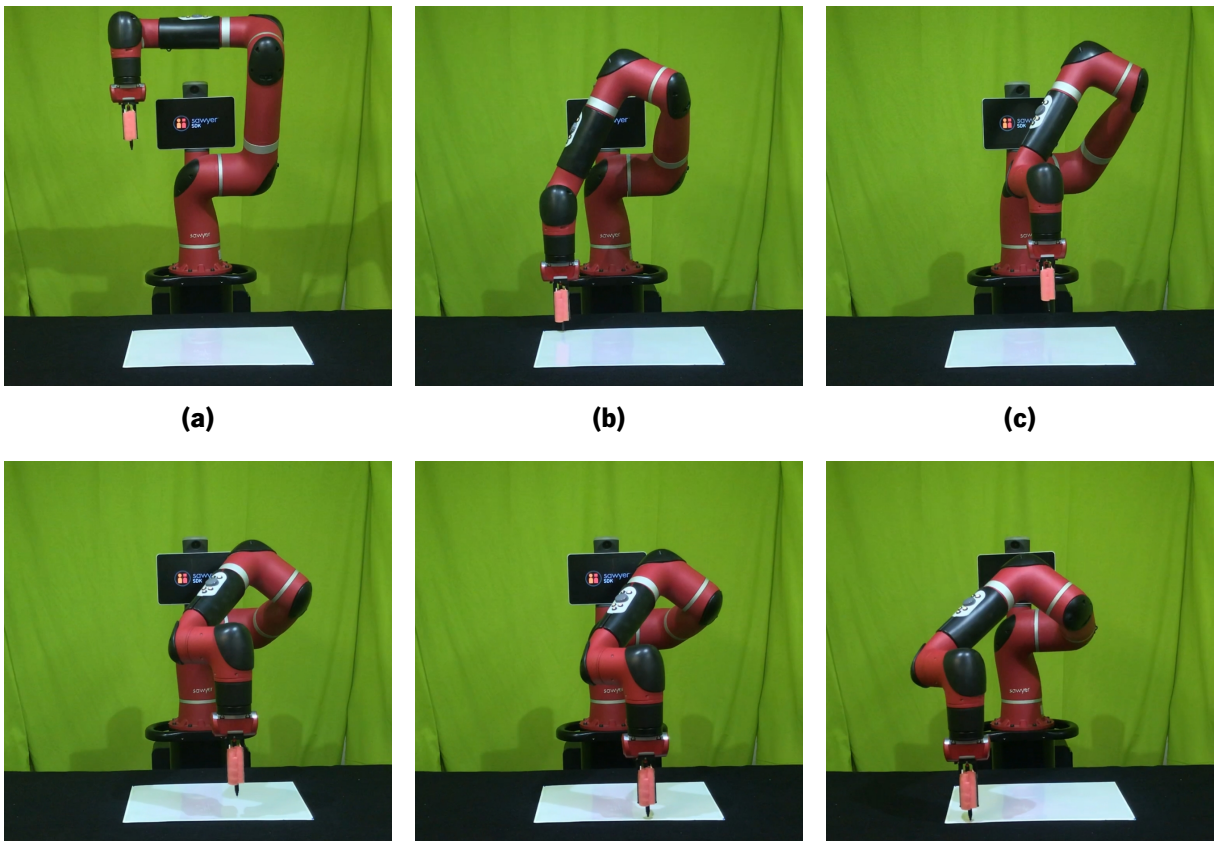


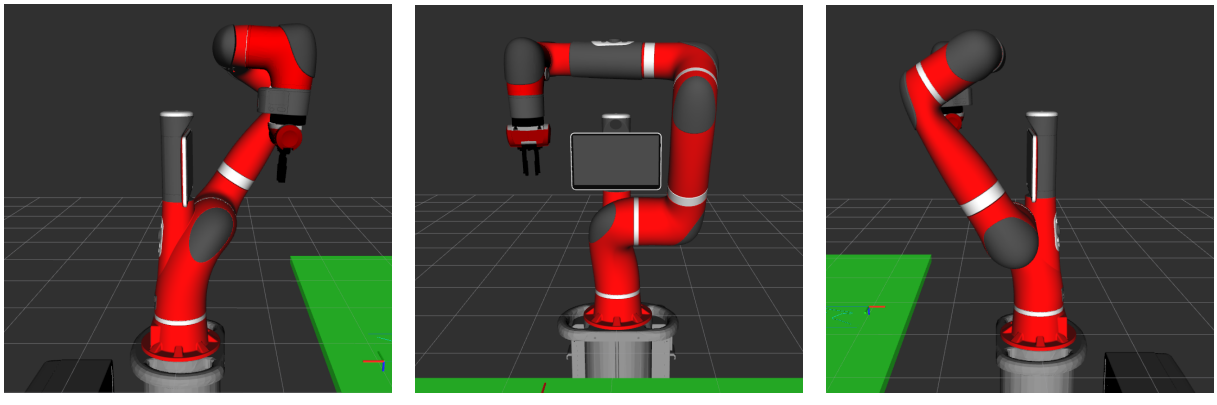
Figure 5.3: Motion sequence for the setup of the PCB position in the working table. The colour green represents the working table where the PCB is placed.



(a) (b) (c)

(d) (e) (f)

Figure 5.4: Motion sequence for the setup of the PCB position in the working table in a real scenario. The white area represents the area where the PCB can be placed.



(a) (b) (c)

Figure 5.5: Home Pose of Sawyer robot. The value of each joint of this posture is present in Table 5.1.

5.2.2 Path Loading

The Path Planner module is responsible for creating the geometric path for the PCB to be performed by the robot. This path is composed by the Cartesian coordinates in a two dimensional space (2D), $P = [x, y]^T$, and the respective orientation, θ, of each point of the path. The Motion Planner loads the path and transforms the position of each point in a three dimensional space (3D), $P = [x, y, z]^T$, and the orientation to a 3D rotational matrix R_{3D}.

The orientation θ is converted in a 2D rotational matrix, defined by:

$$R_{2D} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \tag{5.3}$$

Since the path was created in 2D space, there is just one axis of rotation in a 3D space, thus the rotation will be performed around the z axis. The rotational matrix in a 3D space for the rotation of z axis is represented as follows:

$$R_{3D} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} = \left[\begin{array}{c|c} R_{2D} & 0_{2\times1} \\ \hline 0_{1\times2} & 1 \end{array}\right] \tag{5.4}$$

Regarding the orientation of the desired path, there are two possible options: either the gripper follows the orientation of the path and rotates according to the orientation of the path, or the gripper orientation is fixed for all the points in the path. Figure 5.6 shows the two options, where in Figure 5.6a illustrates the loaded path with the orientation variation along the path, the orientation in each point of the path is taking into account; while, Figure 5.6b presents the option when the orientation in each point is the same. In the results presented next, the orientation of the end-effector is the same in each point of the path.

Considering the assigned task that the robot has to execute, an orientation constraint was imposed to the end-effector of the robotic arm. This orientation constraint is illustrated in Figure 5.7, where it shows that the gripper must be perpendicular and above the area where the PCB is placed. This constraint is only imposed when the robot is performing the trajectory for the desired path.

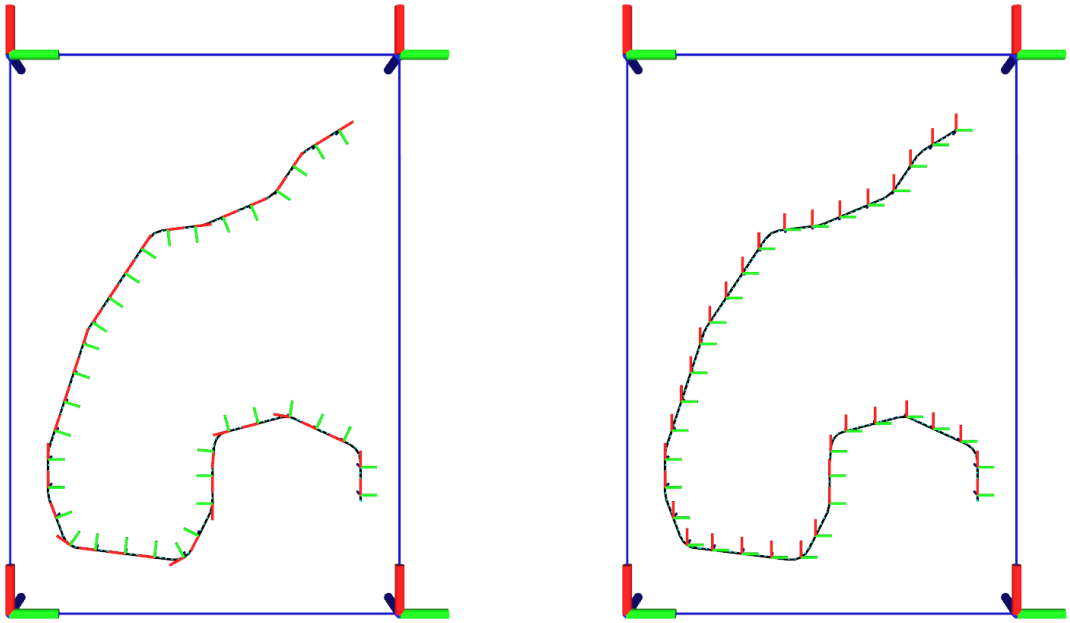


Figure 5.6: Loaded path. (a) represents the path loaded when the gripper orientation follows the orientation of the path, whereas (b) corresponds to the path loaded when the gripper orientation is the same for each point of the path.

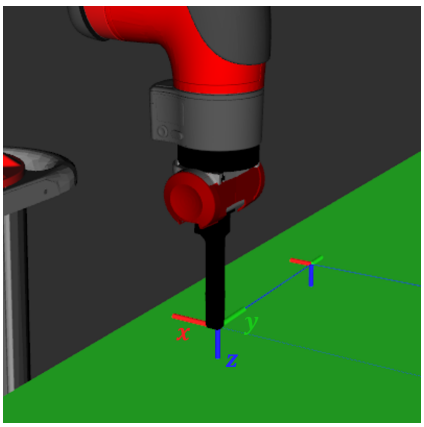


Figure 5.7: Orientation constraint for the Sawyer's end-effector. The axis colours on the tip of the gripper red, green and blue, corresponds to x, y and z axis, respectively.

5.2.3 Smooth Path Execution

This section presents the generated trajectory for the loaded path when the orientation of the end-effector is fixed in all the points of the desired path, as illustrated in Figure 5.6b. It presents the motion sequence for the simulated robot in Figure 5.8 and also for the real robot in Figure 5.9.

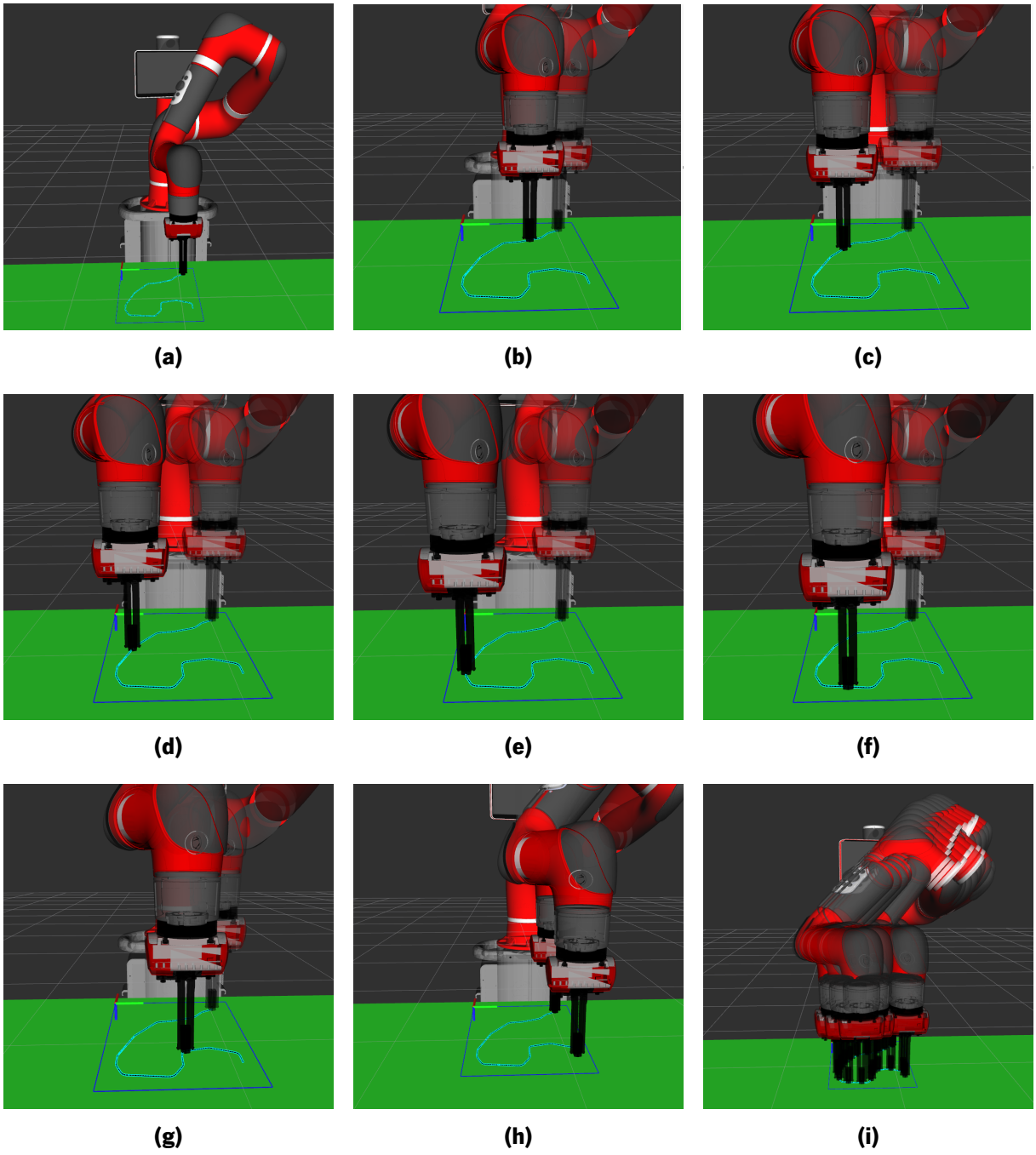


Figure 5.8: Virtual trajectory of the desired path on the PCB. From (a) to (h) shows a specific step of the trajectory, (i) represents all the steps grouped. The colour green represents the working table where the PCB mask is placed, the frame blue represents the dimensions of the PCB and inside of it corresponds to the smooth path.

[2]Complete videos of the execution of the simulation trajectory - "Smooth Path Trajectory (Virtual Scenario)" and real environment - "Smooth Path Trajectory (Sawyer Robot)" available in: https://www.youtube.com/playlist?list=PLTiZ9ch2XF1Dw8zWhBw--MWfUa7YFs-9U.

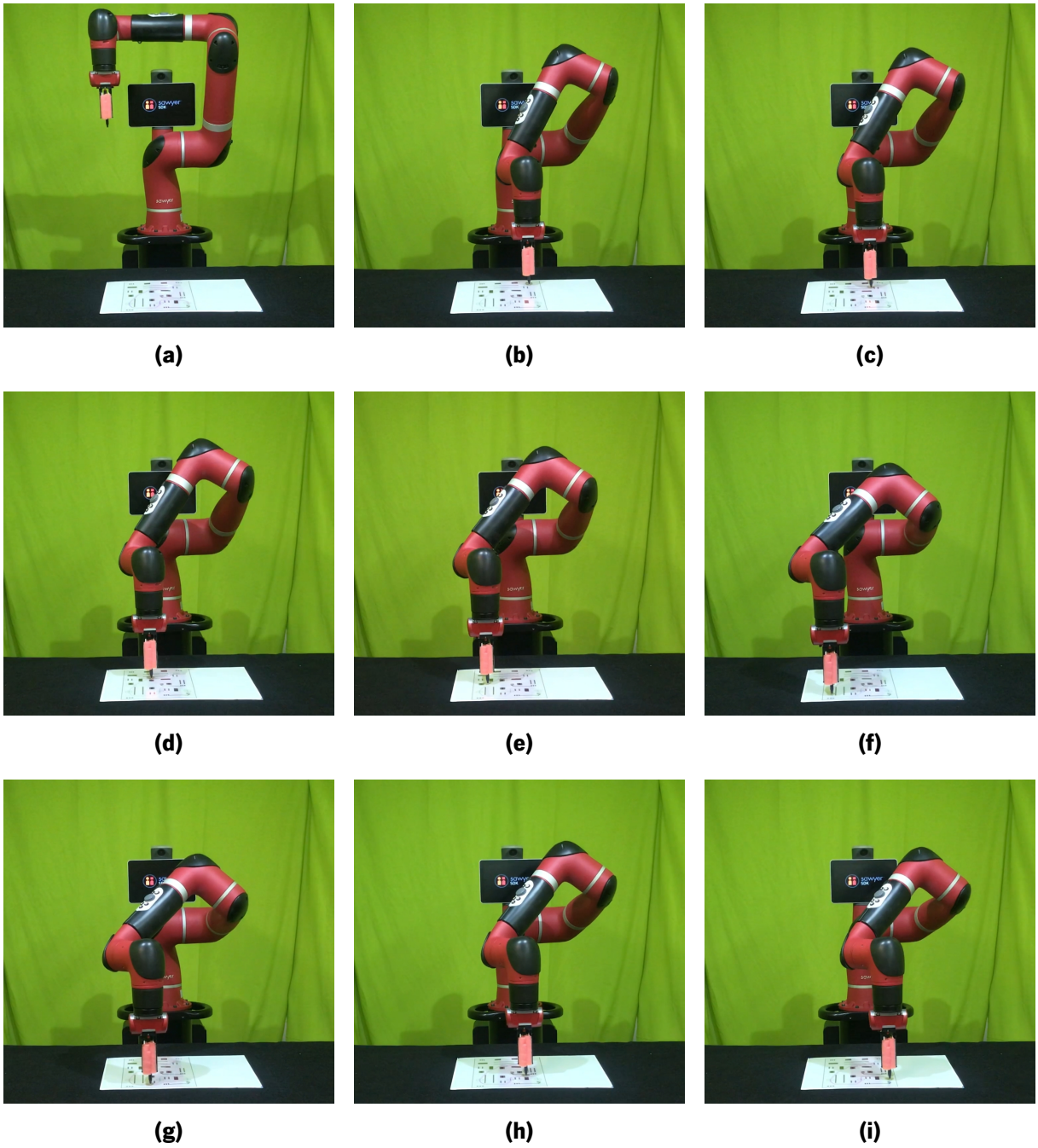


Figure 5.9: Real trajectory of the desired path on the PCB. From (a) to (i) shows a specific step of the trajectory. The colour black represents the working table, and the white area corresponds to the place where the PCB mask is placed.

The following sections will compare the results between the planned and the real ones obtained from the robot. These results will focus on (i) the Cartesian position of the end-effector, (ii) the angular position

of each joint and (iii) the linear velocity of the end-effector.

5.2.3.1 Planned vs Real Path

This section presents the results of the Cartesian position of the end-effector. The purpose is to analyse the precision of the robot along the execution of the desired path. The PCB selected for this and the following results is illustrated in Figure 5.10a, its dimensions are 200 mm x 287 mm. In each circle shown in Figure 5.10a are depicted the targets position in the centre of the circle and the linear segment selected, represented by the tick line in blue. The black colour corresponds to the obstacles present in the PCB. Alongside, Figure 5.11b, illustrates the desired path represented by the green colour and the blue one corresponds to the resulted path traced by the Sawyer robot. The red points depicts the position of each target and the green points are the beginning and end of the straight linear segment.

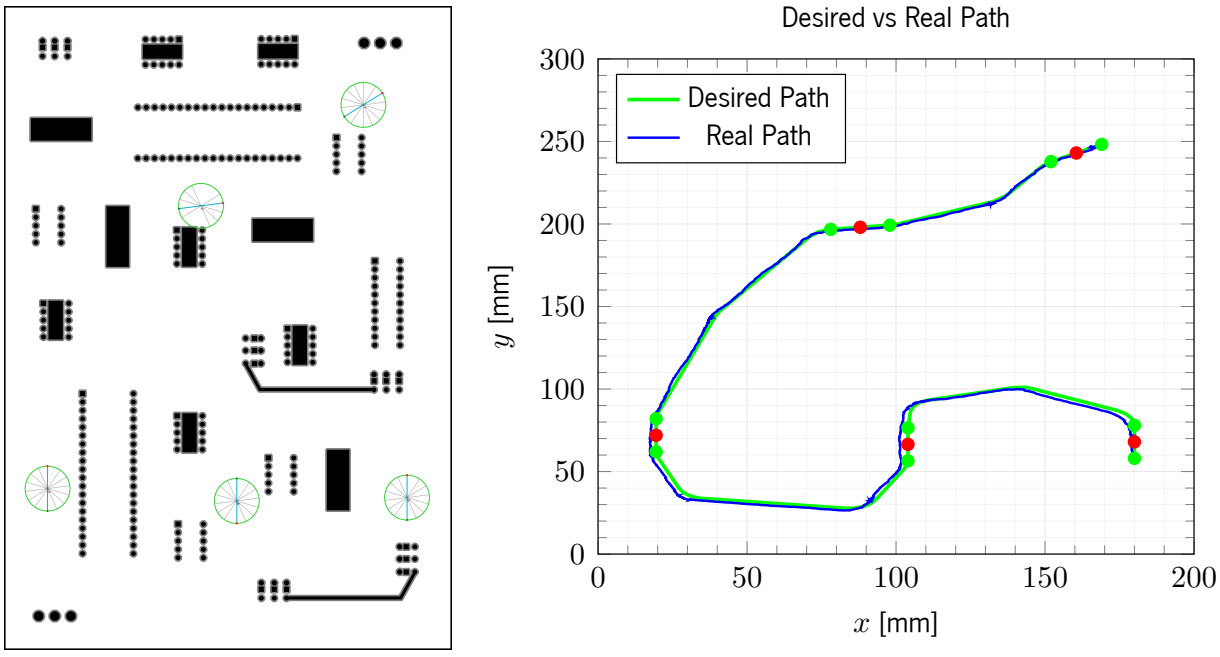


Figure 5.10: PCB and resulted paths: (a) PCB used in the tests; (b) comparison between the desired path and the real path performed by the robot. The red points correspond to the waypoints and the green points represents the beginning and end of the straight linear segment.

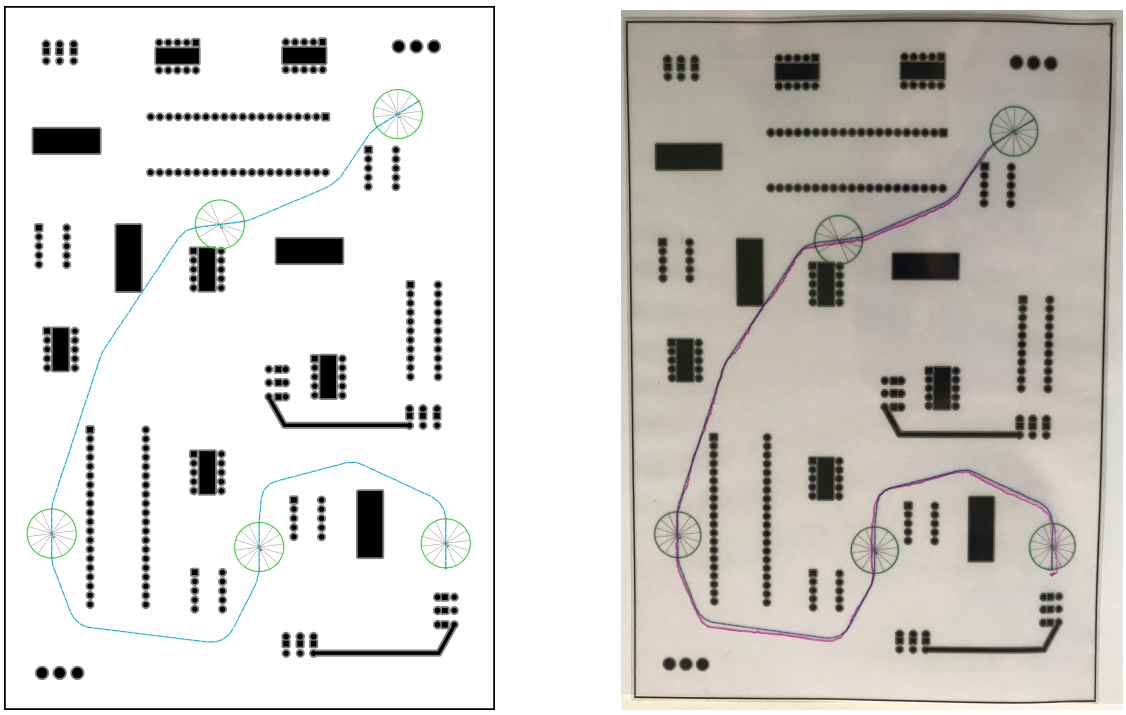
The data presented in Figure 5.11b was acquired through the subscription of two ROS topics, named "/robot/limb/right/commanded_endpoint_state" and "/robot/limb/right/endpoint_state". The first ROS

topic provides data of the desired Cartesian position of the end-effector, while the second one gives the actual Cartesian position of the end-effector. The data from these ROS topics were logged during the execution of the trajectory and the results of both are present in Figure 5.11b. The green line is the planned path generated by the Path Planner, which corresponds to the desired path. The blue line represents the Cartesian positions of the points reached by the end-effector of the real robot. The red points in the graph corresponds to the waypoints, which are the specified targets and the green points represents the beginning and end of the straight linear segment in each waypoint, in this case the segments size was 20 mm. The path traced by the robot is almost collinear with the desired one. However, there are some regions where it is possible to verify a slight difference between them. The highest deviation of the real path from the planned path was around 2 mm.

In addition to the previous results obtained by subscribing the mentioned topics, the position in which the tip of the end-effector performed along the path was also recorded. For this, a red ink pen was placed on the tip of the end-effector, which served to record the path generated by the robot.

The PCB shown in Figure 5.11a is exactly the same as shown in Figure 5.10a, with the inclusion of the desired path in blue. This PCB was used in the test as a physical reference of the position of the PCB, the obstacles and the desired path, since it was printed in real scale. After performing the routine for calibrating the PCB position, the PCB was placed in its proper place. Then, the robot performed the planned trajectory and recorded the position of the end-effector through the pen attached to the tip of the end-effector. The obtained result is shown in Figure 5.11b, which shows the path traced by the robot in red and below it is the desired path to blue, as the Figure 5.11a.

Despite the precision demonstrated by the robot, the results showed that the robot is capable of following most of the points of the desired path.



(a) (b)

Figure 5.11: Path traced by the real robot in the PCB mask. (a) PCB mask used with the desired path; (b) path traced by the real robot, represented by the red colour. In both the obstacles correspond to the black colour and the blue colour represents the desired path for the task.

5.2.3.2 Planned vs Real Trajectory

As previously said, the trajectory is composed by the angular position, the velocity and the acceleration of the joints in each step. In this section, it will be compared the desired and the real angular position of the joints. Figure 5.12 presents the obtained results, these results were acquired from a subscription of a ROS topic called "/robot/limb/right/follow_joint_trajectory/feedback", which contains the data relatively to the desired and actual angular positions of the joints. The maximum velocity and acceleration for the joints configured was 0.37 rad/s and 0.17 rad/s^2, respectively.

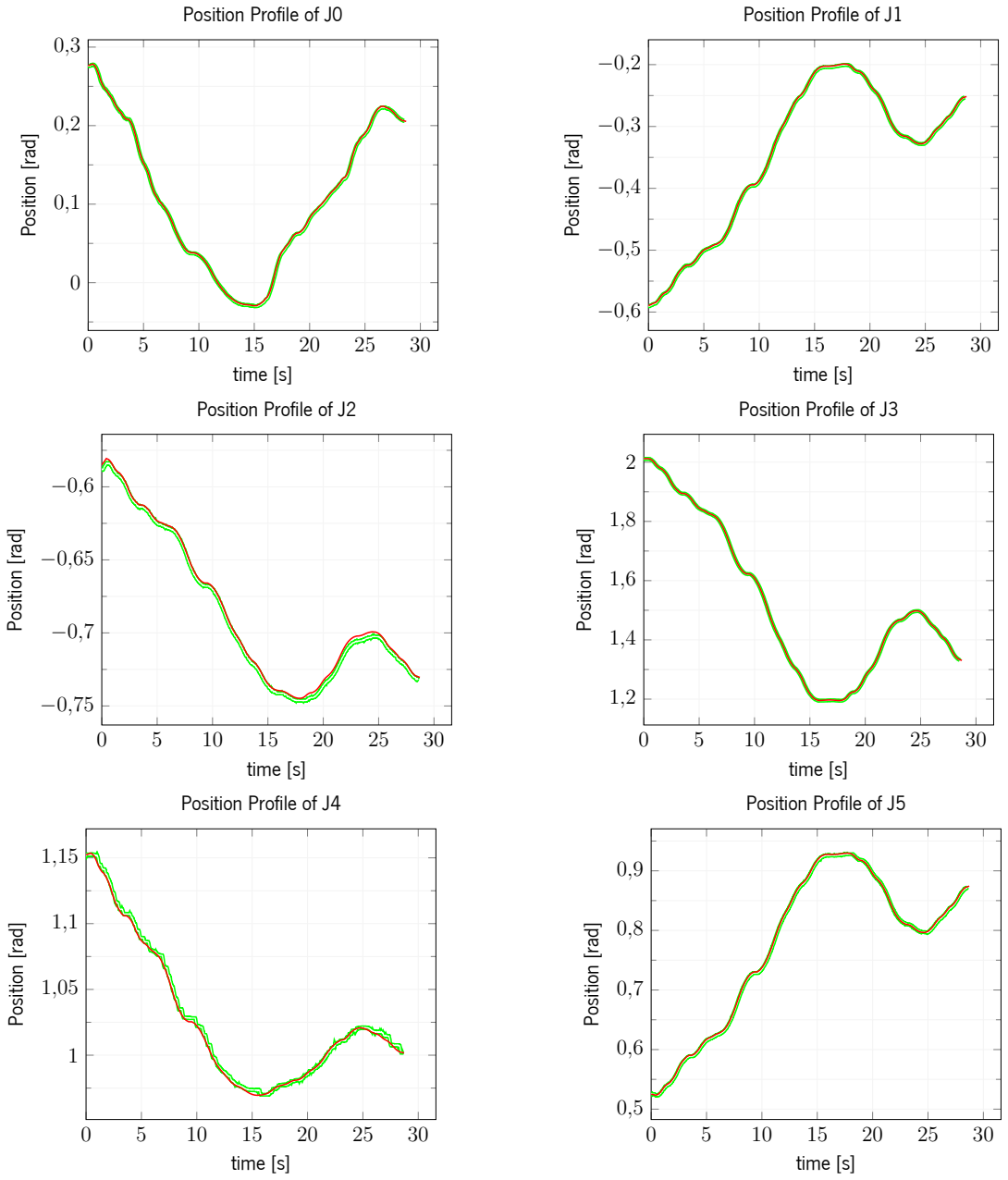


Figure 5.12: Position of each joint of the trajectory Planned trajectory w/o gripper orientation . *(Continues on the next page)*

The majority of the joints presented in the graphs of Figure 5.12 follow the desired position along the trajectory, such as the joints 0, 1, 3 and 6, while the remaining joints 2, 4 and 5 show a variation from the desired position of the joint. These difference influences the position of the end-effector, which causes the small difference in the final path presented previously.

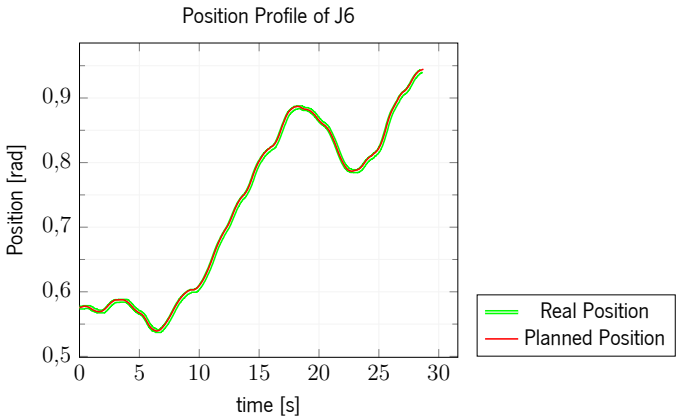


Figure 5.12: Position of each joint of the trajectory Planned trajectory w/o gripper orientation .

5.2.3.3 Planned vs Real End-effector Velocity

The velocity profile of the end-effector was also analysed and compared to the actual values from the real robot. In order to calculate the velocity profile, we resorted to the differential kinematics. The goal of differential kinematics is to find the relationship between the joint velocities ($\dot{q}$) and the end-effector linear ($\dot{pe}$) and angular (we) velocities. This mapping is described by a matrix, termed geometric Jacobian (J), which depends on the manipulator configuration. In other words, it is desired to express the end-effector linear velocity ($\dot{pe}$) and angular velocity (we) as a function of the joint velocities ($\dot{q}$).

$$\begin{aligned} \dot{pe} &= J_P(q) \cdot \dot{q} \\ we &= J_O(q) \cdot \dot{q} \end{aligned} \tag{5.5}$$

where $q = [q_1, q_2, \ldots, q_n]^T$ represents the vector of joints variables. The value n corresponds to the number of joints of the manipulator. J_P and J_O are two sub matrices of Jacobian matrix (J), that provide the relation between each joint velocity and the end-effector linear (J_P) and angular velocity (J_O), respectively. In a compact form, Equation (5.5) can be written as

$$v_e = \begin{bmatrix} \dot{pe} \\ we \end{bmatrix} = J(q) \cdot \dot{q} \tag{5.6}$$

where $J(q) = [J_P \; J_O]^T$ is the Jacobian matrix of the manipulator.

Figure 5.13 illustrates the evolution of the velocity of the end-effector during the execution of the planned trajectory. The line in blue represents the expected linear velocity and the line in green corresponds to the linear velocity of the end-effector. The data of the linear velocity was acquired through the

ROS topic named "/robot/limb/right/endpoint_state", while the expected values were calculated using the differential kinematics presented above.

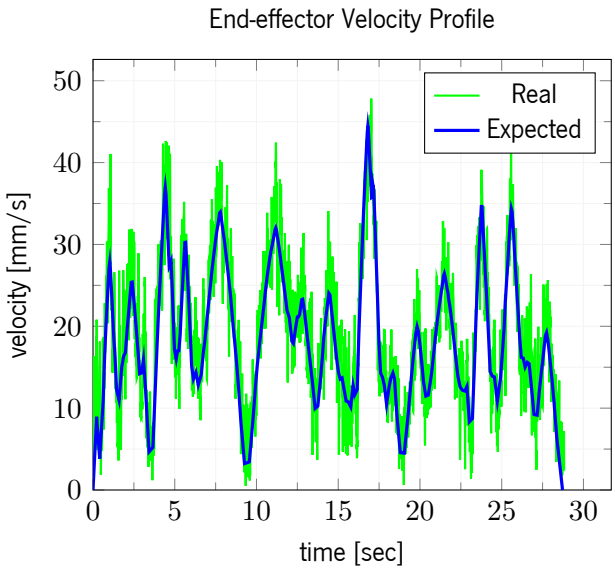


Figure 5.13: End-effector velocity profile.

The velocity profile of the robot followed the expected values, as shown in Figure 5.13, the maximum value achieved was 46 mm/s. As already mentioned, the focus of this project was not on trajectory generation, but in the path generation of the specified task. The velocity profile of the end-effector could be improved as future work, resulting on smoother velocity profiles performed by the end-effector.

5.3 Discussion

This chapter detailed the implementation and presented the results of the validation of the Motion Planner module. The module has proven its capabilities in generating feasible trajectories for the robotic platform Sawyer. In order to plan and execute the trajectories, an application was developed to manage the planing and goal requests through the framework ROS for both scenarios – virtual and real scenario.

First of all, the PCB and the desired path are selected. Then, a routine is performed to specify the exact position where the PCB should be placed. After that, the main trajectory is generated and executed, which corresponds to the trajectory that have to follow the points of the desired path. The results presented in Section 5.2.3 are focused on the main trajectory, where it is compared the data from the planned trajectory

with the trajectory performed by the real robot. These results were analysed and compared regarding to (i) the angular position of each joint, (ii) the Cartesian position of the end-effector and (iii) the linear velocity of the end-effector.

In general, the results obtained were very close to what was expected. Regarding the angular position of the joints, excluding three joints, they were able to follow the expected values. However, the three joints that did not follow the position profile with the same accuracy as the others, influenced the final position of the end-effector, causing a small deviation in position relative to the desired position. This problem directly affected the accuracy at the tip of the manipulator end-effector during the execution of the trajectory, reaching sometimes a maximum deviation of 2 mm. As for the linear velocity, the tests allowed to evaluate the velocity profile of the end-effector, that is, the speed with which the end-effector followed the desired path. The behaviour performed by the robot was very close to what was expected.

In addition to the tests presented in this chapter, preliminary tests were performed in which the orientation of the end-effector had to follow the orientation of each point of the path (as shown in Figure 5.6a). The trajectories were successfully generated and performed by the virtual robot, where it showed that the orientation of the end-effector followed the heading profile of the path. However, when the same trajectory was executed by the real robot, its behaviour differed from what was performed by the virtual robot. The final path traced by the robot had an increased error than desired, particularly in some sections of the path, where the robot performed unexpected paths. The origins of this problem should be related to the motion controllers of the robot and, thus, is out of the scope of this dissertation. It should be further investigated in future work.

Chapter 6

Conclusion

The last chapter presents a brief description of the developed solution, discusses the main conclusions and results obtained in this dissertation and suggests some topics that can be implemented as future work.

6.1 Summary and Discussion

Currently, most industrial activities have been searching for robotic solutions to automate their processes and/or tasks, through the settlement of robotic systems in their infrastructures. This approach provides various improvements in their processes, such as (i) the increase of productivity; (ii) the improvement of the product quality; (iii) the reduction of fabrication costs; and (iv) the replacement of heavy and exhausting tasks for the human operator. Most of these processes are well defined and structured, in which they can be completely automated without any operator intervention, or it can be a combination of the human operator and the robot's abilities, as the case of collaborative robotics.

The task for this case of study consist on the placement of a fibre optic filament in a Printed Circuit Board (PCB) by a robotic platform. This task can be divided in two sub processes, namely (i) the planning of the path in which the fibre optic filament must be deposited and (ii) the planning of the trajectory for the Sawyer robotic system. The main focus of this dissertation project is the first sub process, where a path should be created accommodating all the specified constraints. The developed path planning algorithm – presented in Chapter 4 – covers all the specified constraints, such as (i) multiple targets for the same

path; (ii) straight linear segments on each target and (iii) minimum turning radius. Additionally, a graphical user interface has been developed for the Path Planner module, which allows the user to insert, adjust and modify a wide range of tunable parameters and also the PCB mask. To point out, that the Path Planner outputs a geometric path according to the environment model and the specified constraints, and thus it can also be used for solving other problems with similar constraints, such as mobile robots or other robotic arms.

The method used to represent the environment was the cell decomposition method due to its simplicity of implementation and easy map updating in case of changes in the environment. This method divides the environment into cells of the same size. In order to find the adjacent cells between the initial position and the next target, four pathfinding algorithms were implemented, namely, A*, A*3D, Theta* and Theta*LA. The algorithm A* with some improvements (wA*+) along with the euclidean distance as heuristic cost to estimate the cost to the final cell, it was selected to generate the first path between the targets. Then, the Pruning Method is applied to simplify the path, by eliminating unnecessary nodes. After that, the path is smoothed, by accommodating the turning radius specified through the integration of the Dubins curves.

The validation of the Path Planner – Section 4.5 – was performed through a series of tests that allowed to characterize its behaviour according to the adjustment of a set of parameters. It was verified that adjusting the scale factors of the heuristic method it can improve the time spent and the success rate of the algorithm. The Safety Distance (SD) showed to be an important parameter, since it ensures that whatever the size of the path, never collide with the obstacles present in the environment. There are also the Penalty Radius (PR) and Penalty Weight (PW) parameters, part of the Map Weighing method, that demonstrated to be an improvement on the planner's performance and to move away the path from the obstacles. Regarding to the turning radius, the results performed showed that the final path validated all the three values set as possible turning radius, and that the final path is a combination of Dubins curves with the most suitable turning radius in each section.

Furthermore, the Motion Planner module introduced in Chapter 5, was developed to generate the trajectories for the robotic platform Sawyer. This module manages the exchange of messages through the ROS framework for the virtual and real robot. The generated trajectories are validated in the virtual scenario, ensuring the collision avoidance with the obstacles present in the scenario and also with the robot's own structure, but also avoiding singularities through a smoother motion. The trajectory is created

based on the desired path created by the Path Planner.

The motion sequence and the results performed in Section 5.2 have shown that the trajectories were successfully performed in the virtual scenario as well as in the real robot. However, in the real robot, the trajectory generated for the Cartesian path showed less accuracy at the tip of the end-effector, reaching sometimes a maximum deviation of 2 mm.

Overall, taking into account all tests performed – both in virtual scenario as well as in real environment – we can affirm that we fulfil the objectives proposed for this dissertation. The developed project integrates the two modules Path Planner and Motion Planner. They provide the necessary tools in order to create a smooth path within the imposed constraints and also to generate feasible and smoother trajectories for the robotic arm Sawyer.

6.2 Future Work

In the scope of the developed project and considering the difficulties detected, such as the curvature discontinuity of the Dubins curves, the robot's accuracy and the velocity profile of the end-effector, it is proposed as future work:

- Modify the smoothing planning method based on the Dubins curve, adding a range of possible values for the radius of curvature, instead of only three values;
- Develop a smoothing planning algorithm based on the Bezier curves, so that the curvature profile will be continuous at the junction of each segment ;
- Add a new time parametrization algorithm for the generated trajectories of the Sawyer robot, namely the Iterative Spline Parametrization;
- Implement an algorithm to control the velocity of the end-effector.

References

[1] D. R. Franceschetti, Ed., *Principles of Robotics and Artificial Intelligence*, 1st ed. Grey House Publishing, 2018, p. 415.

[2] M. Hägele, K. Nilsson, J. N. Pires, and R. Bischoff, "Industrial Robotics", in *Springer Handbook of Robotics*, B. Siciliano and O. Khatib, Eds., 2nd ed., Springer, 2016, ch. 54, pp. 1385–1422.

[3] R. H. Taylor, A. Menciassi, G. Fichtinger, P. Fiorini, and P. Dario, "Medical Robotics and Computer-Integrated Surgery", in *Springer Handbook of Robotics*, Cham: Springer International Publishing, 2016, pp. 1657–1684.

[4] H. M. Van der Loos, D. J. Reinkensmeyer, and E. Guglielmelli, "Rehabilitation and Health Care Robotics", in *Springer Handbook of Robotics*, Cham: Springer International Publishing, 2016, pp. 1685–1728.

[5] K. Yoshida, B. Wilcox, G. Hirzinger, and R. Lampariello, "Space Robotics", in *Springer Handbook of Robotics*, Cham: Springer International Publishing, 2016, pp. 1423–1462.

[6] G. Antonelli, T. I. Fossen, and D. R. Yoerger, "Underwater Robotics", in *Springer Handbook of Robotics*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 987–1008.

[7] R. R. Murphy, S. Tadokoro, D. Nardi, A. Jacoff, P. Fiorini, H. Choset, and A. M. Erkmen, "Search and Rescue Robotics", in *Springer Handbook of Robotics*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 1151–1173.

[8] J. Trevelyan, W. R. Hamel, and S.-C. Kang, "Robotics in Hazardous Applications", in *Springer Handbook of Robotics*, Cham: Springer International Publishing, 2016, pp. 1521–1548.

[9] R. R. Murphy, S. Tadokoro, and A. Kleiner, "Disaster Robotics", in *Springer Handbook of Robotics*, Cham: Springer International Publishing, 2016, pp. 1577–1604.

[10] M. Bergerman, J. Billingsley, J. Reid, and E. van Henten, "Robotics in Agriculture and Forestry", in *Springer Handbook of Robotics*, Cham: Springer International Publishing, 2016, pp. 1463–1492.

[11] K. S. Saidi, T. Bock, and C. Georgoulas, "Robotics in Construction", in *Springer Handbook of Robotics*, Cham: Springer International Publishing, 2016, pp. 1493–1520.

[12] E. Prassler, M. E. Munich, P. Pirjanian, and K. Kosuge, "Domestic Robotics", in *Springer Handbook of Robotics*, Cham: Springer International Publishing, 2016, pp. 1729–1758.

[13] M. Hägele, K. Nilsson, J. N. Pires, and R. Bischoff, "Industrial Robotics", in *Springer Handbook of Robotics*, B. Siciliano and O. Khatib, Eds., Cham: Springer International Publishing, 2016, pp. 1385–1422.

[14] W. Robotics, "Executive Summary World Robotics 2018 Industrial Robots", *World Robotic Report - Executive Summary*, pp. 13–22, 2018.

[15] G. Michalos, S. Makris, J. Spiliotopoulos, I. Misios, P. Tsarouchi, and G. Chryssolouris, "ROBO-PARTNER: Seamless Human-Robot Cooperation for Intelligent, Flexible and Safe Operations in the Assembly Factories of the Future", *Procedia CIRP*, vol. 23, no. C, pp. 71–76, 2014.

[16] R. Bloss, “Collaborative robots are rapidly providing major improvements in productivity, safety, programing ease, portability and cost while addressing many new applications”, *Industrial Robot: An International Journal*, vol. 43, no. 5, pp. 463–468, 2016.

[17] J. Iqbal, R. U. Islam, S. Z. Abbas, A. A. Khan, and S. A. Ajwad, “Automating industrial tasks through mechatronic systems – a review of robotics in industrial perspective”, *Tehnicki vjesnik - Technical Gazette*, vol. 23, no. 3, pp. 917–924, 2016.

[18] J. Pires, A. Loureiro, T. Godinho, P. Ferreira, B. Fernando, and J. Morgado, “Welding Robots”, *IEEE Robotics & Automation Magazine*, vol. 10, no. 2, pp. 45–55, 2003.

[19] Z. Liu, W. Bu, and J. Tan, “Motion navigation for arc welding robots based on feature mapping in a simulation environment”, *Robotics and Computer-Integrated Manufacturing*, vol. 26, no. 2, pp. 137–144, 2010.

[20] H. Chen, F. Thomas, and L. Xiongzi, “Automated industrial robot path planning for spray painting a process: A review”, *4th IEEE Conference on Automation Science and Engineering, CASE 2008*, pp. 522–527, 2008.

[21] W. Chen and D. Zhao, “Path Planning for Spray Painting Robot of Workpiece Surfaces”, *Mathematical Problems in Engineering*, vol. 2013, pp. 1–6, 2013.

[22] G. Trigatti, P. Boscariol, L. Scalera, D. Pillan, and A. Gasparetto, “A new path-constrained trajectory planning strategy for spray painting robots - rev.1”, *International Journal of Advanced Manufacturing Technology*, vol. 98, no. 9-12, pp. 2287–2296, 2018.

[23] H. Choset, K. M. Lynch, S. Hutchinson, G. a. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations*, M. Press, Ed., C. Cambridge, 2005, p. 603.

[24] S. M. Lavalle, *Planning Algorithms*. Cambridge University Press, 2006.

[25] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics - Modelling, Planning and Control*, ser. Advanced Textbooks in Control and Signal Processing. Springer London, 2009.

[26] Y. K. Hwang and N. Ahuja, “Gross Motion Planning – a Survey”, *ACM Comput. Surv.*, vol. 24, no. 3, pp. 219–291, 1992.

[27] J.-C. Latombe, *Robot Motion Planning*. Kluwer Academic Publishers, 1991.

[28] A. Gasparetto, P. Boscariol, A. Lanzutti, and R. Vidoni, “Path Planning and Trajectory Planning Algorithms: A General Overview”, in *Motion and Operation Planning of Robotic Systems: Background and Practical Approaches*, G. Carbone and F. Gomez-Barvo, Eds., Springer International Publishing, 2015, pp. 3–27.

[29] O. Khatib, “Real-Time Obstacle Avoidance for Manipulators and Mobile Robots”, *The International Journal of Robotics Research*, vol. 5, pp. 90–98, 1986.

[30] L. E. Kavraki and S. M. LaValle, “Motion Planning”, in *Springer Handbook of Robotics*, B. Siciliano and O. Khatib, Eds., 2nd ed., Springer, 2016, ch. 7, pp. 139–162.

[31] T. Tsuji, P. G. Morasso, and M. Kaneko, “Trajectory Generation for Manipulators Based on Artificial Potential Field Approach with Adjustable Temporal Behavior”, in *Intelligent Robots and Systems' 96, Proceedings of the 1996 IEEE/RSJ International Conference on*, IEEE, vol. 2, 1996, pp. 438–443.

[32] P. Hart, N. Nilsson, and B. Raphael, “A Formal Basis for the Heuristic Determination of Minimum Cost Paths”, *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[33] O. Souissi, R. Benatitallah, D. Duvivier, A. Artiba, N. Belanger, and P. Feyzeau, "Path Planning : A 2013 Survey", *International Conference on Industrial Engineering and Systems Management*, vol. 2013, pp. 1–8, 2013.

[34] L. Steven M., "Rapidly-Exploring Random Trees: A New Tool for Path Planning", 1998.

[35] G. Kang, Y. B. Kim, W. S. You, Y. H. Lee, H. S. Oh, H. Moon, and H. R. Choi, "Sampling-based Path Planning with Goal Oriented Sampling", in *IEEE International Conference on Advanced Intelligent Mechatronics*, 2016, pp. 1285–1290.

[36] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces", *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[37] D. E. Knuth, *The Art of Computer Programming, Volume III: Sorting and Searching*, 1973.

[38] E. Dijikstra, "A note on two problems in connexion with graphs", *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[39] L. E. Dubins, "On Curves of Minimal Length with a Constraint on Average Curvature, and with Prescribed Initial and Terminal Positions and Tangents", *American Journal of Mathematics*, vol. 79, no. 3, pp. 497–516, 1957.

[40] Guang Yang and V. Kapila, "Optimal path planning for unmanned air vehicles with kinematic and tactical constraints", in *Proceedings of the 41st IEEE Conference on Decision and Control, 2002.*, vol. 2, 2002, pp. 1301–1306.

[41] E. P. Anderson, R. W. Beard, and T. W. McLain, "Real-time dynamic trajectory smoothing for unmanned air vehicles", *IEEE Transactions on Control Systems Technology*, vol. 13, no. 3, pp. 471–477, 2005.

[42] G. Parlangeli and G. Indiveri, "Dubins inspired 2D smooth paths with bounded curvature and curvature derivative.", *IFAC Proceedings Volumes*, vol. 43, no. 16, pp. 252–257, 2010.

[43] K. Yang and S. Sukkarieh, "An Analytical Continuous-Curvature Path-Smoothing Algorithm", *IEEE Transactions on Robotics*, vol. 26, no. 3, pp. 561–568, 2010.

[44] Kwangjin Yang and S. Sukkarieh, "3D Smooth Path Planning for a UAV in Cluttered Natural Environments", in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2008, pp. 794–800.

[45] K. Yang and S. Sukkarieh, "Planning Continuous Curvature Paths for UAVs Amongst Obstacles", in *Australasian Conference on Robotics Automation, Canberra, Australia*, 2008.

[46] S. M. LaValle and J. J. Kuffner Jr, "Rapidly-exploring random trees: Progress and prospects", 2000.

[47] P. Shirley, *Fundamentals of Computer Graphics*, 2nd ed. AK Peters, 2005.

[48] A. Nash and S. Koenig, "Theta* for Any-Angle Pathfinding", in *Game AI Pro 2: Collected Wisdom of Game AI Professionals*, S. Rabin, Ed., CRC Press, 2015, ch. 16, pp. 161–171.

[49] K. Yakovlev, E. Baskin, and I. Hramoin, "Grid-Based Angle-Constrained Path Planning Konstantin", in *KI 2015: Advances in Artificial Intelligence*, S. Hölldobler, M. Krötzsch, R. Peñaloza, and S. Rudolph, Eds., Springer, 2015, pp. 208–221.

[50] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Mg, "ROS: an open-source Robot Operating System", in *ICRA Workshop on Open Source Software*, 2009.

[51] M. Quigley, B. Gerkey, and W. D. Smart, *Programming Robots with ROS: a practical introduction to the Robot Operating System*. " O'Reilly Media, Inc.", 2015.

References

[52] J. E. Bresenham, “Algorithm for computer control of a digital plotter”, *IBM Systems journal*, vol. 4, no. 1, pp. 25–30, 1965.

Appendix A

Graphical User Interfaces

The following sections present an overview and enumerates the features provided by developed graphical user interfaces (GUIs). These GUIs were created through the development tool Qt Creator and implemented in C++ language.

A.1 Path Planner

The developed GUI for the Path Planner is illustrated in Figure A.1, where it shows a PCB mask with three planned paths (search, prun and smooth). The main purpose of this application was to provide an easier way and a friendly environment to create the smooth path, to configure the parameters and to see the results. The main features are listed bellow:

- Select and load a PCB mask;
- Define the resolution of the cells grid/ map resolution;
- Load and/ or add the waypoints coordinates;
- Define the length of the straight linear segment of each waypoint;
- Change the orientation of each straight linear segment;
- Define the maximum, preferable and minimum turning radius;
- Choose the pathfinding algorithm, change its parameters and select the heuristic method to be used in the searching phase;
- Plan and save the results;

- Select what it will be shown in the PCB mask, such as the planned paths, the expanded nodes, the obstacles and the grid, for example.

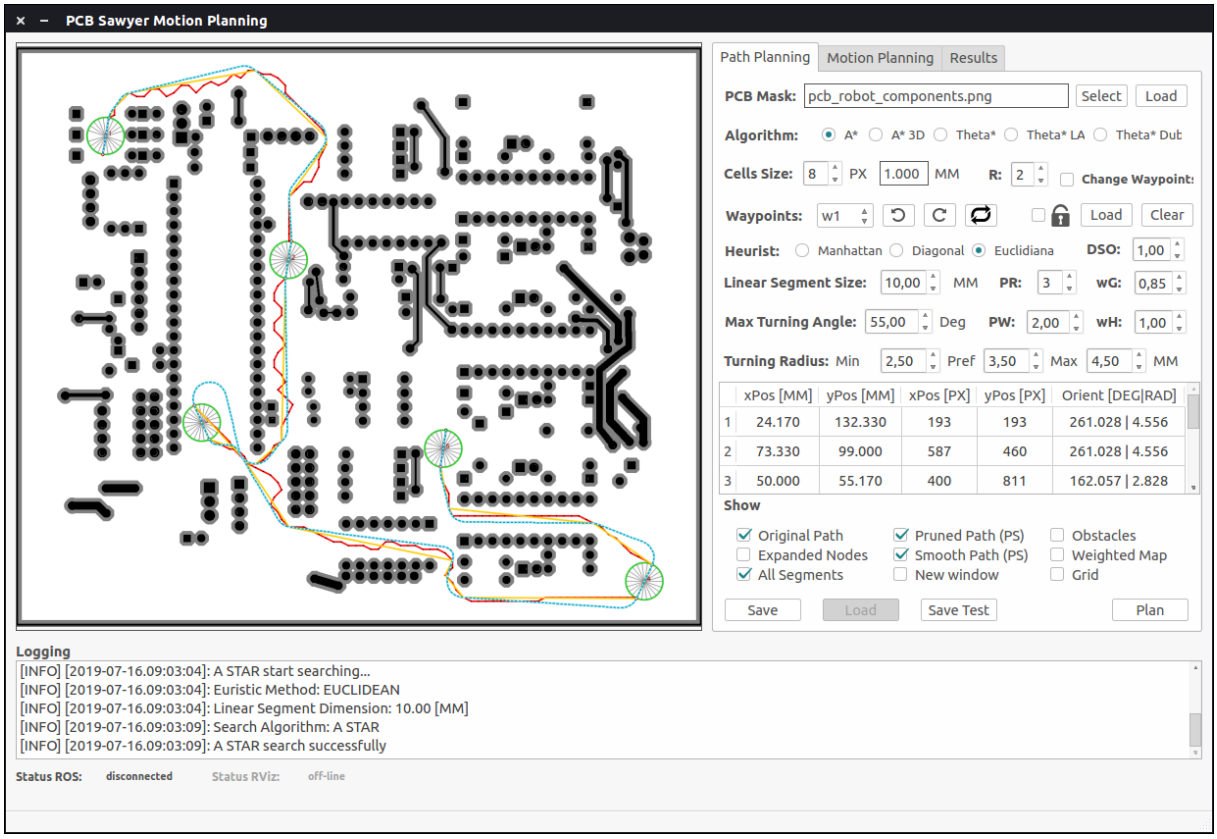


Figure A.1: Graphical Interface of the Path Planner.

The results can be saved and they contain all information about the PCB mask, the values of the parameters, the number of expanded nodes and the consumed time, path length in each phase. Most of these results are shown in the application Figure A.2.

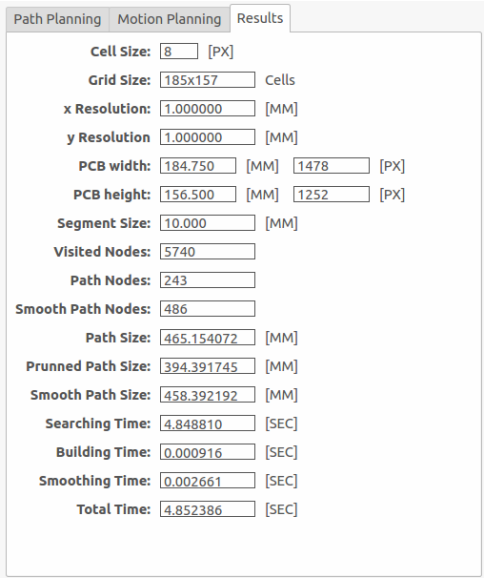


Figure A.2: Results presented in the Path Planner GUI.

A.2 Motion Planner

The Motion Planner GUI Figure A.3 was developed to manage the communication with the robot, the services provided by MoveIt and to send information to the RViz. The main features of the developed application are:

- Establish the communication with the ROS server and the RViz;
- Define the origin of the position of the PCB;
- Load a smooth path (path generated by the Path Planner);
- Select the planner and a set of parameters (number of planning attempts, planning time);
- Choose if it is the virtual or the real robot and load it;
- Plan, execute and save the trajectories.

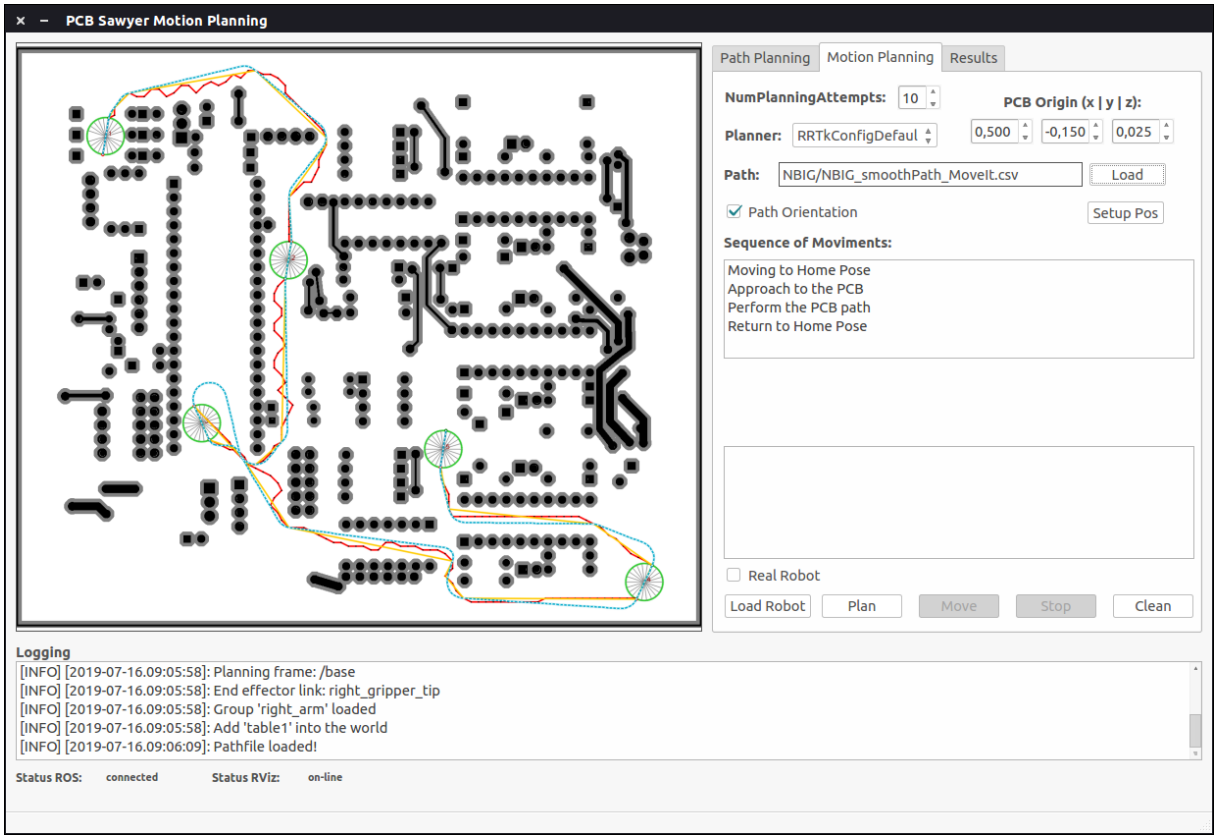


Figure A.3: Graphical Interface of the Motion Planner.