FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Smart collision avoidance system for a dual-arm manipulator

Inês Pinto Frutuoso

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Pedro Luís Cerqueira Gomes da Costa

Co-Supervisor: José Luís Magalhães Lima

Supervisor INESC TEC: Luís Freitas Rocha

July 19, 2018

Resumo

O crescente aumento do interesse e importância da robótica implica um avolumar de desafios e problemas que têm de ser resolvidos. Do particular interesse por robôs humanoides advém a necessidade de lidar com manipuladores de dois braços, os quais têm de ser capazes de executar as tarefas pretendidas sem colidir um com o outro ou com qualquer objeto à sua volta. É este o propósito desta tese.

O desvelo em evitar colisões é inato a qualquer ser humano, no entanto, tal não acontece para seres artificiais, nestes esta característica é conseguida recorrendo a algoritmos de planeamento de trajetória. Estes permitem a um robot, ou a um manipulador, chegar ao destino sem colidir com nenhum obstáculo ou com ele mesmo.

O problema do planeamento de trajetórias tem sido exaustivamente estudado e várias soluções já foram proposta, todavia, nenhuma destas foi especificamente desenvolvida para um manipulador de dois braços. Ao longo deste projeto seis planeadores são escolhidos: RRT, RRT-Conect, KPIECE, PRM, PRM* e EST. Todos eles são explicados e testados num manipulador de dois braços, composto por dois braços UR5, com 6 graus de liberdade cada. Inicialmente os planeadores são aplicados num ambiente sem quaisquer objetos e somente depois em situações simuladas de operações de pick e place.

Os planeadores são comparados tendo em vista a viabilidade de aplicação a manipuladores de dois braços, especialmente quando manipulam objetos. Este projeto não só apresenta uma solução que assegura um caminho sem colisões para um manipulador de dois braços, mas é também, segundo o nosso conhecimento, pioneiro na medida em que afere o comportamento de diferentes planeadores, alguns dos quais não são comummente utilizados para este propósito.

Abstract

The growth in the interest and importance of the robotic field entails a greater number of challenges and problems that must be addressed. The particular passion and utility of humanoid robots lead to the need to handle dual arm manipulators, these must be able to perform the assigned tasks without colliding with each other or with any object in their surroundings. This is the purpose of this thesis.

The concept of collision avoidance for a human is inherently natural, however for an artificial being this is achieved through the usage of motion planning algorithms. These allow a robot, or a manipulator, to achieve the desired goal without hitting any obstacle or other parts of itself.

The motion planning problem has been widely tackled and several different solutions have been proposed, however none of these are specially designed to handle dual arm manipulators. Throughout this project six different planners are chosen: RRT, RRT-Connect, KPIECE, PRM, PRM* and EST. All of them are explained and then tested on a dual arm manipulator, composed of two UR5 arms, with 6 degrees of freedom each. On a first instance the planners are applied in an object free environment and only then in different simulated real life scenarios of picking and placing tasks.

The planners are compared to evaluate which of them are able to handle dual arm manipulators, specially when dealing with objects. This project not only presents a solution that ensures a collision free path on a dual arm manipulator but, to our knowledge, is pioneer in the sense that assess the behaviour of different planners, some of which are not commonly used for this specific purpose.

Acknowledgements

First of all I would like to thank my supervisors Pedro Costa, José Lima and Luís Rocha for all the time spent with me, for all the patience and help whenever requested. To them I owe a big thank you!

Second, I thank you, my mother, for everything you have done for me, I thank all the time you have invested in me, all the caring and all the sacrifices, believe it or not, it did not go unnoticed. To you, my father, I thank you for all you have taught me, for all the times I had to go get the flip flop and for every 3 questions ever made, I can only wish to ever know a tiny bit of everything you do. To you, my beloved parents, I say the biggest thank you I will ever say to anyone! In that big thank you I cannot exclude having given me a sister... To you, Luísa, I say thank you for so many little things and for so many big ones. I could have written a whole thesis on how many things I can thank you for.

To you seven, I can think of a thousand different reasons to thank each and everyone of you, strangely enough you are probably the ones I feel less need to do so.

Next, I would like to thank the ones that by being by my side have become a part of me. I thank you, my old friends, for keeping alive a part of me that otherwise would have been long gone. And I thank you, my not so old friend, for each day being there, because sometimes that is more than enough.

And since we must save the best for last, I thank you, João Ricardo, that through night shifts, movies, cookies, ice cream and pizzas have become the best part of myself.

Inês Frutuoso

"I may not have gone where I intended to go,
but I think I have ended up where I needed to be"

Douglas Adams, *The Long Dark Tea-Time of the Soul*

Contents

List of Figures

List of Tables

Abbreviations

DoF	Degree of Freedom
IK	Inverse Kinematics
FK	Forward Kinematics
DH	Denavit–Hartenberg
PRM	Probabilistic Roadmap Method
RRT	Rapidly-exploring Random Trees
CT	Configuration-Time
ROS	Robotic Operating System
OMPL	Open Motion Planning Library
RRT-Connect	Rapidly-exploring Random Trees-Connect
KPIECE	Kinematic Planning by Interior-Exterior Cell Exploration
EST	Expansive Space Trees

Chapter 1

Introduction

Robots have an ever increasing importance in today's society and in our daily life. Industry wise, dual arm robots play a central role since they are designed to behave in a similar manner to a human being and present a more versatile set of behaviours and tasks than a single arm robot. This thesis aims to operate in an anthropomorphic dual arm robot, in order to start we will do a first approach on what is an anthropomorphic dual arm robot and its schematics.

1.1 Anthropomorphic Dual Arm Robot

An anthropomorphic robot, as the name indicates, is a robot that somehow mimics a certain human condition, this means that an anthropomorphic dual arm robot is a robot that intends to replicate human arms and their functions.

To assemble any kind of manipulator one needs to use both joints and links; joints are responsible for the degrees of freedom (DoFs) - the movements - of the robot and links are the parts that keep the joints attached. Robotic joints are divided in manly two categories: prismatic and revolute (there are more but these are the most common and relevant for this work). As depicted in Fig. 1.1, prismatic joints are responsible for linear movements and revolute joints for rotational movements.

An anthropomorphic manipulator, as stated before, aims to replicate the movements of a human arm, therefor it has to have at least three revolute joints, as is represented in Fig. 1.2 - there are also some anthropomorphic manipulators that in addition have three prismatic joints, one in between each of the revolute joints. The manipulator is fixed in a base and attached to it has a rotational joint that rotates along an axis z_0, this axis goes through another revolute joint that rotates around z_1 and has parallel to it another joint that rotates along z_2. An additional revolute joint is added in order to replicate the rotation of the wrist. A dual arm anthropomorphic manipulator, in addition to two anthropomorphic manipulators, has also a rigid body - that can take several forms, from a simple cylinder to an elaborated torso - that connects the two arms. The details and schematics of the manipulator used throughout this project will be presented in Chapter 2.

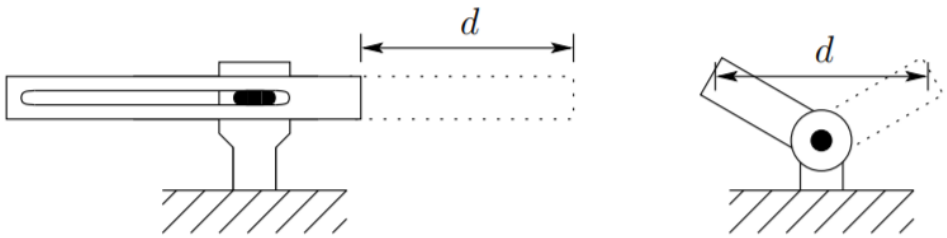


Figure 1.1: Prismatic (Left) and Revolute (Right) Joints [1]

1.2 Problem Overview

The problem we aim to tackle is simple to understand, if there are two robots working in the same workbench and on the same piece what might happen is, when the arms are moving to the defined working position, they may bump into each other. It is obvious that this entails a series of problems, not only will the arms not reach the desired positions, and therefor not perform the wanted task, they can be damaged in the process. It is then in paramount importance that a solution or a method to avoid this presents itself.

Collision avoidance was born through the necessity of addressing this problem. In the present day collision avoidance is already implemented in many devices (one obvious example is its implementation in automated vehicles) and it is an absolute indispensable feature when it comes to security.

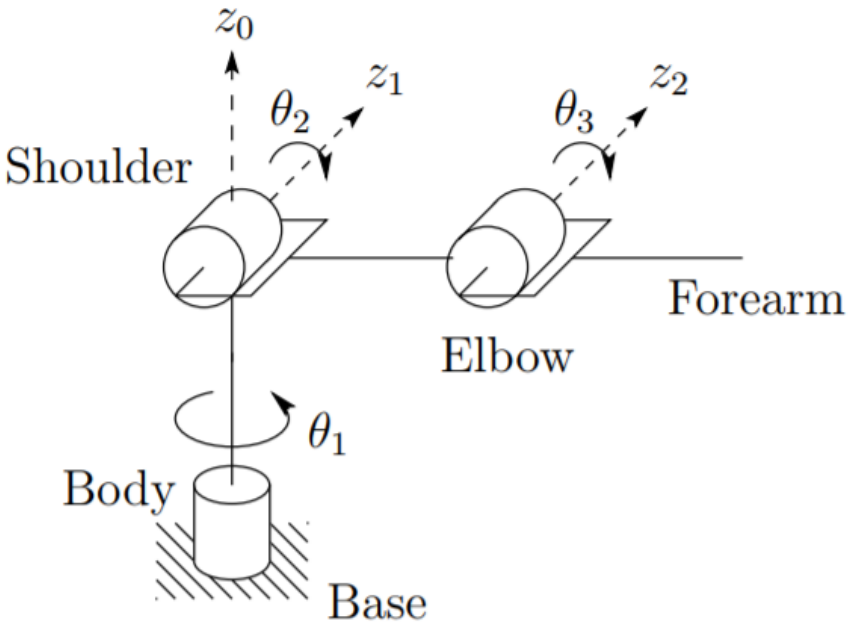


Figure 1.2: Schematics for an anthropomorphic manipulator [1]

The base concept of collision avoidance for the human mind is quite simple, if there is an obstacle do not crash with it, but for a robot this concept is not easy to implement. There are two different types of collision avoidance: one that the robot knows the plant (its surroundings and the obstacles that it contains) and has to plan the path to follow without hitting anything and the other where for each defined moment the robot has to sense its surroundings and evaluate if there is any object close by that needs to be avoided and, if so, decide how to do it. Both methods entail different disadvantages, the latter implies that the robot has a series of sensors that allows it to sense its surroundings for obstacles, the former is not equipped to deal with unforeseen situations.

In this project we will deal with a senseless robot, which means the path planning is done before the motion of the robot and cannot be changed midways. Due to the frequency of this problem there are already several proposed solution and methods developed throughout the years to solve the collision avoidance issue, however these methods are not specially designed for dual arm manipulators. This work aspires to analyze the existing solutions and evaluate how they perform when applied to a dual arm manipulator. In the end, we aim to find a motion planner capable of planning the path for both arms in a way that they do not bump into one another or with any object present in their surroundings.

1.3 Document Overview

Throughout this document there is a detailed explanation of all steps necessary to complete this work. First, in Chapter 2, the forward and inverse kinematics of each arm will be thoroughly explained. Chapter 3 presents a review of several motion planners, commonly used for collision avoidance, between them are the planners that will further one be applied to the dual arm manipulator. After having understood the kinematics and the motion planners we step in Chapter 4 where a study of the state of the art of motion planning on dual arm manipulators is done. Following this Chapter 5 presents the used implementation and some experiments' results. Here the programs along with some first tests' results are exposed. Chapter 6 replicates and discusses the results obtained when simulating real life like scenarios. Finally, Chapter 7 states all the conclusions taken from this work as well as some recommended future work to pursue.

Chapter 2

Forward and Inverse Kinematics

In order to operate any kind of manipulator it is fist necessary to understand its kinematics, i.e. how the manipulator moves and how each change in one of the joints affects the others and the end effector pose; it is also important to know which configurations can be achieved by the manipulator and which cannot. There are two types of kinematics: forward kinematics (FK) and inverse kinematics (IK). The former seeks to calculate the end effector's pose for a given joint configuration whereas the latter intends to know the joints' configuration for a wanted end effector's position.

As stated in Chapter 1, an anthropomorphic manipulator can take many forms. During this project we will use two anthropomorphic arms, each with a spherical wrist, whose schematics can be seen in Fig. 2.1. In addition to the body, shoulder and elbow joints in Fig. 1.2, there are three other revolute joints that grant the manipulator three more degrees of freedom. Each revolute joint, q_i, allows a rotation of θ_i around its z axis. Connecting each joint q_i to q_{i+1} there is a link with a size of d_i. The end effector is represented as a gripper in the far right of the picture.

Before we start analyzing the forward and inverse kinematics of the manipulator there are a few concepts that are fundamental to grasp. The first of them is the concept of frame, a frame can be interpreted as a referential, in $\mathbb{R}^3$ it has an x, y and z axis, all perpendicular with one another, being the origin the point where they intercept. In robotics is common that each joint has associated with itself a certain frame, therefor it is important to know how one can transform a point $\vec{Q}$ in frame i, with coordinate (x_i, y_i, z_i), to the same point but in respect to frame j, with coordinate (x_j, y_j, z_j). In order to achieve this one can use a transformation matrix T_i^j, that will transform coordinates in frame i to the frame j, being T_i^j:

$$T_i^j = \begin{bmatrix} R_i^j & \vec{P}_i^j \\ 0 & 1 \end{bmatrix} \tag{2.1}$$

Where R_i^j is the rotation matrix that characterizes the rotation between both frame and $\vec{P}_i^j$ the vector around which the translation is carried out. Having the transformation matrix we can compute the wanted $\vec{Q}_i^j$:

$$\vec{Q}_i^j = T_i^j \vec{Q}_i \tag{2.2}$$

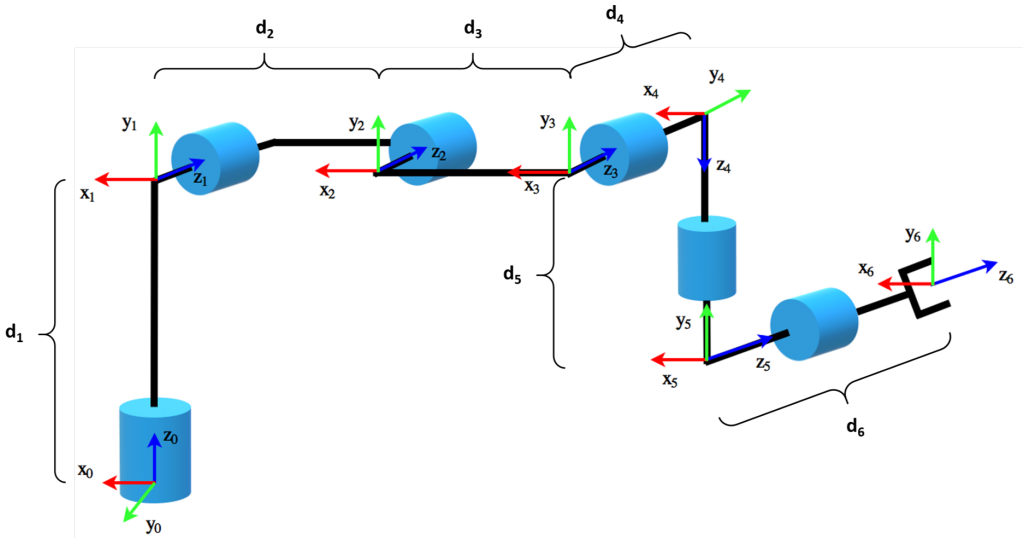


Figure 2.1: Schematics of an anthropomorphic arm with spherical wrist

Another important feature about transformation matrices is that they can be associated with one another, i.e. if we have a point $\vec{Q}_0$ in frame 0 and want to express it in frame k we can apply (2.2) where T_i^j will be the successive multiplication between all the frames from 0 to k, such that:

$$\vec{Q}_0^k = \prod_{i=0}^{k-1} T_i^{i+1} \vec{Q}_0 \tag{2.3}$$

Having understood these basic concepts we can now look into the forward and inverse kinematics of the manipulator.

2.1 Forward Kinematics

Forward Kinematics is the process of calculating the end effector's position based on the configuration of the joints. This implies that if we have set a certain joint configuration we can then compute the end effector's pose.

In fact, a homogeneous transformation can be represented by six numbers, three of them in respect to the translation and the other three to the rotation. However Jacques Denavit and Richard Hartenberg [2] introduced what is now known as the Denavit–Hartenberg (DH) method - or simply DH method - that allows the homogeneous transformation to be represented by only four elements, if the right coordinate frame is chosen. The coordinate frame assignment must follow the following rules:

1. z_i should be the q_i joint's actuation axis

2. x_0 and y_0 should be assigned according to the right hand rule

3. iteratively chose o_i, x_i, y_i and z_i with respect to o_{i-1}, x_{i-1}, y_{i-1} and z_{i-1} such that, if:

   (a) z_i and z_{i-1} are not in the same plane: then there is a line perpendicular to both axes that connects them by the shortest distance. This line should be x_i and the point where it intersects z_i shall be the origin of the frame, o_i;

   (b) z_i and z_{i-1} are parallel: x_i should be in the line perpendicular to both axes and should cross o_{i-1};

   (c) z_i and z_{i-1} match: x_i should be in the line perpendicular to them and can start at any point of the axes;

   (d) z_i intersects z_{i-1}: x_i should be perpendicular to the plane formed by z_i and z_{i-1} and o_i is the point where z_i and z_{i-1} intersect.

After this step is complete one can then compute a homogeneous transformation as:

$$A_i = \begin{bmatrix} \cos\theta_i & -\sin\theta_i\cos\alpha_i & \sin\theta_i\sin\alpha_i & a_i\cos\theta_i \\ \sin\theta_i & \cos\theta_i\cos\alpha_i & -\cos\theta_i\sin\alpha_i & a_i\sin\theta_i \\ 0 & \sin a_i & \cos a_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.4}$$

Where the DH parameters are:

- a_i - distance between z_{i-1} and o_i;
- α_i - angle between z_{i-1} and z_i;
- d_i - distance between x_i and o_{i-1};
- θ_i - angle between x_{i-1} and x_i;

To sum up if one seeks to compute the transformation matrix from joint q_i to q_j one can obtain it by:

$$A_i^j = \prod_{k=i+1}^{j} A_k \tag{2.5}$$

Applying this to the manipulator represented in Fig. 2.1 we get the DH parameters in Table 2.1. Where θ_i, $i = \{1,...,6\}$ are the only variables during the manipulator's operation.

With the Table 2.1 and (2.4) we can easily compute the end effector pose with each joint configuration.

Having calculated the DH parameter and understood the forward kinematics we can now step in to the inverse kinematics.

Joint	a_i	α_i	d_i	θ_i
1	0	$\frac{\pi}{2}$	d_1	θ_1
2	d_2	0	0	θ_2
3	d_3	0	0	θ_3
4	0	$\frac{\pi}{2}$	d_4	θ_4
5	0	$-\frac{\pi}{2}$	d_5	θ_5
6	0	0	d_6	θ_6

Table 2.1: DH parameters

2.2 Inverse Kinematics

Completed the analysis of the forward kinematics and examining the DH parameters in Table 2.1 we can conclude that the only parameters that change with the pose of the end effector are the angles θ_i. Since the inverse kinematics concerns itself with determining the configuration of each joint for a given end effector pose what it aims to do is, given T_6^0, which is the end effector's position and orientation in frame 0, determine each θ_i.

We start with a desired end effector pose:

$$T_6^0 = \begin{bmatrix} x_x & y_x & z_x & p_x \\ x_y & y_y & z_y & p_y \\ x_z & y_z & z_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.6}$$

where x_i, y_i and z_i, $i = \{x, y, z\}$ represent the orientation of the end effector and p_x, p_y and p_z represent the position.

First we will determine the angle θ_1. In order to do that we analyze the transformation between frame 1 and 5. We know that T_6^0 must be equal to $\prod_{k=1}^{6} A_k$, using (2.5) to manipulate these expressions to find the transformation from frame 1 to 5 we get:

$$(A_1)^{-1} T_6^0 (A_6)^{-1} = \prod_{k=2}^{5} A_k \tag{2.7}$$

If we examine the position on the z axis - the third line of the forth column - of each multiplication it will result in the following equality

$$(p_x - d_6 z_x) \sin\theta_1 + (d_6 z_y - p_y) \cos\theta_1 = d_4 \tag{2.8}$$

Knowing that

$$\vec{P}_5^0 = \begin{bmatrix} p_x - d_6 z_x \\ p_y - d_6 z_y \\ p_z - d_6 z_z \end{bmatrix} \tag{2.9}$$

And aided by the upper view seen in Fig. 2.2 we conclude that:

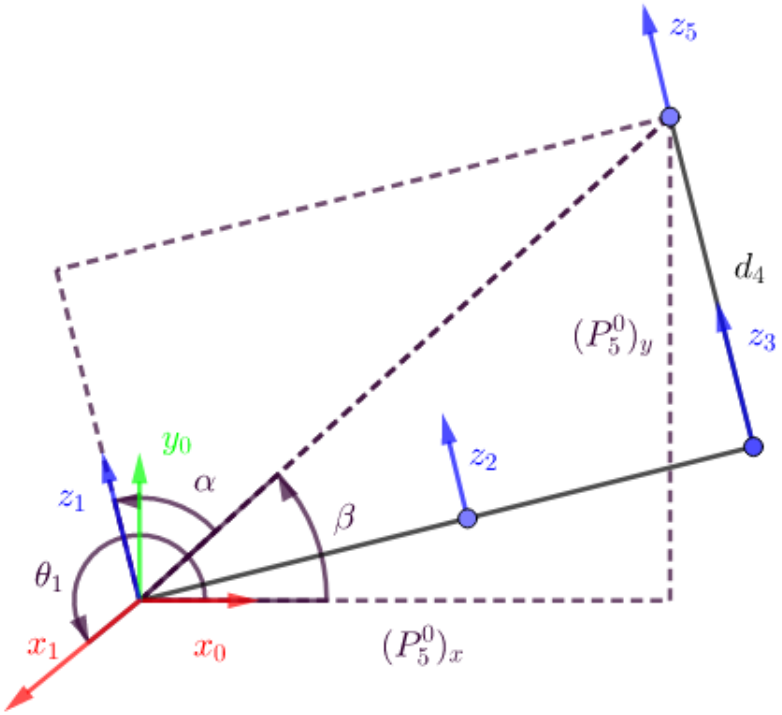


Figure 2.2: Upper View

$$\theta_1 = \beta + \alpha + \frac{\pi}{2} \tag{2.10}$$

where

$$\beta = atan2\left(\left(\vec{P}_5^0\right)_y, \left(\vec{P}_5^0\right)_x\right) \tag{2.11}$$

$$\alpha = \pm \arccos\left(\frac{d_4}{\sqrt{\left(\vec{P}_5^0\right)_y^2 + \left(\vec{P}_5^0\right)_x^2}}\right) \tag{2.12}$$

Having computed θ_1 it is possible to obtain θ_5 by repeating the same process but now considering the transformation from frame 1 to 6, resulting in:

$$(A_1)^{-1} T_6^0 = \prod_{k=2}^{6} A_k \tag{2.13}$$

Examining once again the position on the z axis yields:

$$p_x \sin\theta_1 - p_y \cos\theta_1 = d_4 + d_6 \cos\theta_5 \tag{2.14}$$

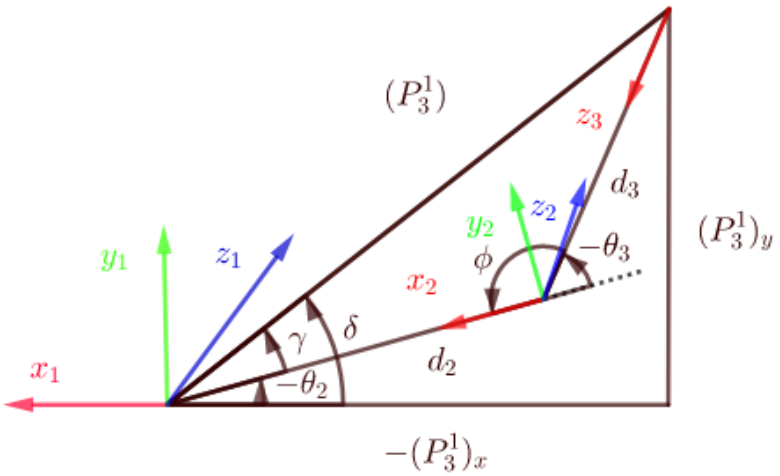


Figure 2.3: Schematics of joints 1 and 2 as a 3R manipulator

Allowing us to directly compute θ_5 as:

$$\theta_5 = \pm \arccos\left(\frac{p_x \sin\theta_1 - p_y \cos\theta_1 - d_4}{d_6}\right) \tag{2.15}$$

Calculated θ_1 and θ_5 we can now compute θ_6 using once again the matrices in (2.13) but this time considering the z coordinates of the rotation, i.e. the third line except the last column, which results in:

$$\begin{bmatrix} x_x \sin\theta_1 - x_y \cos\theta_1 \\ y_x \sin\theta_1 - y_y \cos\theta_1 \\ z_x \sin\theta_1 - z_y \cos\theta_1 \end{bmatrix} = \begin{bmatrix} \cos\theta_6 \sin\theta_5 \\ -\sin\theta_5 \sin\theta_6 \\ \cos\theta_5 \end{bmatrix} \tag{2.16}$$

Solving the system will lead to:

$$\theta_6 = atan2\left(\frac{x_x \sin\theta_1 - x_y \cos\theta_1}{\sin\theta_5}, \frac{y_x \sin\theta_1 - y_y \cos\theta_1}{-\sin\theta_5}\right) \tag{2.17}$$

There are now three joints whose angles we still need to compute, however if we analyze Fig. 2.1 we can see that, if only joint 2 and 3 are considered, the manipulator becomes a classical 3R manipulator. A visualization of this is present in Fig. 2.3

In this type of manipulators θ_3 is computed by:

$$\theta_3 = \pm \arccos\left(\frac{(P_3^1)_x^2 + (P_3^1)_y^2 - d_2^2 - d_3^2}{2 * d_2 * d_3}\right) \tag{2.18}$$

Where (P_3^1) is the position of the frame 3 in respect to frame 1, which is encoded in the fourth column of the matrix $A_1^{-1} T_6^0 A_6^{-1} A_5^{-1} A_4^{-1}$. From Fig. 2.3 we can also see that $\theta_2 = -(\delta - \gamma)$, from

simple trigonometry we can state that

$$\delta = atan2\left(\left(P_3^1\right)_y, \left(P_3^1\right)_x\right) \tag{2.19}$$

And from the law of sines we compute γ as:

$$\frac{\left(P_3^1\right)_x^2 + \left(P_3^1\right)_y^2}{\sin\phi} = \frac{d_3}{\sin\gamma} \tag{2.20}$$

$$\gamma = \pm \arcsin\left(\frac{\sin\phi * d_3}{\left(P_3^1\right)_x^2 + \left(P_3^1\right)_y^2}\right) \tag{2.21}$$

We are now missing only θ_4, however from (2.4) we know that:

$$\theta_4 = atan2\left(\left(A_4\right)_{2,1}, \left(A_4\right)_{1,1}\right) \tag{2.22}$$

being $\left(A_4\right)_{i,j}$ the element from A_4 in row i, column j. Knowing all the other joints' configurations it is simple to compute A_4 as:

$$A_4 = A_3^{-1} A_2^{-1} A_1^{-1} T_6^0 A_6^{-1} A_5^{-1} \tag{2.23}$$

Having computed the angles of each joint the inverse kinematics is complete, we can now know the configuration of each joint for a desired end effector. In addition it is important to mention that this type of manipulator entails that for a desired pose there is more than one possible configuration. For example one can put the elbow up or down, the wrist can be "in/down" or "out/up". This characteristic can be exploited as an advantaged but can also present some challenges since not all the configuration present the same manipulability.

Here we presented the forward and inverse kinematics for a single arm, since the kinematics is applied separately to each arm and the robot's body has only fixed joints there is no need to have in consideration the body of the robot.

Chapter 3

Path Planning Algorithms

Path planning, or motion planning, is a highly valuable and studied field not only in robotics but in other fields of studies. It has been used in digital animation [3], computational biology [4], self driving vehicles [5], medical surgery [6] and several different robots, from domestic to industrial [7]-[8].

Motion planning has the goal to find a path from A to B without hitting any obstacle. This can be done either online, where the plan is designed as the robot moves, or offline, where the path is decided ahead and then executed [9]. The former usually relies on sensory based information provided along the movement, whereas the latter might be less capable of dealing with unforeseen situations or unexpected obstacles.

The first step in motion planning is the definition of the configuration space, which can sometimes be a difficult and daunting task. The configuration space is the space of all transformations possible. Lozano-Perez [10] divides this space, C, in two classes: C_{free} if the configuration does not contain any obstacle or C_{obs} otherwise. The robot is commonly represented as a configuration, q, which has the same dimension as the configuration space it moves in. The goal and start configuration are usually designated as q_{goal} and q_{start} and a path is a set of connected configurations that connects q_{start} to q_{goal} without any of them being in C_{obs}.

Throughout this chapter we will approach several different motion planning algorithms, in doing so we will also discuss what are the main advantages and downsides of each of them. To do this there is a concept that has to be clear, that is the notion of completeness. Completeness, in motion planning, refers to the ability of an algorithm yielding a solution if there is one in a finite time. However, due to the complexity of some problems and to computational limitations, this is not always achievable, leading the way to different forms of completeness, such as resolution completeness, that allows the algorithm to find a solution if one exists at a given resolution of discretization, and probabilistic completeness, which means the probability of the algorithm finding a solution if there is one converges to 1 as time goes by [9].

3.1 Roadmaps

Roadmaps are a familiar concept to anyone that has ever wanted to go from point A to point B, which is precisely what motion planing aims to do, so it is easy to understand why this concept has been extended to the field of path planning.

The similarity from common roadmaps to roadmaps in the scope of motion planning is that when we want to travel from city A to city B we usually leave city A, get on the closest highway network and when we are approaching city B we take the nearest exit onto "smaller" roads. Applying this to motion planning we can say that we have a starting point, q_{start}, a set of known configuration that form several paths between themselves - the equivalent of the highway network - and a ending point, q_{goal}. Like in the previous example we have to have a path that connects q_{start} to the known configurations and one that connects these to q_{goal}.

It is important to define exactly what is a roadmap (from now on we will refer to roadmap in the scope of motion planning). A roadmap, **RM**, is a union of one dimensional curves if for all q_{start} and q_{goal} there is a path such that [9]:

- there is a path from q_{start} to q'_{start}, where $q'_{start} \in$ **RM**;
- there is a path from q'_{goal} to q_{goal}, where $q'_{goal} \in$ **RM**
- there is a path from q'_{start} to q'_{goal} in **RM**

Thus the final path will be $q_{start} \rightarrow q'_{start} \rightarrow q'_{goal} \rightarrow q_{goal}$.

One main advantage of roadmaps is that the same configuration can be used multiple times without the need to be computed once again. For any q_{start} and q_{goal} it only needs to compute a path to or from the roadmap and not the whole path or even the whole roadmap.

Although there are several different roadmaps algorithms, Delaunay triangulation [11], adaptive roadmaps [12], silhouette methods [13], just to mention a few, we will only approach two of them: visibility graphs and Voroni diagrams. Both of these methods are based on graphs, which are defined by nodes and edges, which is the name given to the connection between the nodes.

3.1.1 Visibility Graph

Visibility graphs are the simplest instance of a broader category named visibility maps. In this type of map each node is a vertex of an obstacle, and, as the name states, a node is only connected to another if it is visible to him, i.e. if we can draw a straight line from vertex i to j without intersecting any obstacle. Besides the vertices of the obstacles, the graphs also include the start and end point, q_{start} and q_{goal}. An example of this architecture can be seen in Fig. 3.1.

The process of creating a visibility graph is simple and follows an algorithm commonly known in computational geometry as plane sweep algorithm. This algorithm sweeps a line segment l, starting in the vertice we are analyzing the visibility, from 0 to 2π; for each intersection between l and a vertex the angle of l is stored as α_i. Then a vertex list, V, is initialized with the angles α_i

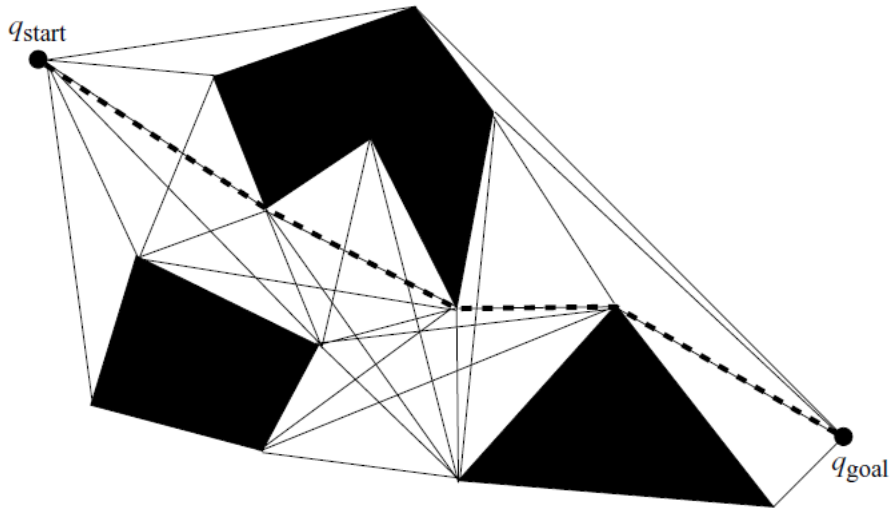


Figure 3.1: Example of a Visibility Graph. The thin lines represent the edges and the doted line the shortest path between q_{start} and q_{goal} [9]

sorted. Next a set, S, is initialized with the edges that intersect l when it is horizontal, i.e. with angle 0. Finally the list is updated. The whole algorithm for a vertex v can be described as:

1. Rotate l from 0 to 2π storing for each intersection with a vertex the angle α_i;
2. Sort α and store it in V;
3. Initialize S with the edges that intersect l when its angle is 0;
4. for each element of V:
   (a) check if the vertex of V is visible to v, if so add the edge to the visibility graph;
   (b) check if if the vertex of V is the beginning of an edge not yet in S, if so add the edge to S;
   (c) check if if the vertex of V is the ending of an edge in S, if so remove the edge from S;

Where step 4 describes the process of updating both the visibility graph, V and S. This last set is the one that allows to know if a the vertex v_i is visible to v since if the line that connects both vertices does not intersect the closest edge in S and if l does not intersect the interior of an obstacle, then v_i is visible to v.

This method, in addition to being complete, also yields the optimal path in terms of length but this presents one major disadvantage, since the nodes mostly represent obstacles vertices, it implies that the path to follow will be very close to the obstacle, in case of minor errors of control or disturbances this may lead to unwanted collisions. To avoid this problem one can represent the obstacles bigger than they actually are or change the optimal path found to one, that although non optimal, will force the robot to move further away from the obstacles. Another feature worth

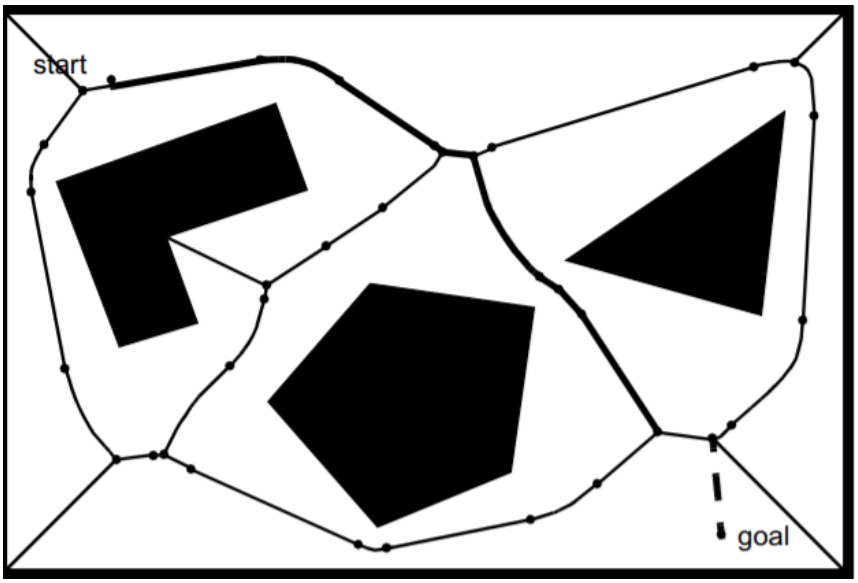


Figure 3.2: Example of a Voroni Diagram. [14]

mentioning is that as the number of obstacle or vertices increases so will the size of the graph, hence this method is fast and efficient in sparse environments whereas other methods may provide better results for denser scenarios [14].

3.1.2 Voroni Diagram

Another example of a Roadmap type motion planning algorithm are Voroni Diagrams. We can trace Voroni Diagrams' history as far back as the nineteenth century and their applications in computer science are vast, from file searching, cluster analysis to collision avoidance [15].

This method, opposite to the visibility graph, has not in its core the vertices or edges of an obstacle but rather the middle point between these. A Voroni Diagram is then a connection between points that are equidistant to the obstacles that surround them - Fig. 3.2.

There are three ways to create a Voroni Diagram: sensory based construction, polygonal space analysis or through the bushfire method. We will now briefly approach the operation mode of each one of them.

3.1.2.1 Sensory Based Construction

As the name states this method implies that the robot must have in its features sensors capable of detecting objects around it. To create the Voroni Diagram the robot must first scan the environment and for each point measure the distance between himself and the surrounding objects, placing himself in the middle point of these. As he wanders around the Voroni Diagram will be created.

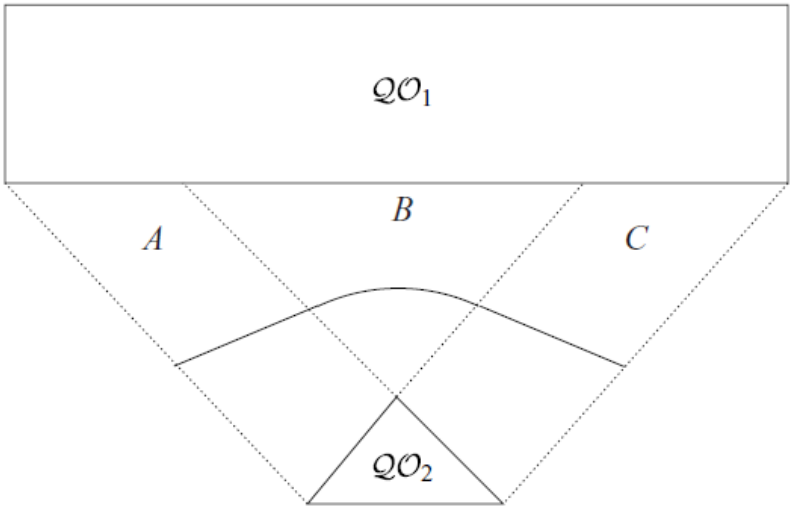


Figure 3.3: Types of equidistant set of points. [9]

3.1.2.2 Polygonal Space

An obstacle can be defined by its vetices and edge, the polygonal space analysis takes advantage of these features to create the Voroni Diagram. The set of points equidistant between two edges is a line, between two vertices is also a line and between an edge and a vertices is a parabola, as it can be seen in Fig. 3.3.

3.1.2.3 Bushfire Method

The bushfire method, or bushfire algorithm, requires that a grid be lay on the environment, each square on the grid shall be labeled with either 0 if it has no obstacle in it or 1 otherwise. This grid is then iteratively updated being the final result the same grid with each free pixel now containing the distance to the nearest obstacle.

If we view the bushfire method as a wave that starts at each obstacle and then propagates throughout the free space, then the points where the waves crash are points on the Voroni Diagram.

After the Voroni Diagram is created q'_{start} and q'_{goal} can be computed by drawing a line along which its distance to the boundary of the obstacles increases the fastest.

Although this algorithm is also complete it does not yield the optimal solution path in terms of length.

3.2 Cell Decomposition

Another form of motion planning is based on cell decomposition, where the field is divided into several polygons called cells. The most common cell decomposition is the trapezoidal decomposition, where the environment is carved into trapezes. Each trapeze is then adjacent to one or

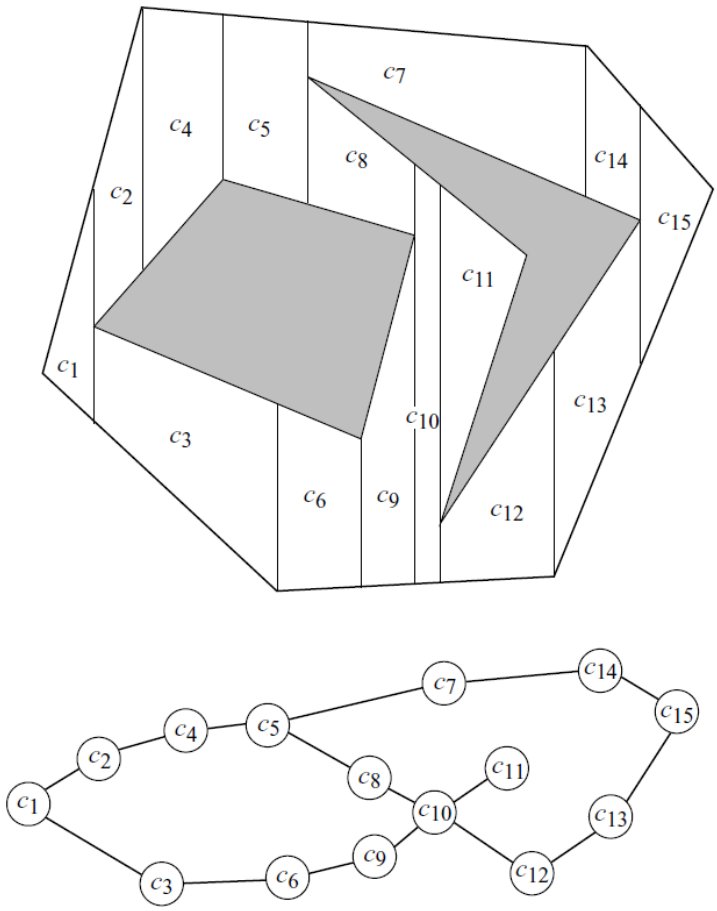


Figure 3.4: Field division in different cells (Up) and the respective adjacency graph (Bottom). [9]

more trapezes, this allows the cell decomposition to be seen as an adjacency graph, where each node is a cell - trapeze - and an edge between node exists if the trapezes are adjacent, i.e. if they share a boundary. Interestingly this adjacency graph can also be seen as a roadmap, however, cell decomposition differentiates itself from other methods by providing coverage, in the sense that can provide a path that covers all points in all the free space.

We can now break down this method into two steps:

1. Division of the field into cells (and consequent construction of the adjacency graph);

2. Computation of the path from q_{start} to q_{goal}.

The first step follows a plane sweep line algorithm as explained in 3.1.1 but the environment is swept by a vertical line, l, from left to write - along the x axis - and for each vertex an upper and lower vertical line is drawn. The vertex list V is now a sorted list with the x coordinates of each vertex and S is still the list of edges that l intercepts at the moment. After this step we obtain a cell decomposition and an adjacency graph as seen in Fig. 3.4

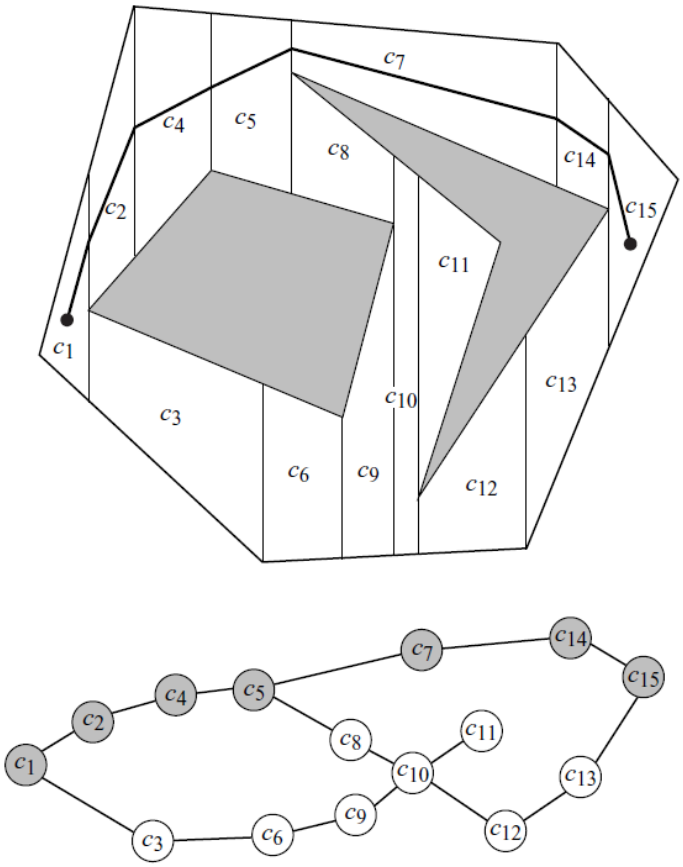


Figure 3.5: Final path from q_{start} - in c_1 - to q_{goal} - in c_5 - (Up) and the respective sequence nodes (grey) in the adjacency graph (Bottom). [9]

Obtained the adjacency graph it is now possible to apply a search algorithm to find the sequence of nodes from q_{start} to q_{goal}. The final path can then be computed by connecting midpoints of the boundaries to the center of each trapezoid as in Fig. 3.5

Here we explained in further detail the trapezoidal decomposition, however, other polygons can also be used, to this we call exact cell decomposition. Still, it is also possible to decompose the surroundings in to approximate cells [16], this type of division allows the cells to be labelled full, if all of its area is composed by an obstacle, mixed, if only part of its area is filled by an obstacle, or empty, if no obstacle is inside it.

3.3 Sampling Based Planning

So far we have studied two different kinds of motion planning algorithms, however as the complexity of the problems grows these methods fail to yield a fast solution, thus Sampling Based Planning (SBP) is usually the path to pursuit.

SBP was proposed in order to solve path planning problems in a 6 DoF robot [17], which makes it an ideal solution for the problem that this project aims to tackle. This method, as the

name refers, does a sampling of the configuration spaces and then with the samples it recovered defines the path to follow. The sampling of the surroundings provides SBP methods the peculiar feature of not having to handle the whole configuration space but only the sampled points of it. Although this type of algorithms have proven themselves successful in solving complex problems they do not ensure completeness, but rather probabilistic completeness, where the probability of finding a solution converges to one as time goes by.

The family of SBP algorithms is broad so we will not approach all of the known methods.

3.3.1 Probabilistic Roadmap Method

Kavraki et al. [18] divide the PRM in two phases: the learning phase, where the roadmap is created, and the query phase, where the path between a start and end configuration is computed. PRM is also an example of a roadmap method, so the rules stated in 3.1 still apply. The main difference between this method and the other roadmaps described before is that the creation of the roadmap does not deal with the whole configuration space but rather samples of it. Despite that PRM still handles the configuration space like a graph, where each configuration is a node and each edge is a path between configurations.

The sampling of the surroundings can be done following several different distributions, from uniform to gaussian, and the choice of the sampling process is of the utmost importance. After the sampling strategy has been defined the construction of a probabilistic roadmap goes as following:

1. A node q_{rand} is sampled from the configuration space;
   (a) If $q_{rand} \in C_{obs}$ forwent q_{rand};
   (b) If $q_{rand} \in C_{free}$ add q_{rand} to the graph;
2. Find all nodes near q_{rand};
3. Attempt to connect all nodes found to q_{rand};
4. Check if there is collision and disconnect colliding paths;

This process is repeated until a certain number of nodes is reached resulting in the roadmap seen in Fig. 3.6

After the learning phase is completed the query phases is the same as any other roadmap method, a path from q_{start} and q_{goal} to the roadmap is found, resulting in q'_{start} and q'_{goal}. Then a path between q'_{start} and q'_{goal} is computed. Being the final path $q_{start} \rightarrow q'_{start} \rightarrow q'_{goal} \rightarrow q_{goal}$.

3.3.2 Rapidly-Exploring Random Trees

Especially designed for handling nonholonomic constraints and high degrees of freedom manipulators RRTs were introduced by LaValle [20] and, although they are descendant from PRM, differ from it by being biases towards the free configuration space. Another particular characteristic of this method is that the path is iteratively searched from q_{start} to q_{goal}.

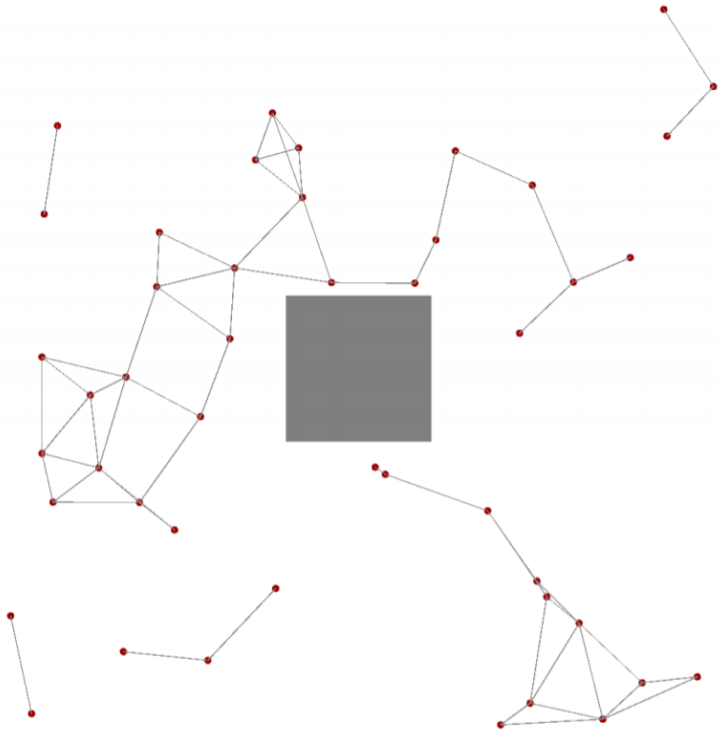


Figure 3.6: Example of a probabilistic roadmap. [19]

So far, and in most path planning problems, the configuration space is usually used as the metric space, however in RRT a more general concept is used, the state space. The state space can have many interpretations, but in RRT it encodes both the configuration space and velocity. Similarly to the configuration space, a state space X can be divided in X_{obs} - in this are also included states that violate the velocity bounds - and its complement X_{free}. A nonholonomic constraint is defined by $\dot{x} = f(x,u)$, being $\dot{x}$ the derivative of the state x over time. In order to achieve a new state x_{new} a control u is applied to x for a period of time Δt, which mathematically translate to integrating $f(x,u)$ for a fixed interval of time Δt.

A tree - in mathematics - is a particular type of graph where each pair of nodes is connected by exactly one path, so RRT stores the state space as a series of nodes and edges. In this method the path from x_{start} to x_{goal} is computed according to:

1. The search begins at x_{start}
2. A random state x_{rand} is sampled from the configuration space;
   (a) If $x_{rand} \in X_{obs}$ forwent x_{rand};
3. The nearest state in the tree to x_{rand}, x_{near}, is computed using the Nearest Neighbor method;
4. An input u is chosen in order to minimize the distance between x_{near} and x_{rand}
5. The input u is applied to x_{near} for a time Δt, resulting in x_{new}

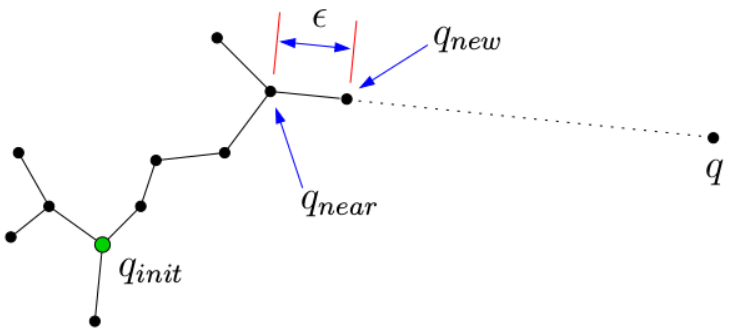


Figure 3.7: Rapidly-Exploring Random Tree - Algorithm example. [21]

6. x_{new} is added to the tree, along with a connection - edge - with x_{near}

When x_{new} is x_{goal} then a path from x_{start} to x_{goal} has been found. It is important to notice that the need to compute x_{near} comes from the fact that a fixed interval of time Δt is applied to u, which results in a maximum distance, ε, that can be travelled from x_{near}- Figure 3.7.

3.3.3 Rapidly-Exploring Random Trees-Connect

A couple of years after RRT was created a different approach to it is developed, thus giving place to Rapidly-exploring Random Trees-Connect, commonly referred to as RRT-Connect. This method was specially conceived in order to manipulate 7 DoF human arms for the automatic animation of grasping and manipulation tasks for animated characters in interactive 3D virtual environments [21].

RRT-Connect combines the concept of RRT with a greedy heuristic that attempts to connects two trees, the existence of two trees is another aspect that differentiates the two methods. Each tree is initialized with either the initial or final configuration. The idea is that RRT provides a rapidly uniform exploration of the free space while RRT-Connect takes advantage of these characteristics but directs it towards the connection of both trees, leading RRT-Connect to yield faster results.

In Section 3.3.2 and in Figure 3.7 is clear that there is a maximum distance of ε that can be travelled towards q, in RRT after x_{new} has been added to the tree a new x_{rand} is sampled, however in RRT-Connect the path between q_{near} and q_{rand} is continuously built until q_{rand} or an obstacle is reached, only then a new random sample is taken. Each tree is extended in turns and in the end of each turn an attempt to connect the new node to the other tree's nearest node is made, if this attempt is successful then both trees are connected and a path connecting the start and goal configuration has been computed.

3.3.4 Kinematic Planning by Interior-Exterior Cell Exploration

As robotic systems become more complex, physics-based simulation becomes a necessity, this is the main catalyst for the development of Kinematic Planning by Interior-Exterior Cell Explo-

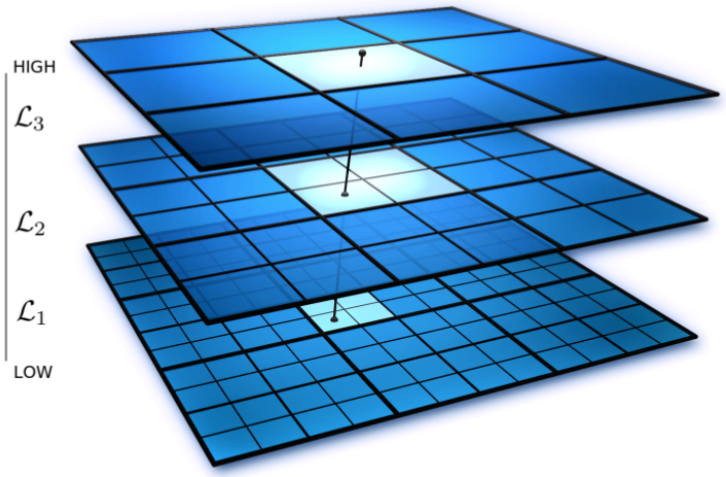


Figure 3.8: State space discretization. [22]

ration (KPIECE) [22]. Although it is also categorized as a sampling planning algorithm, KPIECE presents a distinctive approach from the previous two methods, first it is designed for systems where only a forward propagation routine is possible and second no state sample or distance metric are necessary (which is a great advantage for problems where these are hard to define). All these features make this model appropriate to handle problems not characterized by equation of motion but rather by physical models.

Analogously to the previous examples, KPIECE also builds a tree of motions in the state space. Each motion μ is composed of (s,u,t), where s is a state in the state space, u is a control in the control space and t is the amount of time that u is applied from s to produce a motion. The state space is discretized and divided in levels and each level in cells. Each level provides a different resolution, being that lower levels have higher resolution - Figure 3.8. A level k can then be formally defined by:

$$\forall_{i\in\{1,\ldots,k\}} : \mathscr{L}_i = \{p_i | p_i \text{is a cell in the grid at level } i\} \tag{3.1}$$

$$\forall_{i\in\{2,\ldots,k\}} : \forall_p \in \mathscr{L}_i, \mathscr{D}_p = \{q \in \mathscr{L}_{i-1} | q \in p\}, \text{ such that} \tag{3.2}$$

$$\forall_p \in \mathscr{L}_i, \mathscr{D}_p \neq \emptyset$$
$$\cup_{p\in\mathscr{L}_i}, \mathscr{D}_p = \mathscr{L}_{i-1}$$
$$\forall_{p,q} \in \mathscr{L}_i, p \neq q, \rightarrow \mathscr{D}_p \cap \mathscr{D}_q = \emptyset$$

Viewing the whole problem we get that for a motion μ there is one, and only one, cell at each level that μ is part of, if this requirement is not fulfilled then the motion is splitted into smaller motions. Hence, a motion will be a whole set of cells - one for each level -, this set is called a cell chain, i.e. a cell chain for k levels of discretization is k-tuple. These cell chains are the base of the motion tree.

It is important to understand that the cells are only instantiated if a motion requires so, this leads to the concept of exterior and interior cells (thus the algorithm's the name). A cell is considered as exterior if it has, at its level, less then $2n$ instantiated neighboring cells (diagonal neighboring cells are ignored), being that n is the dimension.

In the beginning of the algorithm the motion tree is initialized with a motion μ_0 that contains the initial state, q_{start}. An empty grid structure, G, is created and the motion μ_0 is added to it according to:

1. Divide the motion so that the motion is contained in one cell at each level;
2. For each divided motion
   (a) Find the respective cell chain, Instantiating cells if needed
   (b) Add the said motion to the cell at the lowest level in the chain
   (c) Update the lists of interior and exterior cells

This process can be extended to any motion one seeks to add to G. At this moment the motion tree only has the initial motion, in order to expand it is needed that, for each iteration:

1. A cell chain c is chosen from G, with a preference for exterior cells;
2. A motion μ is selected from c with a half normal distribution
3. A state s is selected along μ
4. A control u and a time t are randomly sampled
5. If a motion (s,u,t) is valid then
   (a) The valid motion $\mu_0 = (u,s,t)$ is constructed
   (b) If μ_0 already reaches the goal region then the path to μ_0 is returned
   (c) The motion μ_0 is added to G

KPIECE has proven to yield solution to complex problems and with relative fast planning times, mostly due to its parallel implementation, taking advantage of the increasing computational power over the last few years.

3.3.5 Expansive Space Trees

Extensive Space Trees (EST) [23] share some of PRM's base principle, similarly to this algorithm EST also creates a roadmap, however its sampling is directed towards the areas of interest of the problem to solve.

EST is based in the principle that if an expansive configuration is uniformly sampled at random and for each pair of samples only points that have straight line paths are connected, then the resulting roadmap are highly probable to be connected regions of the configuration space.

Besides the parity with PRM due to the usage of a roadmap EST also share a common feature with RRT-Connect, in both planners two separate trees are built, one from the initial configuration, q_{start}, and the other from the final one, q_{goal}. However, instead of building the roadmap for the whole configuration space, merely the part of the roadmap that is connected to either the initial or final configuration is built. In a first instance only the neighbourhood of q_{start} and q_{goal} are sampled, then after a connection is found the neighbourhood of the new found configuration is sampled, this process repeats itself until a path is found. The algorithm can be then divided in two different phases, the expansion and the connection.

The expansion process is executed by:

1. Picking a node, n from the roadmap with a probability $\frac{1}{w(n)}$
2. Sampling K points from the neighbourhood of n
3. For each q configuration picked;
   (a) Calculate the probability $w(q)$ and retain q with a probability of $\frac{1}{w(q)}$
   (b) If q is not in C_{obs} and there is a straight line path between n and q then add q to the tree with an edge connecting n and q

There is a key consideration in these steps and that is the probability $\frac{1}{w(x)}$, where is x represents a configuration (or node). $w(x)$ is the number of sampled nodes in the tree that lay in the neighbourhood of x, being that this neighborhood is defined by all the configuration within a maximum set distance.

The process explained above applies to both trees, the one whose root is the initial configuration, T_{start}, and whose root is the final configuration, T_{goal}. It is necessary to check if a connection between the two is possible, and if so a path from q_{start} to q_{goal} has been found. Thus the connection phase is set by:

1. For every x in T_{start} and y in T_{goal}
   (a) Check if x and y are close enough to one another
   (b) If so, determine if there is a straight line path from x to y

Step 1a is necessary to reduce computational costs, since it is unlikely that a straight line path can be computed if the configurations are too far apart. If the final line returns a positive outcome then a path has been computed.

3.4 Search Algorithms

Having described several motion planning algorithms there is still one more issue that we must discuss in order to wrap up this chapter and that is search algorithms. We have refrained from explaining how a path between two nodes within a graph can be found and that is what search

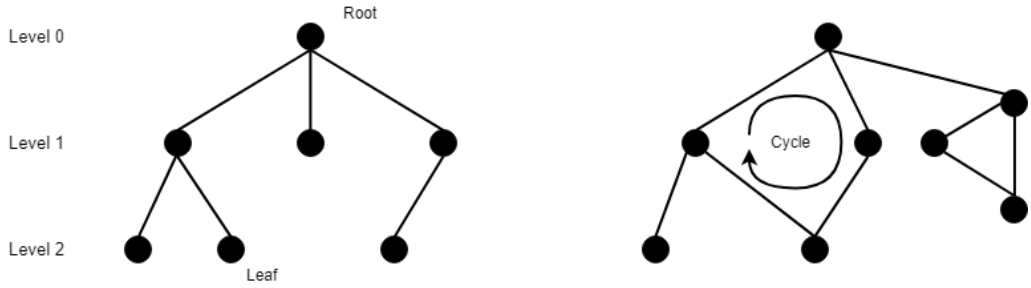


Figure 3.9: Example of a tree (Left) and a graph (Right).

algorithms are used for, they seek to find a sequence of nodes within a graph that, by being connected, can lead from a starting to a goal node.

Previously we have explained that a graph is the combination of nodes and their connection, to which we call edges, we have also referred to trees, that are acyclic graphs - you cannot form a path from and to node n_i without visiting one other node at least twice. For clarification an example of a graph and a tree is presented in Fig. 3.9. Each node in a tree can have children - nodes in the higher level - if a node has no children then it is called a leaf. All nodes must have a single parent - a node in the lower level - except for the root that is characterized by not having a parent node.

In this section we will explain two search algorithms, depth-first and breadth-first search, by first stating how they apply to trees and then extending them to graphs. We will then describe two other search algorithms commonly used in motion planning.

3.4.1 Depth-first Search

Depth-first search in a tree starts by examining the root then moves on to one of its children, then to the child's children and so on until a leaf is found. If so, then it backs out a level and moves to another child until it finds a leaf again. If the goal node is found then the search stops and a path from the root is found.

In a graph there is no concept of levels or children but in order to simplify we will assume that, in a graph, a node's n_i children are the nodes connected to it whose edge did not lead to it. The concept of level is replaced by what is called link length, which is the number of edges in a path. Extending the tree depth-first search to a graph we would have an initial node, from whom we would choose a child - in this case every node connected to it is a child -, this process would be repeated iteratively increasing the link length from the start node until a childless or already visited node is reached.

The difference between tree and graph depth-first search can be seen in Fig. 3.10.

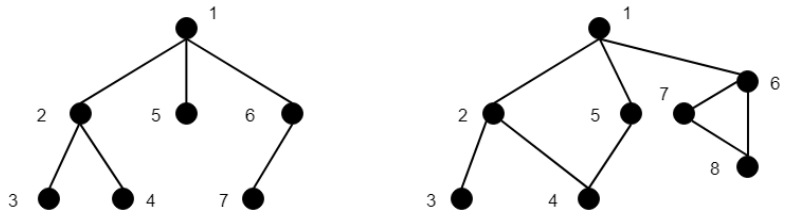


Figure 3.10: Depth-first search in a tree (Left) and in a graph (Right)

3.4.2 Breadth-first Search

Opposite to depth-first search the breadth-first search starts at the root then visits all of its children, then all of its grandchildren and so on until the goal node is found. Analogously, in a graph each node's child is visited, i.e. each node with the same link length, then one of this children is chosen and the process repeats itself until the goal node is found. Fig. 3.11 is an example of this algorithm for both trees and graphs.

3.4.3 A* Algorithm

Depth-first and breadth-first methods both search a graph without any particular influence or bias to a particular node, however if there is some information about the goal node, e.g. the coordinates of the final configuration, it can be used towards a more efficient search. It is important when discussing search algorithms to distinguish from efficient and optimality, efficient refers to the number of the nodes in the path, whereas optimality refers to path length. One cannot expect depth-first or breadth-first algorithms to yield an efficient path, although breadth-first can return an optimal path.

Assuming that an useful information about the end node is provided one can try to guess what will be the best node to explore, for example if, like stated before, we know the final configuration's coordinates we can choose the nodes that lead successively to a shorter distance to the final point. To this guess rule we call heuristic.

An A* algorithm will yield an optimal path if the heuristic chosen is optimistic, i.e. if the "guess rule" returns a value that is less or equal to the cost of the shortest path, in the graph, from the current to the goal node. An example of an optimistic heuristic is the use of the hypotenuse when computing the path between its vertices, since the actual path would be sum of the cathetus.

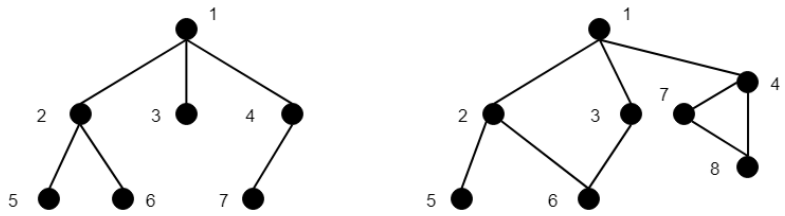


Figure 3.11: Breadth-first search in a tree (Left) and in a graph (Right)

Instead of presenting the steps that define A* we will first, for clarification purposes, provide an example and only then do a formal and general exposure of the algorithm.

Lets suppose we have a graph as presented in Fig. 3.12 where the start node is **A** and the goal node is identified, the length of each edge is established next to it and inside each node there is the heuristic value. A* needs to form a list of nodes sorted by their priority, which is the sum of each edge length needed to achieve the node from the start plus the heuristic value of that node. The smaller the priority value, the higher the priority.

The list starts with the starting node, then this node is removed and its adjacent nodes are added, resulting in the first table in Fig. 3.13. The best node to expand now is **B**, it is then removed from the list and its adjacent nodes are inserted - note that a node that was once in the list cannot be inserted in it again so the start node is ignored. The list is then re-sorted and **E** becomes the node with higher priority (second table of Fig. 3.13), however **E** has no neighbor nodes so it is not possible to continue the search. When this occurs the next node on the list becomes the node to expand, in this example that is **C**. Once again **C** is removed from the list giving place to its adjacent nodes **H** and **F**. After reorganizing the list **H** becomes the node to pursuit, since the goal node is a neighbor of **H** it is possible to compute the length of complete path, that is the sum of the edges that lead so far from **A** to the goal node. Still the algorithm does not stop here due to the fact that there are nodes in the list that present a smaller priority value then the length of the found path, this means that a better path could still be found. To verify this A* expands the nodes whose priority values are smaller than the path's length already found. In this case it expands **D**, resulting in the final table seen in Fig. 3.13. Updating the list with the adjacent nodes of **D** and computing the priority values it is clear that there is no better path since the priority values are all equal the path's length and neither of this node is yet the goal one. A* has then computed the final path $\mathbf{A} \rightarrow \mathbf{C} \rightarrow \mathbf{H} \rightarrow$ Goal.

After this example it is easier to understand the algorithm overall process. In order to operate there are a few *tools* it needs:

- A priority list, P;
- A set PC that stores the nodes that have already been processed;
- $Neighbor(n)$, the adjacent nodes of n;
- $l(n_1, n_2)$, length of the edge connecting n_1 and n_2
- $length(n)$, function that computes the sum the edges that connect n to the start node;
- $h(n)$, function that computes the heuristic value of node n;
- $f(n) = length(n) + h(n)$, function that compute the priority value of n.

Having this the whole search algorithm for a goal node n_{goal} can be defined as:

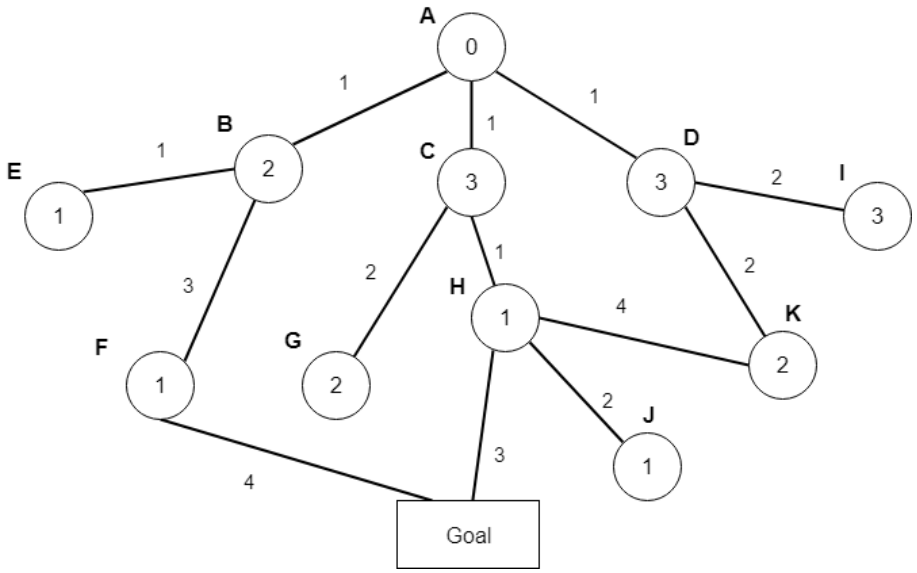


Figure 3.12: Example of a graph label from A to K where the the initial node is A and the goal node is identified. Each edge has associated with it a length (identified next to it) and each node has inside it the heuristic value

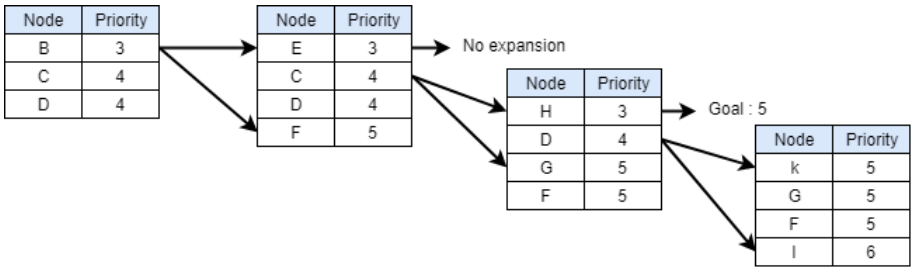


Figure 3.13: Set of nodes in the list. The priority is the some of the length of the edges from the start node the current one and the priority of the current node.

While P is not empty
    Choose $n_{best} \in P$ such that $f(n_{best}) \leq f(n), \forall n \in P$
    Remove n_{best} from P and add it to PC
    If $n_{best} = n_{goal}$:
        Stop.
    For all $x \in Neighbor(n_{best})$ and $\notin PC$
        If $x \notin P$
            Add x to P
        Else if $length(n_{best}) + l(n_{best}, x) < length(x)$
            Update path until x to include n_{best}

An A* algorithm will search a graph efficiently but is highly sensitive to the chosen heuristic, which makes this a fundamental step that has to be adjusted accordingly to the problem it faces.

There are also two particular cases of A* that are worth mentioning. The first one is when $f(n) = h(n)$, that is when the search only considers the heuristic value, i.e. the guess for the best path; to this we call a greedy algorithm. The second is when we choose $f(n) = length(n)$, which implies that the algorithm will always choose the node with the shortest edge length, this type of algorithm is called Dijkstra's algorithm.

Reached the end of this chapter we now have all the means necessary to implement a complete motion planning - and collision avoidance - algorithm, that will not only imply a discretization or sensing of the environment, but also a creation and search of a graph. Resulting in a path from A to B without crashing with any obstacle. In the next chapter we will present different examples of case studies that implement motion planning algorithms on dual-arm manipulator.

Chapter 4

Motion Planning on Dual-arm Manipulator

Motion planning algorithms can be applied to several different problems, however their implementation in robotics is, not only obvious, but imperative. Given that this project aims to find a collision avoidance system for a dual-arm manipulator it is both relevant and necessary to study the state of the art in the field, i.e. what has already been developed to tackle problems similar to the one we are trying to solve. Throughout this chapter we will do an overview of some of the work that applies motion planning algorithms on dual-arm manipulators. Note that we will only present the most relevant aspects of each project and it is not our aim to provide an extensive and detailed explanation of what was done, for that we refer to the original paper.

In robotics, motion planning can be applied to three different cases: single body mobile robots, fixed manipulators or mobile robots with attached manipulators. It is clear that motion planning - and collision avoidance - is essential in robotics, therefor it is expected that the literature is vast which forbids us to address it all. We will, however, overview the most relevant work in the light of our own problem, i.e. we will only focus on fixed manipulators since these are the one we will handle during our work.

4.1 First Steps Towards Dual Arm Motion Planning

One of the first experiments produced in dual-arm motion planning was done by Koga and Latombe in 1992 [24]. Their work is important to comprehend the first steps taken in order to solve the dual-arm manipulating problem at a time where most planning research was directed at mobile robots with fix kinematics. The goal of their experiment is to plan the motion of two robot arms in order to transfer an object from an initial to a desired position in a workspace constituted by obstacles. The manipulators considered are rather simple - maximum of 3 revolute joints - and the act of grasping is defined by positioning the end effector of each arm on each end of the object. To move the object both arms must be grasping it.

To address the defined problem the authors propose three different planners. For each of them the paths computed are an alternation of three different subpaths: one-arm transit path, where only one of the arms is moved while the other holds the object, these are then classified as obstacles; two-arms transit path, where both arms can move and neither of them hold the object; transfer path, where both arms hold the obstacle and the three corpses move as one.

Planner 1 assumes that the obstacles and the object are in a parallel plane to the arms, which results in single collision possible between the arms themselves. First the planner computes the object's path among the obstacles based on the best-first search potential field. Then for each configuration of the obstacle's path the grasp configuration for each arm is calculated and finally a path for each arm is generated.

Planner 2 considers the case where the obstacles, the object and both arms are in the same plane, meaning that the arms may also collide with the obstacles. This method constructs a manipulation graph, where each node is a grasp configuration and a connection between nodes is formed if a transit path between nodes exists. The created graph is then searched through a breadth-first search algorithm.

Planner 1 and 2 can only deal with a few DoFs, thus a third planner is presented in order to solve more complex problems. The planner mainly moves as a closed loop chain - where both arms are grasping the object. It traces the descent gradient of a potential field until it finds a minimum. If this minimum is not the global minimum, i.e. it is not the goal configuration, it will attempt to leave it by either a random walk or by planning a transit path. Although it is suitable for more DoFs this last planner still fails when the number of obstacles in the workspace is increased.

These algorithms are still in an early stage and present a great number of limitation. Nevertheless the fact that a non fixed kinematic robot is consider tackles new issues for the time - one of them being the dynamic kinematic and the other the constraints imposed on the arms' motion - paving the way for further work and improvements.

4.2 Heuristic Search on Dual Arm Motion Planning

Hardware enhancement allied with an increasing interest in the field of robotics allowed the development of more sophisticated and effective motion planning algorithms, like the heuristic search proposed by Cohen et al. [25]. This method will be explained in further detail due, not only to the yield results, but mainly to its similarity to the problem we aim to tackle. They purpose a heuristic search-based approach to motion planning that can handle the high-dimensionality of manipulation. The results for a dual-arm robot with 7 DoFs in each arm are presented.

The method proposed is based on three key principles: a manipulation lattice graph, ARA* search and informed heuristic.

The representation of the planning problem is done through what the authors call a manipulation lattice graph. In this graph each node is considered a state, i.e. a joint configuration of the robot. Thus a state of an n joint manipulator is defined as the n-tuple $(\theta_0, \theta_1, \ldots, \theta_n)$. An edge is a transition between two states, these transitions are named motion primitives, that can either

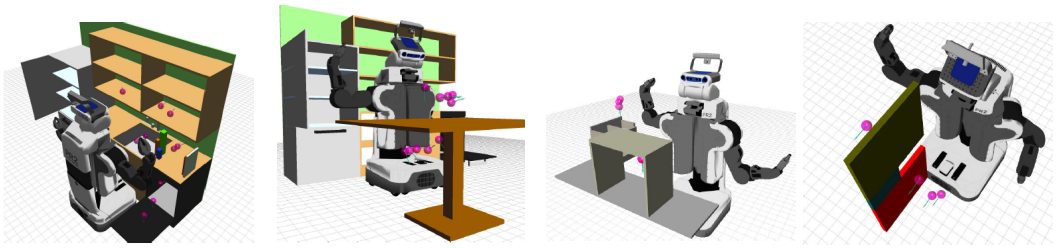
be static or adaptive. Thus an edge of an n joint manipulator is defined as the vector of joint velocities $(v_0, v_1, \ldots, v_n)$. The type of motion primitives is defined according to when they are generated, static primitives are chosen before the search begins and their purpose is to uniformly explore the space for a valid path, adaptive primitives on the other hand are computed whenever they are needed for a certain state expansion. The latter are particularly useful when the end effector of a particular state is in short distance to the goal, then an adaptive primitive can be use to compute the path from the state to the goal. Before moving on to the next principle there is another feature of the approach that must be noticed and that is its non uniform dimensionality. The method takes advantage on the fact that not every DoF needs to be used to compute a save path, so it first computes a path to a region within the range of the goal without using all DoFs - lower dimensionality - and when in that region it computes the final path using all the DoFs - the full dimensionality of the problem.

After the construction of the manipulation lattice graph the search is done by an anytime search heuristic, ARA*. ARA*, although similar to A*, finds solutions bounded on suboptimality and improves these solutions until a timeout is reached, thus allowing faster computations. This search presents two main differences to the A* algorithm explained in 3.4.3, the first being the reuse of the states it already computed in previous iterations and the second being the multiplication of the heuristic by a factor ε that is decreased until a timeout. The cost of the solution will then be ε times more expensive then the optimal one. The search stops when ε is 1 or the timeout established is reached. The cost function used aims to minimize the path length while maximizing the distance to obstacles.

The final part of the approach is the definition of the heuristic, for this the authors use a single heuristic that guides the search towards the goal position (x, y, z) - the orientation of the goal is disregarded. To calculate the heuristic value a 3D breadth first search that calculates the costs of the least-cost collision free path from the goal to every cell in the grid is applied.

This approach is applied to a dual arm robot, first considering only one arm and then both. For the one arm case the robot is placed in four different environments depicted in Fig. 4.1, for each of them a comparison between the performance of the proposed method, ARA*, RRT* and RRT-Connect is presented. Table 4.1 and 4.2 replicate the results produced; in Table 4.1 ARA* and RRT* stop after the first solution found whereas in Table 4.2 the algorithms run for 60 seconds. The authors conclude that ARA* is competitive in most environments and produces shorter paths. They also test the consistency of the method, i.e. if given similar goal configurations the path computed is similar and it revealed to be so, this implies predictability which is a desirable feature for a human controlling or interacting with a robot.

For the dual-arm case, where both arms are considered, a constraint is added to the goal pose, it is assumed that the object to grasp has to be in an upright pose. This allows a dimensionality reduction since the pitch and roll angles are set to zero. It is also important to refer that each state is not defined by the 14-tuple but rather by a 6-tuple $(x, y, z, \theta_{yaw}, \theta_{left}, \theta_{right})$, where x, y, z represent global position of the center of the object, θ_{yaw} the yaw angle and $\theta_{left}, \theta_{right}$ the elbow angles of the left and right arm respectively. At any point this configuration can be mapped to



(a) Kitchen (b) Tabletop (c) Industrial (d) Narrow Passageway

Figure 4.1: Tested scenario for one arm experiment. The pink spheres with cyan arrows indicate the desired 6D goal poses for the right end-effector [25]

Environment	Kitchen			Tabletop			Industrial			Narrow Passageway		
Planner	ARA*	RRTC	RRT*	ARA*	RRTC	RRT*	ARA*	RRTC	RRT*	ARA*	RRTC	RRT*
Planning time *(mean, sec)*	0.31	0.01	0.87	0.98	0.01	0.03	0.14	0.01	6.06	0.74	0.66	3.90
Planned length (joint space) *(mean, rad)*	9.52	13.13	12.90	10.97	10.20	10.19	5.76	12.12	12.33	17.33	25.66	23.24
Simplified length (joint space) *(mean, rad)*	6.93	9.81	9.30	7.37	8.14	7.71	4.09	8.81	6.88	9.54	13.56	12.60
Success rate	100%	100%	87%	100%	100%	100%	100%	100%	80%	100%	100%	100%

Table 4.1: Performance comparison of three planners for single-arm manipulation in the scenarios shown in Figure 4.1 [25]

Environment	Kitchen		Tabletop		Industrial		Narrow Passageway	
Planner (joint space) *(mean, rad)*	ARA*	RRT*	ARA*	RRT*	ARA*	RRT*	ARA*	RRT*
Planned length (joint space) *(mean, rad)*	7.74	9.21	6.51	10.71	4.43	14.26	12.58	23.18
Success rate	100%	80%	100%	100%	100%	94%	100%	100%

Table 4.2: Results of ARA* and RRT* with a fixed planning time budget of 60 seconds [25]

Figure 4.2: Tested scenario for dual.arm experiment.Clockwise from top left: stick around a pole, wood board in bookshelf, tray with wine glasses under a table, tray with wine glasses near wall and tray with a scotch glass in bookshelf [25]

the 14 dimension space. To assess the method's output twelve experiments that deal with real life scenarios were conducted. Fig. 4.2 shows the simulation's environments, the obstacles are in purple and the collision model of the manipulated objects in cyan. For each of the experiments a timeout of 15 seconds is set and the time until the first solution is found, the number of expansions until the first and final solutions as well as the parameter ε - which is initialized at 100 - are registered and can be seen here in Table 4.3.

4.3 Re-grasping Act of a Dual Arm Manipulator

Another interesting work in manipulation motion planning is conducted by Vahrenkamp et al. [26], not only the movement of two arms cooperating with one another is addressed but also the act of re-grasping an obstacle. In spite of the fact that the robot used not only two 7 DoFs arms but also a 3 Dofs hip the re-grasping actions can nonetheless be applied to dual-arm manipulators.

Two different planners based on RRT algorithms (see 3.3.2) are proposed. The first one is J^{+}-RRT where the extension of the RRT toward a goal pose is biased by the pseudoinverse Jacobian. The other method is called IK-RRT and is based on bidirectional RRT, which means instead of one single RRT two are initialized, one with the start configuration and the other with the goal. The trees will then expand and try to connect each other. For the RRT with the goal node a random grasp configuration is chosen and a call to the randomized IK solver is performed.

Time until First Solution *(sec)*	Nº of Expansions until First Solution	ε_{final}	Nº of Expansions until Final Solution
0.31	182	3	8.161
0.15	76	3	7.584
0.33	182	3	6.265
2.01	544	5	5.021
1.07	379	4	7.991
0.98	432	4	6.445
14.88	6.773	100	6.785
0.56	31	3	6.714
0.57	34	3	5.960
1.06	322	5	4.932
0.14	62	3	6.437
0.13	68	3	6.437

Table 4.3: Results from 12 simulated trials for the dual-arm experiment [25]

After experiments on real life scenarios both methods proved themselves capable of yielding a feasible path, however the IK-RRT approach performs better than the Jacobian based planner.

4.4 Motion planning with Moving Obstacles

Lastly we will discuss the work of Tsai et al. [27] where the main goal is to solve the dynamic motion planning problem. This paper presents a stimulating approach in dealing with moving obstacles, where time is added as a new dimension to the configuration space, turning it into the configuration-time (CT) space. In this space only the moving - dynamic - obstacles will change position over time so if the path planned is kept far away from these their influence will be decreased. The method proposed is thus a CT-RRT planner, which is a bidirectional RRT that takes the time dimension in account. For dual-arm planning if the goal is in the shared workspace of both arms one arm computes the trajectory while considering the other as a moving obstacle.

Throughout this chapter some previous work relevant to our project was reviewed, however, as we said before, the literature is vast and the works of Srivastava et al. [28], Saut et al. [29] or Kurosu et al. [30] among many others could also have been mentioned. Finally it is important to notice that neither of the works explained, as well as the ones found throughout the research, solve or tackle the same problem we aim to. In the cases explained the manipulator was either to simple [24], the cases tested were too restricted [25], the focus was on the re-grasping task [26] or on moving obstacles [27]. Neither of them seeked to compare the already existing planners and analyze their performance in the execution of different task without the need to tune any specific motion.

Chapter 5

Implementation and Experiments

So far we have reviewed the theoretical grounds for this project, so now we have all the foundations to start the work itself: finding a motion planner capable of planning a collision free path for each arm of the manipulator. This chapter is divided in two main parts, the first one briefly exposes the programs used to simulate a functional robot and the second addresses the experiments carried out.

5.1 Workbench

To simulate a functional robot it is first necessary to create the robot's model, then there needs to be a way for it to communicate - exchange joint values, position in the environment, etc. -, finally there has to be a tool that implements the motion planning. To accomplish this several different programs are used:

- Robotic Operating System (ROS) [31] - open source code that has already implemented several key feature for robot programming, such as Robot Description Language, Standard Messaging or Diagnostics Tools;
- Gazebo [32] - free for use robot simulator with robust physics engine and high-quality graphics. Together with ROS can establish a movable simulated robot;
- MoveIt! [33] - open source motion planning framework, which is already prepared to work alongside ROS. MoveIt provides many tools useful in motion planning, spearing the user from wasting valuable time reimplementing certain feature. One of these tools is its vast offer of motion planners such as:
  - Open Motion Planning Library (OMPL) [34] - vast library of planners that provides several planning algorithms, from PRM, RRT to KPIECE or EST - mentioned in Chapter 3.

To simulate a dual arm manipulator with 6 DoF in each arm a UR5 robot model is used to represent each arm. Due to its popularity this specific robot entails a vast documentation, therefor

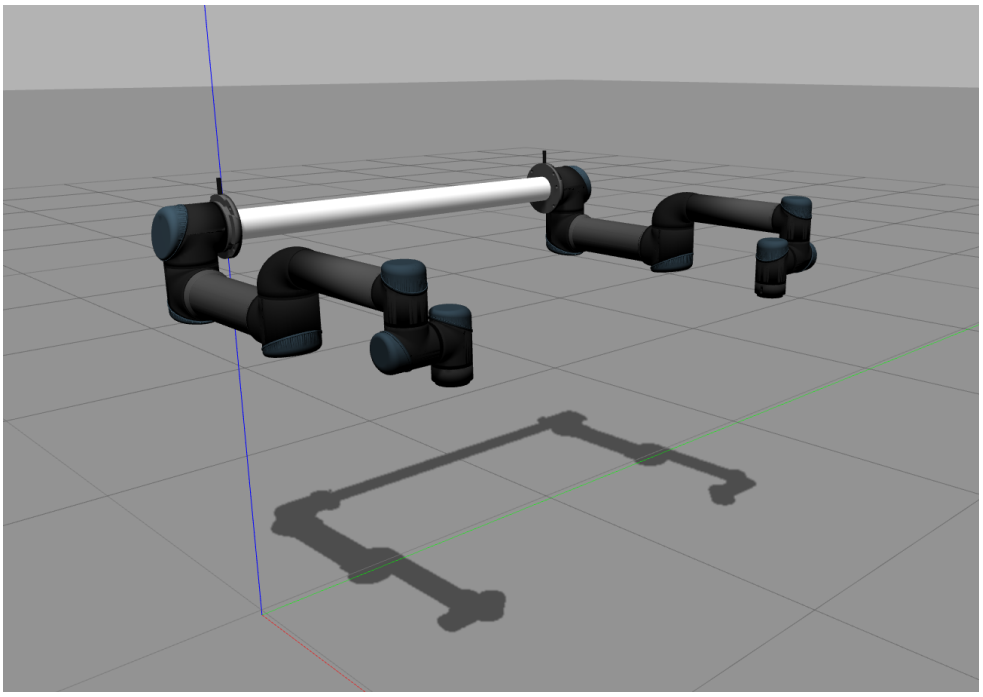


Figure 5.1: Simulated dual arm manipulator

an open source code was adapted and reconfigured to replicate the two arms. The robot's body is represented by a cylinder connecting the two arms. Each arm has a length of about 0.9m and are separated by 1m long cylinder. The final model can be seen in Figure 5.1.

To sum up, we use ROS to handle the robot, Gazebo for the simulation and MoveIt in collaboration with OMPL to motion plan.

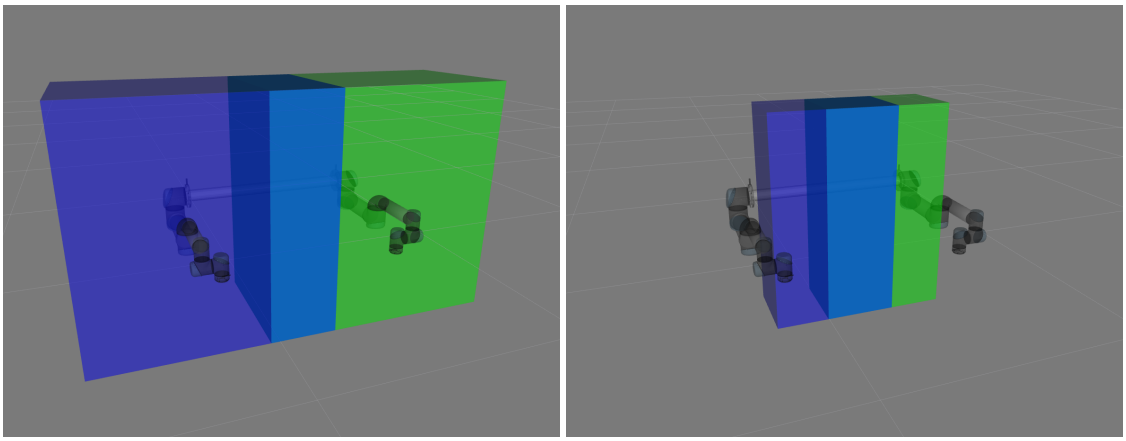
5.2 Experiments

Having set the robot's model as well as all the necessary tools to simulate and plan the wanted path it is now time to start comparing the performance of several motion planners when facing different scenarios.

Due to the extensive number of available planners a selection has to be done, therefor, after considered thought, the following six planners presented themselves as the most relevant for the mentioned reasons:

- RRT - since they were specially designed for handling nonholonomic constraints and high degrees of freedom manipulators;

- RRT-Connect - yields competitive results to RRT in less time and was conceived to handle arms with high DoFs - such as ours;



(a) Experiment A, C (b) Experiment B, D, E and F

Figure 5.2: Experiment's feasible end-effector's final position for the left (blue) and right (green) arm

- PRM - although it is not commonly used when kinematic constrains are present, it is nevertheless interesting to explore its behaviour when facing a non typical roadmap problem. The search algorithm used is A*;
- PRM* - similar to PRM yet converges to the optimal path. This can be proved useful for faster computations. The search algorithm used is A*;
- EST - given it combines the concept of roadmap with the construction of two motion trees, this method provides a different, though overlapping, approach of PRM and RRT-Connect ;
- KPIECE - this algorithm was specially designed for systems with complex dynamics, where physics-based simulation is necessary. Although it is not very widely known it addresses the motion planning problem in a distinct way.

To test the dual manipulator six different experiments will be carried out, these six experiments can be severed in two major groups relating to the movement of the arms. In experiment A, B and E the planning and movement of the right arm is executed and only if this succeeds will the left arm's path be planned and executed, as for experiments C, D and F both arms move at the same time. The experiments are thought so the feasible set of the end-effector's final poses will be reduced, promoting the possibility of collision or crossing of the arms over one another; for an easier visualization the sets for each experiment are represented in Figure 5.2. Every test is made under the same conditions, there is a range of x, y, z - position - and $roll, pitch$ and yaw angle - orientation - values for the end-effector, within the established scope a number of random points is selected. The robot's initial position is always the same, with all its joints set to zero (as seen in Figure 5.1) and for each point all six planners are tested. Table 5.1 registers the condition of each experiment.

For all evaluations both the planners outcome and planning time are registered, however, because of MoveIt!'s implementation methods when both arms move at the same time - and are

planned also at the same time - there is no way of knowing which arm led to the failure of the algorithm. Thus, for the both arms movement only the overall outcome and planning time are registered, whereas for the separate planning each arm performance and planning time is noted, which allows to further on compute the final outcome and planning time. The maximum allowed planning time is set to 5 seconds after this time the planner will yield failure.

Experiment	Type of movement	No of points tested	x	y		z	$roll$	$pitch$		yaw	
				Right Arm	Left Arm			Right Arm	Left Arm	Right Arm	Left Arm
A	Right Arm First	500	[-0.1;0.7]	[0.2;1.9]	[-0.8;0.9]	[-0.7;0.7]	0	0	0	$-\frac{\pi}{2}$	$\frac{\pi}{2}$
B	Right Arm First	500	[0.2;0.6]	[0.3;1]	[0;0.7]	[-0.6;0.6]	0	0	0	$-\frac{\pi}{2}$	$\frac{\pi}{2}$
C	Both Arms	500	[-0.1;0.7]	[0.3;1.7]	[-0.7;0.7]	[-0.7;0.7]	0	0	0	$-\frac{\pi}{2}$	$\frac{\pi}{2}$
D	Both Arms	500	[0.2;0.6]	[0.3;1]	[0;0.7]	[-0.6;0.6]	0	0	0	$-\frac{\pi}{2}$	$\frac{\pi}{2}$
E	Right Arm First	500	[0.2;0.6]	[0.3;1]	[0;0.7]	[-0.6;0.6]	0	$\frac{\pi}{2}$	$\frac{\pi}{2}$	0	0
F	Both Arms	500	[0.2;0.6]	[0.3;1]	[0;0.7]	[-0.6;0.6]	0	$\frac{\pi}{2}$	$\frac{\pi}{2}$	0	0

Table 5.1: Experiments' details

5.3 Experiments' Results

Following the implementation explanation we will now present the results for the experiments detailed before, some conclusions withdrawn from each test will be exposed and whenever possible visual representation will be displayed.

5.3.1 Experiment A

As it can be seen in Table 5.1 this experiment not only moves one arm at a time but also has a very wide feasible set of end-effector positions, this leads to a very low number of cases of possible collisions. However, it allows us to perceive each planners behaviour when presented with more simple scenarios. Before starting to state the results yielded from this experiment it is important to remember that for this type of movement - one arm at a time - the right arm is first planned and then moved, and only if this succeeds will the left arm be planned and moved, so, when talking about the left arm's success and failures we will only be referring to the cases where the planning is actually carried out, i.e. when the right arm has previously succeed.

From the 500 points tested a total of 73 are impossible to plan to, either because they are out of reach for the specified arm or because they are in collision with the robot's body or arm. These points automatically lead to a decline in the maximum possible success rate, being that for the right arm only 91.60% of those points are reachable and for the left one 93.67%, which means that overall, for both arms, the maximum possible success rate is 85.80%. The values that can be consulted in Table 5.2 replicate the results concerning the success rates for this experiment, these are already scaled to consider only the possible cases. Columns 1, 2 and 3 register the success rate for the right and left arm and the final overall performance, respectively. The overall outcome entails the cases where both arms managed to succeed,

Analyzing Table 5.2 in more detail we can see that all the planner show very similar success rates, never deviating more than 6% from the lowest and highest value. However, it is clear that the PRM* presents the best result for both the right and both arm, as for the left arm it is not the most successful but still presents competitive results. On the other hand PRM exhibits the worst results for all three cases - right, left and for both arms. This is due to the way each algorithm works, as explained before PRM creates a randomly sampled roadmap and tries to connect each node to a fixed number of neighbours as for PRM* it gradually increases the number of connection attempts as the roadmap grows in a way that provides convergence to the optimal path, this leads to a more concentrated roadmap.

For an easier visualization of the results Figure 5.3 represents the failed cases of each planner for the right arm. In accordance with what was already stated it is obvious that PRM* has the lowest number of failed points. We can also perceive that there are a lot of common points for all planner, this is due to the impossible cases already mentioned. It is clear that all planners present great difficulty when planning for the boundaries of the defined feasible set of possible final points and that there seems to be a cluster of points surrounding the initial position of the arm. A visualization of the failed cases for the left arm is not present due to the challenge of

Planner	Right Arm	Left Arm	Overall
RRT	82,97%	84,28%	69,93%
RRTConnect	82,53%	84,17%	69,46%
PRMStar	86,68%	84,44%	73,19%
PRM	81,66%	81,07%	66,20%
KPIECE	82,31%	85,80%	70,63%
EST	83,19%	85,46%	71,10%

Table 5.2: Experiment A - Success Rate (%)

representing, for all failed points, the end-effector of the left and right arm, since the latter clearly influence the planning of the former.

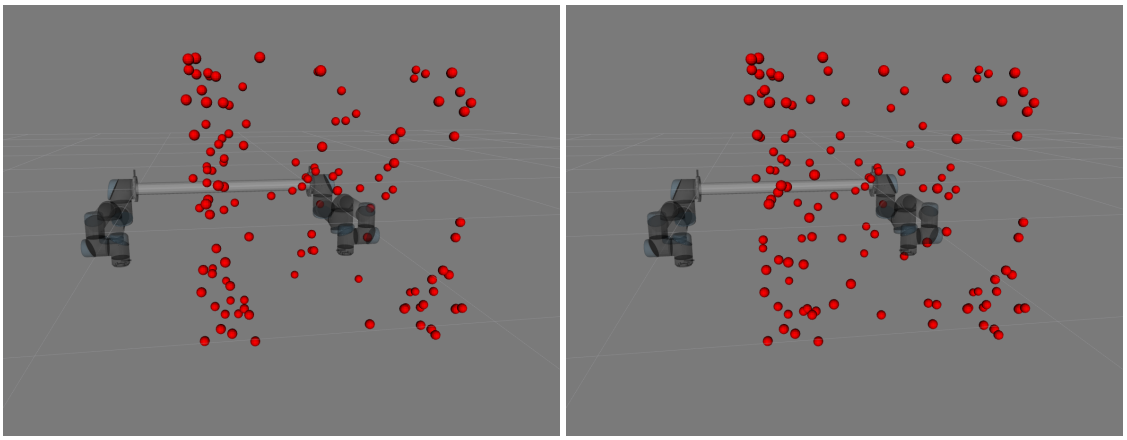
The success rate is, without a doubt, a crucial factor in a planner's evaluation, however it is not the only one. The time it takes to compute a path, also known as the planning time, must also be considered. Because of that, Table 5.3 allows us to compare each planner's planning time, being that the planning time when each planner succeeded is noted for each arm. The *Overall* column is the average planning time of the whole process, i.e., the time it took to plan a path for the right plus the left arm. We can clearly conclude that, although it presents exceptional success rates, PRM*, alongside PRM, presents the longest planning time. This phenomenon was expected since for each query both planner have to create a roadmap. Another aspect that is worth to notice is that, when successful, RRT Connect does take less time to compute a path than RRT. We can now infer that judging by the planning time alone RRT Connect would be the way to go, although any of the other three planners, RRT, KPIECE or EST, could also yield fast results.

5.3.2 Experiment B

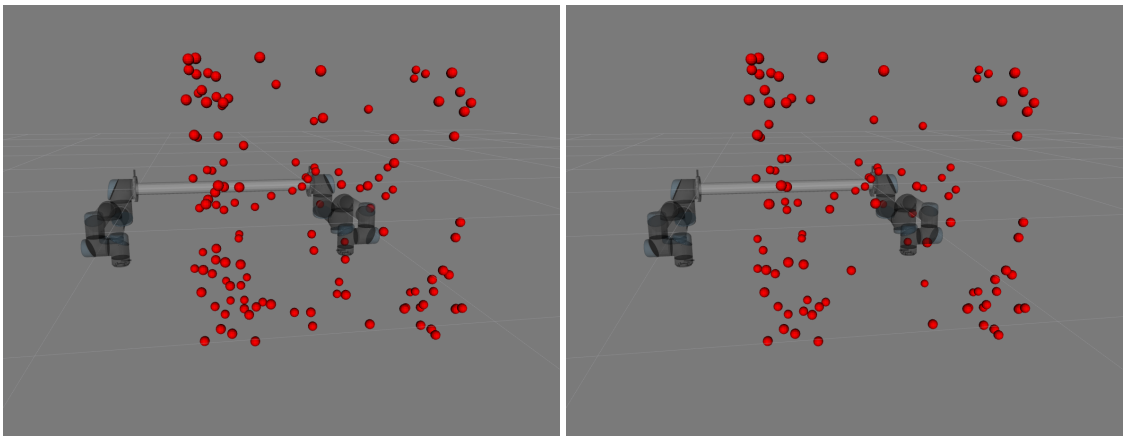
Experiment B shares a lot of common ground with the previous one, differing only in the feasible set of possible final positions. By making this set smaller in this test we want to increase the occurrence of more difficult scenarios, by doing so we also decreased the number of unreachable, therefor impossible, cases, adding these to a total of 43 of the 500 points tested. As we explained before this affects the maximum possible success rate, and once again the values in Table 5.4 already rule out this cases.

Planner	Right Arm	Left Arm	Overall
RRT	0,053	0,056	0,110
RRTConnect	0,038	0,038	0,076
PRMStar	5,023	5,023	10,046
PRM	5,013	5,014	10,026
KPIECE	0,102	0,103	0,204
EST	0,085	0,084	0,167

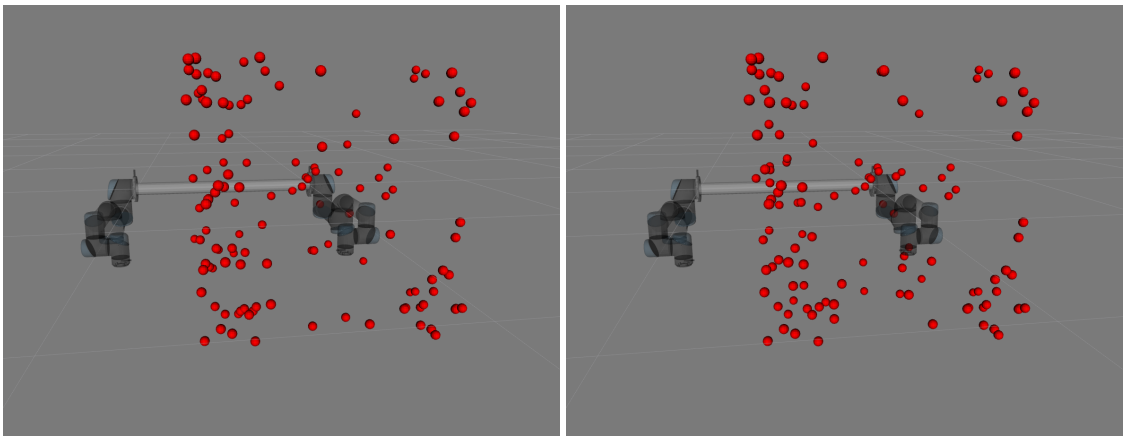
Table 5.3: Experiment A - Average Planning Time (seconds)



(a) RRT (b) RRT Connect



(c) PRM (d) PRM*



(e) KPIECE (f) EST

Figure 5.3: Experiment A - right arm failed points

Planner	Right Arm	Left Arm	Overall
RRT	87,63%	88,40%	77,46%
RRTConnect	87,21%	85,56%	74,62%
PRMStar	89,94%	86,13%	77,46%
PRM	88,68%	87,35%	77,46%
KPIECE	88,05%	85,49%	75,27%
EST	89,10%	84,73%	75,49%

Table 5.4: Experiment B - Success Rate (%)

Once we start to inspect Table 5.4 we can notice, like in Experiment A, that all success rates are very similar between planners, having a maximum difference of 4% form the lowest to the highest rate. This allows us to conclude that there is not a planner that stands out for the task given, at least when it comes to the success rate. However we can, nevertheless, point out that RRT Connect produced the worst outcome for the right and both arms case. Another interesting fact is that RRT, PRMStar and PRM present the exact same success rate for the overall result, making them the most effective planners.

Like in Experiment A Figure 5.4 shows, for each planner, the points in which the right arm's planning failed. All six figures are alike and we can see a clear concentration of failed points in the left boundary of the planning space, being that the cluster present in the top and bottom left, common to all planners, are unreachable points for the arm.

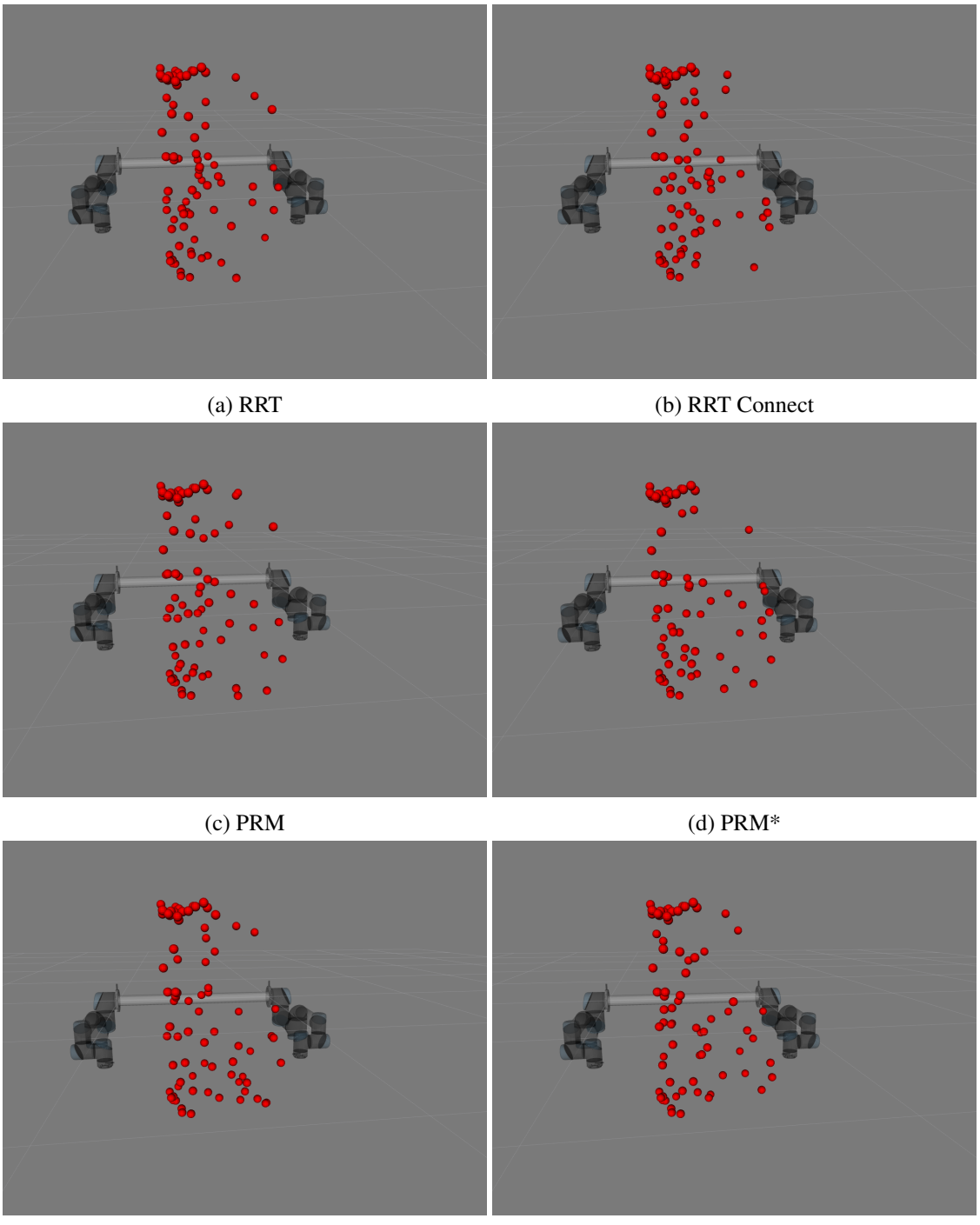
As for the average planning time it is detailed in Table 5.5, where we can see that, as expected, the planning times are very close to the ones presented in Table 5.3 relating to Experiment A. Once again both PRM and PRM* performed poorly, taking about 5 seconds to plan for each arm - which relates to the time to create the roadmap. All other planners present a planning time of circa 1 second.

5.3.3 Experiment C

All the experiments mentioned so far planned and moved one arm at a time, this is the main difference between experiment C and A. In this test both arms are planned and moved simultaneously, which entails a greater problem since when one arm is being planned it has to take in account that there is a moving object in its surroundings - the other arm. It is therefor expected that this

Planner	Right Arm	Left Arm	Overall
RRT	0,062	0,068	0,130
RRTConnect	0,045	0,045	0,089
PRMStar	5,023	5,025	10,047
PRM	5,013	5,013	10,026
KPIECE	0,119	0,120	0,236
EST	0,107	0,116	0,223

Table 5.5: Experiment B - Average Planning Time (seconds)



(a) RRT (b) RRT Connect

(c) PRM (d) PRM*

(e) KPIECE (f) EST

Figure 5.4: Experiment B - right arm failed points

Planner	Success Rate
RRT	66,19%
RRTConnect	64,99%
PRMStar	67,63%
PRM	66,67%
KPIECE	66,19%
EST	66,19%

Table 5.6: Experiment C - Success Rate (%)

experiment will yield worse results than the previous ones. In order to be able to perform a fair comparison experiment C has the exact same query points as experiment A.

It is important to keep in mind that for this type of planning is not possible to assign the failure to either of the arms, so there is only one success rate for the overall outcome. Table 5.6 registers the efficiency of each planner. Before further analyzing the results it is necessary to explain the disparity between the maximum possible success rate in this experiment and the one presented in Table 5.2 for experiment A. Although the points are exactly the same the number of impossible cases is not, simply because, as explained before, for experiment A the left arm is only planned if the right arm's planning is successful, so there are some impossible cases for the left arm that are not accounted due to the failure of the right arm's planning. When both arms are planned at the same time all impossible cases are considered, justifying the lower maximum possible success rate in this experiment.

As it was expected this test exhibits a worse performance than the one achieved in experiment A, some conclusions are nonetheless alike. Once again the efficiency of each planner is almost identical and, analogously to experiment A, PRM* proves to be the most successful planner. However RRT Connect replaces PRM in experiment A as the less reliable.

Although this type of movement falls short compared to one at a time arm movement when we analyze the average planning time for this experiment - Table 5.7 - we can conclude that it yields far better results for PRM and PRM*, since there is no need to create two roadmaps - one for each arm. However there is a slightly slower response for all other planners, this is due to the higher complexity of the problem; not only it has to plan for two objects but each one of them has a series of joints that must be taken in account.

Planner	Success
RRT	0,144
RRTConnect	0,071
PRMStar	5,029
PRM	5,028
KPIECE	0,183
EST	0,152

Table 5.7: Experiment C - Average Planning Time (seconds)

Planner	Success Rate
RRT	70,42%
RRTConnect	71,96%
PRMStar	72,41%
PRM	73,95%
KPIECE	69,54%
EST	73,73%

Table 5.8: Experiment D - Success Rate (%)

Finally there is another interesting evaluation that must be done, that is the analysis of which points were successful for experiment C but not for A. Between all planner there is a total of 224 points where this occurs - not common for all planners though. To verify if the failed planner is actually unable to plan to these points another test has to be done, where each planner is allowed to re-plan the points it failed when a successful outcome was possible. After this experiment the number of failed points dropped from 224 to 63, this new outcome can be explained by the nature of the planners, i.e. all of them rely on a sampling process, which will lead to different outcomes for the same query points.

5.3.4 Experiment D

Analogously to the first two experiments there is also a need to test the planner's behaviour when both arms have to move at the same time, but now in a smaller set of feasible final positions. For the same purpose that experiment C had the same query points as A so does experiment D uses the exact same query points as experiment B and can then be seen as a combination of experiment C (sharing the same type of movement) and B (using the same query points).

Analyzing Table 5.8 it is easy to grasp the similarities between experiment B and D when it comes to success rate. In the table it is visible that PRM* reveals itself as the most effective. However in this test RRT did not yield such a satisfying result, although it is not the worst planner it takes the second worse, being outran by KPIECE.

Given that the type of movement is the same as in experiment C it is expected that the average planning times will be identical, that can be confirmed in Table 5.9 where the average planning time for each planner is registered. Once again PRM and PRM* present the longest planning times, although, as it happened in experiment C, they achieve faster times than when planning for single arm movement. We can also confirm a consistent increase in the planning time of the other planners due to the higher complexity of the problem to solve.

It is notorious that there is a parallelism between experiment A and B and experiment C and D, since in both pairs there is an increase in the overall success rate. There is also a parity between experiment C and D concerning the average planning time.

Planner	Success
RRT	0,229
RRTConnect	0,083
PRMStar	5,030
PRM	5,025
KPIECE	0,239
EST	0,188

Table 5.9: Experiment D - Average Planning Time (seconds)

5.3.5 Experiment E

The previous experiments test the planners moving separately and combined in a large and small set of feasible final positions, however they all have one thing in common, the same end-effector final orientation. Experiment E serves to evaluate the planner's behaviour when facing similar situations but with a different orientation.

Although the former experiments have tested two different sets of possible final positions, experiment E and F deal only with the smaller set since, in most real life situation, most of the work is done in this space.

Given that this experiment deals with single arm movements the values of each arms' planning performance alongside the overall outcome are registered and are here reproduced in Table 5.10. First of all is important to enhance the proximity in success rates, never deviating more than 2% from one another. This fact hinders the comparison between planners since there is not a clearly superior or inferior one. Second we can also see, as it has been for other experiments, the best planner for the right arm - RRT Connect - is not the same for the left - PRM. As for the overall outcome PRM confirms to be the best planner, whereas PRM* and EST prove to be the worst - although the success rate is only 2% inferior.

In order to understand the reason behind such a low success rate - when compared to experiment B - the points where the right arm failed are represented as red dots in Figure 5.5. This visualization aids in understanding not only where each planner failed, but also in verifying if there is any consistency between planners. For example, it is clear that there is a cluster of failed points in the top left corner, this is explained by the lack of capacity to reach these points given the final orientation set, which also explains the common failure for all planners. It is likewise in-

Planner	Right Arm	Left Arm	Overall
RRT	83,16%	80,53%	66,96%
RRTConnect	83,37%	80,32%	66,96%
PRMStar	81,88%	80,15%	65,63%
PRM	82,09%	82,39%	67,63%
KPIECE	81,45%	81,94%	66,74%
EST	81,66%	80,36%	65,63%

Table 5.10: Experiment E - Success Rate (%)

Planner	Right Arm	Left Arm	Overall
RRT	0,086	0,131	0,225
RRTConnect	0,041	0,040	0,082
PRMStar	5,028	5,025	10,053
PRM	5,024	5,019	10,042
KPIECE	0,172	0,184	0,347
EST	0,120	0,166	0,287

Table 5.11: Experiment E - Average Planning Time (seconds)

teresting to underline the impact that the final orientation has in the points that can be reached by the manipulator, since for experiment B reaching points in the upper space is not so problematic. Another fact that we can see is that there are few points in the middle of the work space where the planners fail to plan to, being that KPIECE presents the higher number of failed points in this space corroborates its poorer performance.

The average planning time - present in Table 5.11 - however, exhibits several discrepancies, specially for PRM and PRM*. This distinction has been consistently observed in all experiments and is justified by the development of a roadmap for each arm on each query point. The other planners also show a regular value, never exceeding an average of 1 second on the overall planning time for single arm movement.

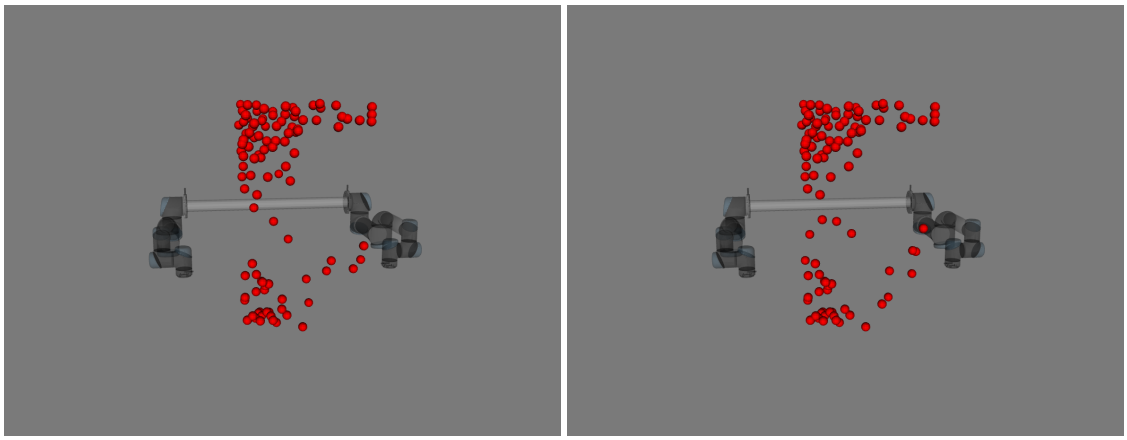
5.3.6 Experiment F

For the sake of comparing the outcome of each planners between dual and single arm movement experiment F is executed with the exact same points as in experiment E, differing only on the type of movement.

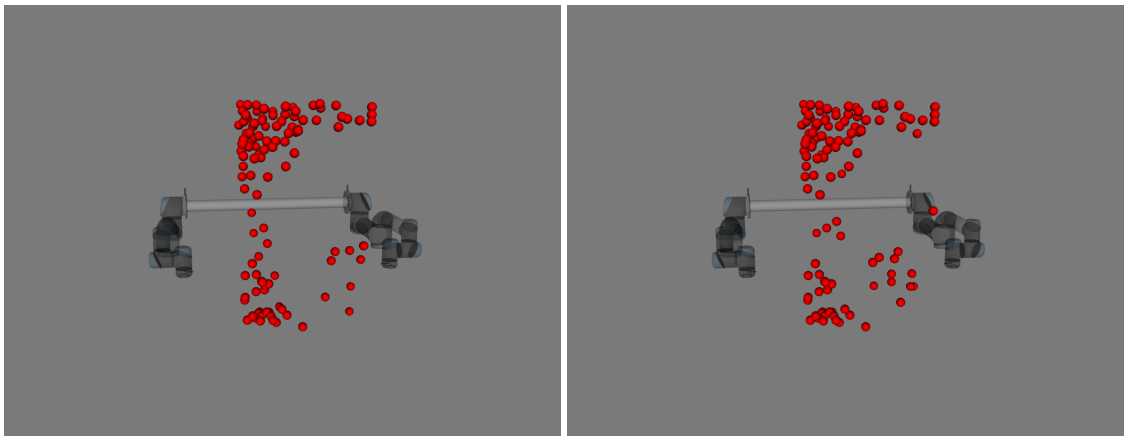
Table 5.10 not only tells us the success rate for each planner but also allows us to observe a decline in the success rate in view of experiment E. This is an expected observation due to the higher complexity of the problem, it is also consistent with the decline documented between experiment B and D. There is a difference between experiment E and F though, the planner with the best success rate changes from PRM to RRT Connect and neither KPIECE or PRM* are the worst planner but rather RRT.

Planner	Success Rate
RRT	58,88%
RRTConnect	64,27%
PRMStar	63,82%
PRM	63,60%
KPIECE	59,33%
EST	63,60%

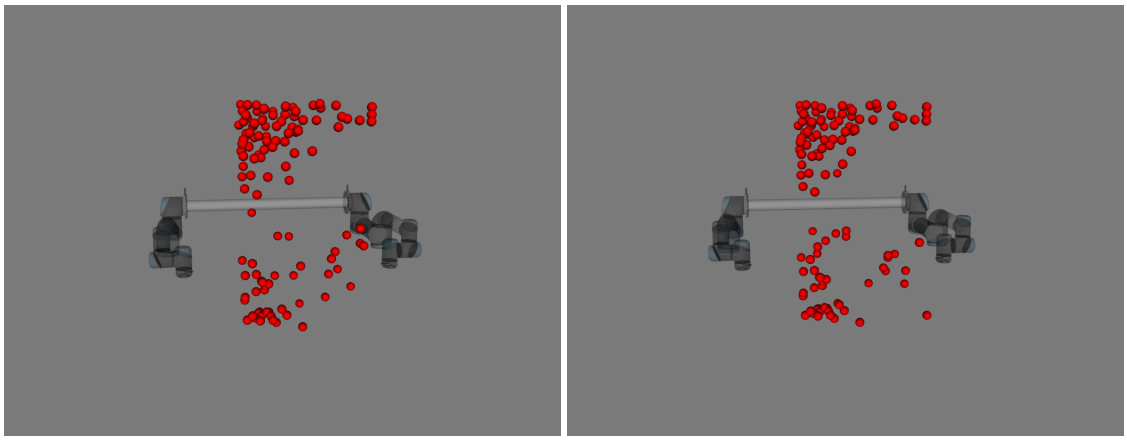
Table 5.12: Experiment F - Success Rate (%)



(a) RRT (b) RRT Connect



(c) PRM (d) PRM*



(e) KPIECE (f) EST

Figure 5.5: Experiment E - right arm failed points

Planner	Success
RRT	0,316
RRTConnect	0,096
PRMStar	5,031
PRM	5,024
KPIECE	0,392
EST	0,314

Table 5.13: Experiment F - Average Planning Time (seconds)

As for the planning time - Table 5.11 - we can state that it is, as expected, faster than for the single arm movement and is in accordance with the planning times for other experiments that move both arms at the same time. The planning times for the algorithm that require the development of a roadmap are just a bit above 5 seconds, as for all other planners the overall planning time does not exceed 1 second. This leads us to conclude that a higher planning time leads to a better success rate, this is not only expected but is also the core of sampling based algorithms, since these are probabilistic complete, i.e. the probability of finding a solution converges to one as time goes by.

5.4 Summary

To sum up, after a series of experiments on a free object environment, where different final poses were tested we can conclude that all planners can, to a certain degree, handle dual arm manipulators, with 6 Dofs on each arm. Although it is possible to distinguish for each experiment a better and worse planner the disparity on the success rates are not significant enough to outline a definite conclusion when it comes to choosing the most effective planner.

It was also possible to observe the decline of the success rate when both arms must move at the same time, thus translating in a more complex problem.

Finally, regarding the average planning time, it was observed that RRT, RRT-Connect and EST present similar outcomes, whereas PRM and PRM*, by being solely based on roadmaps, take the maximum available time to compute a path.

Chapter 6

Real Life Scenarios' Simulation

After being established that all chosen planners can, to a certain degree, compute a feasible, collision free plan for a dual arm manipulator it is now time to appraise each planners performance when facing real life like scenarios.

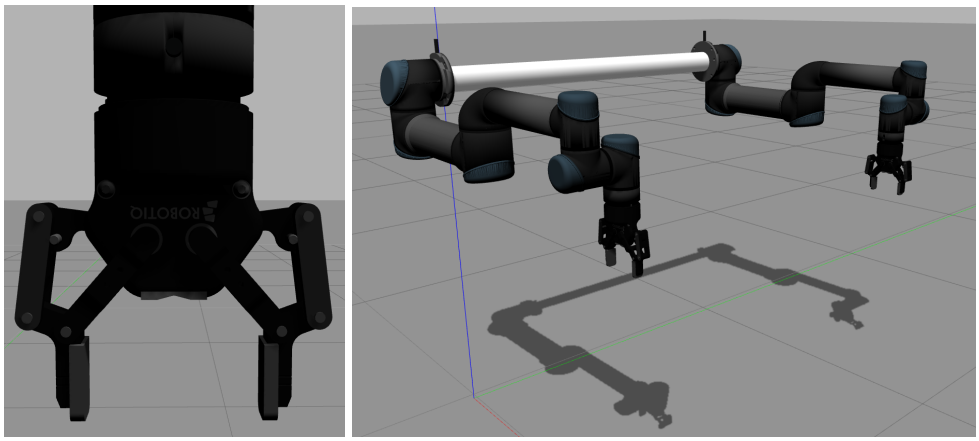
The approach taken in this chapter is similar to the one in the experiments, first the scenarios tested will be explained and only then will the results be presented and discussed.

6.1 Scenarios

The results in Chapter 5.3 prove that the planners chosen can, in fact, deal with a dual arm manipulator, however, in real life, one does not seek to plan in an obstacle free environment, usually manipulators are surrounded by others objects that make the motion planning problem harder. Thus there is a necessity to simulate real life scenarios where the manipulator must avoid and interact with several objects to evaluate if the planners can nonetheless yield precise results.

First of all, it is mandatory to implement a tool that can manipulate the objects that the arms have to interact with. To solve this problem a 2 finger gripper - Figure 6.1 - is attached to each arm end effector, when one seeks to grasp an object each joint of the gripper is actuated, leading the fingers to move together and thus grasping the object, to release the reverse process is executed.

One of the most common operations performed with robotic arms is the pick and place action, the gripper grabs an object at a certain pose and aims to place it at different one. Due to the wide spread usage of this type of motion the first real life scenario simulation - for mentioning purposed will be called Scenario A - consists in replacing six cubes, all with the same size, placed on top of a table that is 60 cm from the ground and 40 cm away from the manipulator's body. The initial and final position of each cube is depicted in Figure 6.2, the number associated with the cubes represent the order in which they are picked and placed, being that the left manipulator grasps the odd numbered cubes and the right the even ones. It is relevant to state that, since the focus of the project is not on the grasping action but rather on the motion the end effector's orientation and pose are selected manually. Although the final orientation is selected to increase the probability of success a set of different grasping poses are chosen in order to test each planners' capability



(a) Gripper (b) Dual arm manipulator with gripper

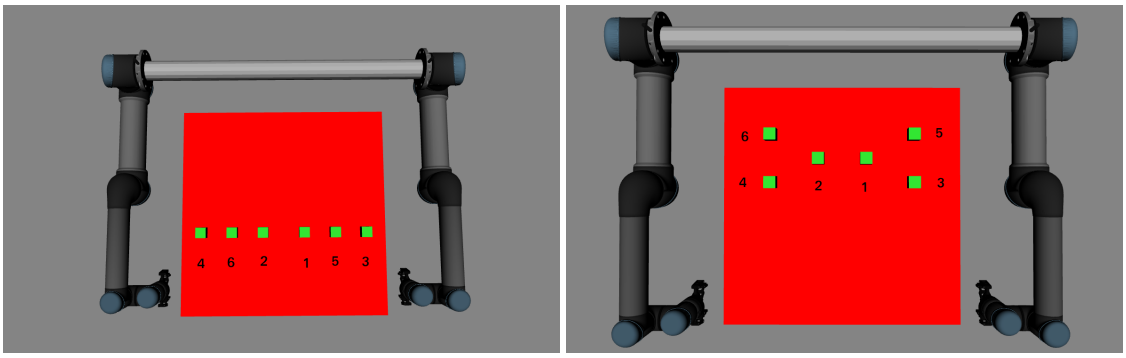
Figure 6.1: Gripper (left) and dual arm manipulator with gripper (right)

to plan for various final orientations. For an easier visualization each cube's grasping and picking pose is represented in Figure 6.3.

Scenario A represents a basic pick and place operation with small and identical objects, however real life situations usually include more complex and sundry objects. In the simulation the objects are generated by MoveIt! and this only allows the spawning of simple shaped obstacles (such as boxes and cylinders) so, for Scenario B, the cubes are replaced by two cylinders and boxes with different orientations. All objects have a height of 20 cm and a maximum width of 5 cm, considering that the same table mentioned in Scenario A is used this implies that the vertical cylinder and cube - objects 3 and 4 of Figure 6.4a - are only 20 cm apart in the z axis, thus reducing the free space available for the manipulators. Once again each initial and final configuration is depicted in Figure 6.4 - the same numbering system in Scenario A is used - also the picking and placing orientation can be seen in Figure 6.5.

For all scenarios, and analogously to the methodology of the experiments, two types of movements are tested. On the first one the right arm moves to the assigned object, than so does the left one, after both of them have grasped the respective object the right arm places the object at the final position and only then will the left arm do this procedure, this is repeated for all objects - the left side of Figure 6.6 illustrates the single arm movement algorithm diagram. The other movement type is the simultaneous motion - and therefor planning - of the two arms, they will pick or place two objects - one for each arm - for each planners' call, as shown in the right side of Figure 6.6.

Lastly it is vital to explain that each scenario simulation is ran 10 times, and for each planning attempt a total of 10 tries are allowed, i.e., to pick or place an object a planner can fail up to 10 times before the simulation is stopped. This is due to the fact that the planners are random sampling based, consequently the outcome will change for each call, which might lead to a failure on the first try - due to poor sampling - but a success on the next. Each planner has a total of 5 seconds before yielding failure to plan a motion path.



(a) Scenario A initial object placing (b) Scenario A final object placing

Figure 6.2: Scenario A - initial and final object placing. The number in each cube represents the order they are picked and placed, being that the odd numbers are handled by the right arm and the even ones by the left arm.

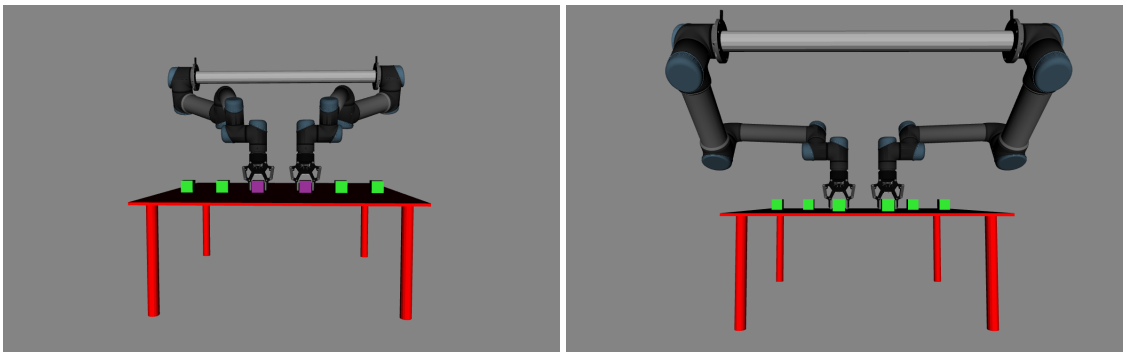
6.2 Results

The results registered for each scenario are presented throughout the next sections, first the outcome of the single arm movement (one arm at a time) is discussed and only after that is the simultaneous arm movement analyzed. For each type of movement two tables are produced, one registers the success rate for each path, being that each object has associated with itself two paths, picking and placing. The success rate for each path is relative to the number of successful plannings within the ones that were possible, i.e., if the placing of object 1 is 50% successful, and the next path has an 100% success rate it means that it succeed in all possible cases - in all simulation that reached that point. In the same table, next to the success rate, is indicated the average number of tries that it took to find the specific path. The final column - named *Overall* - registers the percentage of simulations that reached the end, meaning the number of times a planner was able to compute all paths. The second table indicates the average planning time it took to compute the path for the picking or placing of each object, this time is only indicated for the successful cases since the failed ones are due to an exceeding of the 5 second established time out. The symbol "-" means that the value is not registered because neither simulation got to that point, that is, the maximum number of tries was achieved and the simulation ended earlier.

6.2.1 Scenario A

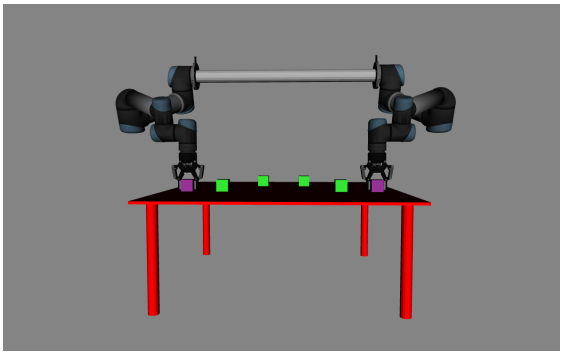
Scenario A is a pick and place action with simple and small objects, nonetheless, there single existence and the adding of the table - that greatly reduces the workspace - increase the problem's complexity considerably.

In the previous chapter we concluded that all planner yielded a similar success rate for all experiments, however this is not verified in this real life scenario. Analyzing Table 6.1 it is obvious that RRT behaves poorly, being that only 1 of the 10 simulations reached the end and it usually takes it more than one try to compute a motion plan, this is due to the fact that it only creates one motion tree, that is extended a maximum of ε per sampling - see Section 3.3.2 - thus, if given a

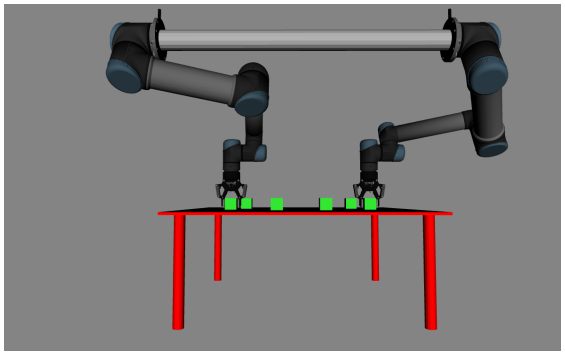


(a) Object 1 and 2 grasping position (front view)

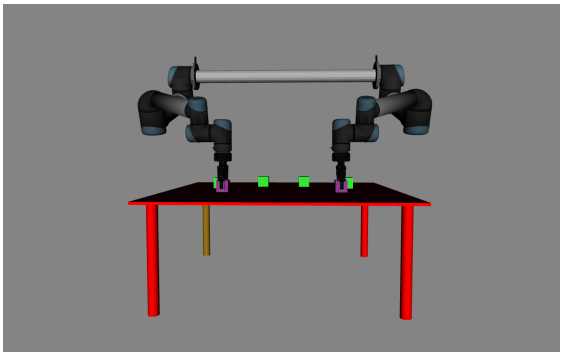
(b) Object 1 and 2 placing position (rear view)



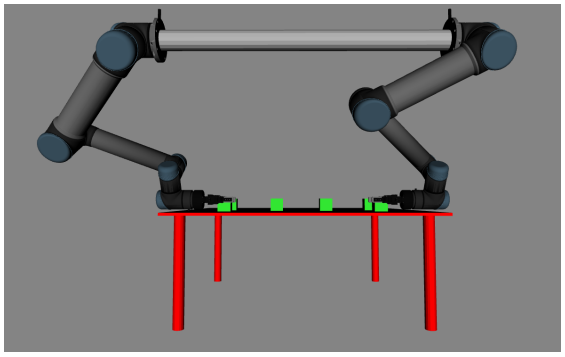
(c) Object 3 and 4 grasping position (front view)



(d) Object 3 and 4 placing position (rear view)

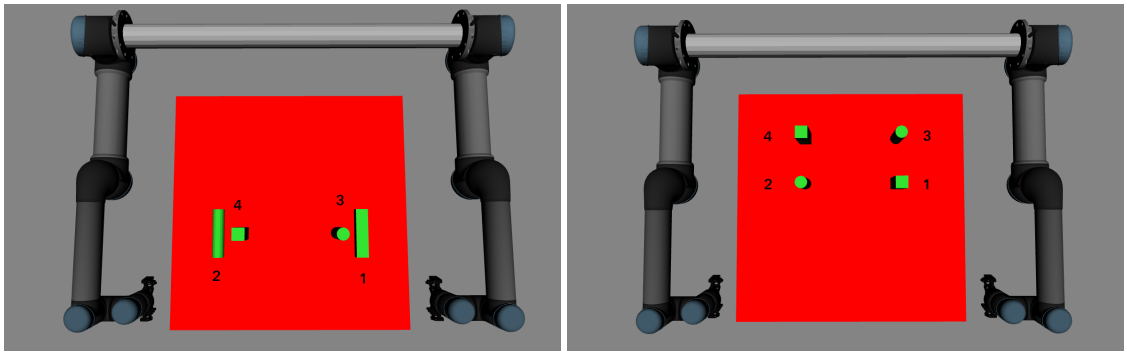


(e) Object 5 and 6 grasping position (front view)



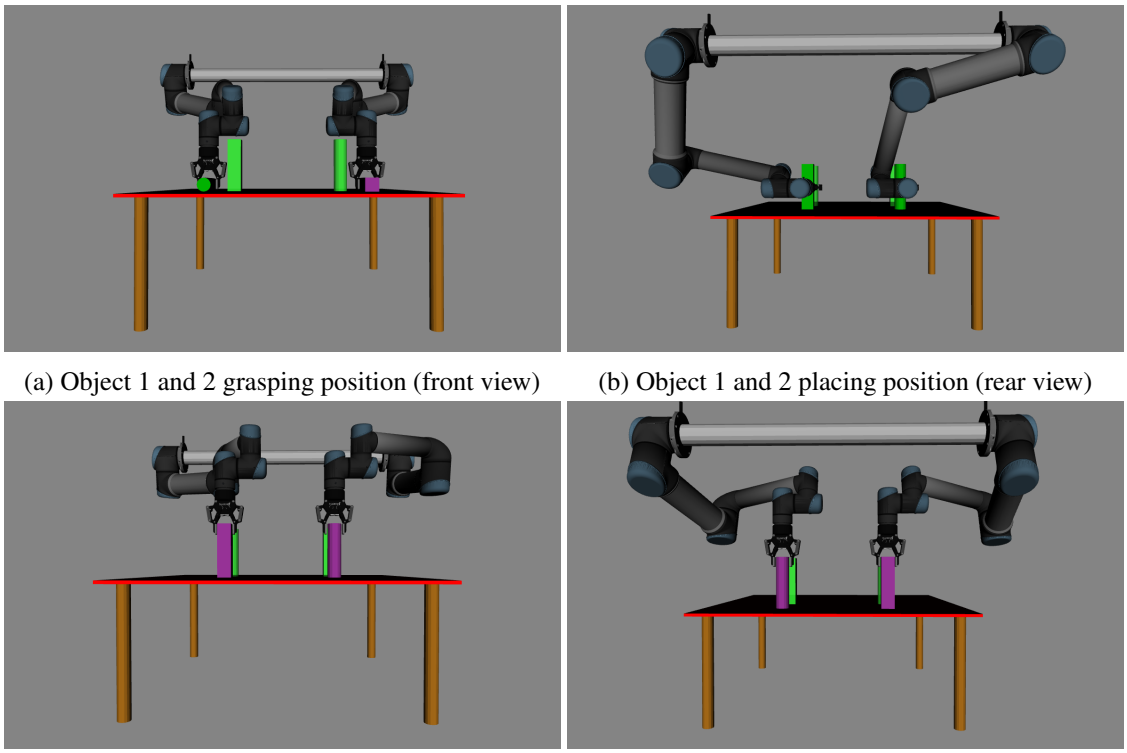
(f) Object 5 and 6 placing position (rear view)

Figure 6.3: Scenario A - Objects' grasping and placing position (the numbers correspond to the ones in Figure 6.2



(a) Scenario B initial object placing (b) Scenario B final object placing

Figure 6.4: Scenario B - initial and final object placing. The number in each cube represents the order they are picked and placed, being that the odd numbers are handled by the right arm and the even ones by the left arm.



(a) Object 1 and 2 grasping position (front view) (b) Object 1 and 2 placing position (rear view)

(c) Object 3 and 4 grasping position (front view) (d) Object 3 and 4 placing position (rear view)

Figure 6.5: Scenario B - Objects' grasping and placing position (the numbers correspond to the ones in Figure 6.4

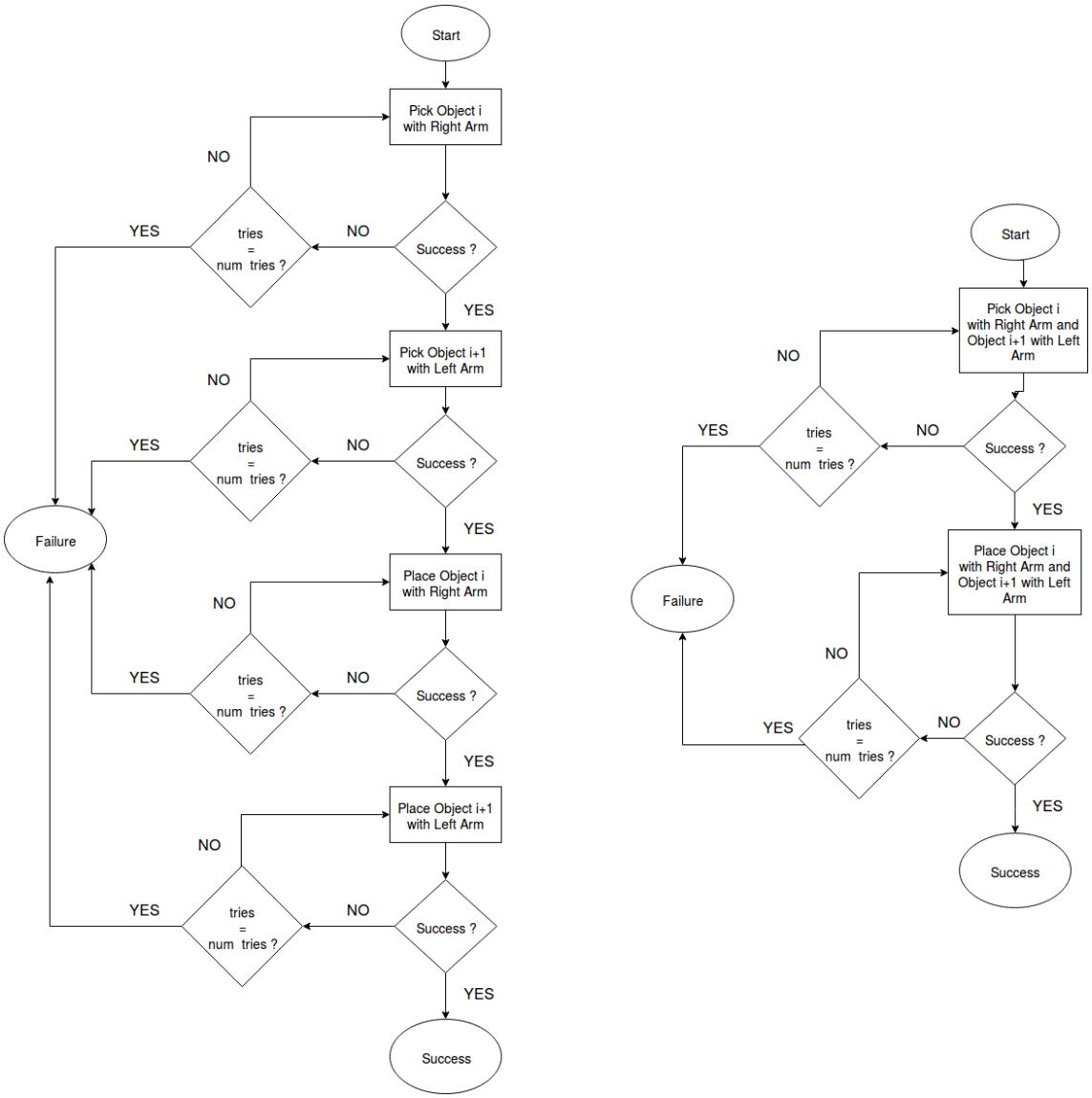


Figure 6.6: Single (left) and simultaneous (right) arm movement algorithm

longer planning time it may be able to compute a solution. On the other hand, both RRTConnect and PRM* were able to succeed in all simulation, still RRTConnect has a slight advantage when it comes to the number of tries it needs. As for KPIECE, PRM and EST the results are approximated, having a success rate between 70 and 90%, however KPIECE is more effective - takes fewer tries - than PRM and EST. It is interesting to notice that besides RRT all planners create either roadmaps or two motion trees.

The planning times of each algorithm are present in Table 6.2, where it can be noticed that, as observed before, both PRM and PRM* take about 5 seconds to compute a path, as for the other planners - that in Chapter 5.3 planned a solution in about 1 second - it can be seen a increased in the average planning time, taking sometimes more than 4 seconds to find a solution. This behaviour is expected and can be justified by the environment's enhanced complexity, this claim is supported

Planner	Picking Object 1		Picking Object 2		Placing Object 1		Placing Object 2		Picking Object 3		Picking Object 4	
	SR (%)	NoT	SR (%)	NoT	SR (%)	NoT	SR (%)	NoT	SR (%)	NoT	SR (%)	NoT
RRTConnect	100,00%	1,00	100,00%	1,00	100,00%	2,40	100,00%	2,70	100,00%	1,00	100,00%	1,00
RRT	100,00%	1,00	90,00%	1,22	66,67%	3,00	66,67%	2,50	75,00%	4,33	66,67%	2,50
KPIECE	100,00%	1,10	100,00%	1,00	100,00%	3,60	70,00%	3,00	100,00%	1	100,00%	1,00
PRM	100,00%	1,00	100,00%	1,00	100,00%	3,00	100,00%	1,60	100,00%	1,00	100,00%	1,10
PRMStar	100,00%	1,00	100,00%	1,00	100,00%	1,40	100,00%	1,60	100,00%	1,00	100,00%	1,00
EST	100,00%	1,00	100,00%	1,10	80,00%	2,25	100,00%	5,38	100,00%	1,00	100,00%	1,00

Planner	Placing Object 3		Placing Object 4		Picking Object 5		Picking Object 6		Placing Object 5		Placing Object6		Overall
	SR (%)	NoT	SR (%)	NoT	SR (%)	NoT	SR (%)	NoT	SR (%)	NoT	SR (%)	NoT	SR (%)
RRTConnect	100,00%	1,00	100,00%	1,00	100,00%	1,00	100,00%	1,00	100,00%	1,00	100,00%	1,50	100,00%
RRT	100,00%	5,00	100,00%	1,00	100,00%	2,50	100,00%	1,00	50,00%	1,00	100,00%	3,00	10,00%
KPIECE	100,00%	1,43	100,00%	1,29	100,00%	1,14	100,00%	1,00	100,00%	2,75	100,00%	4,86	70,00%
PRM	100,00%	1,30	100,00%	1,00	100,00%	1,30	100,00%	1,00	90,00%	1,22	100,00%	1,00	90,00%
PRMStar	100,00%	1,10	100,00%	1,00	100,00%	1,10	100,00%	1,00	100,00%	1,20	100,00%	1,70	100,00%
EST	80,00%	1,38	80,00%	1,38	100,00%	1,13	100,00%	1,00	100,00%	1,50	100,00%	2,63	80,00%

Table 6.1: Scenario A - Single arm movement Success Rate (SR) and average Number of Tries (NoT) it took to compute a path for picking and placing of each object

by a longer planning time for path that required more tries (e.g. the placing of object 6), which implies that the solution is harder to compute, thus taking more time to do it.

Moving on to the simultaneous movement of both arms the planners face now an even more challenging problem, since the arms have to move at the same time there is a greater number of constraints and a more complex kinematics to take into account, it is then expected an inferior success rate. This expectation is verified in Table 6.3, that exposes the success rate and number of tries for each path when the two arms have to move at the same time. Consistently to the results discussed before RRT's outcomes prove its incapability to solve the problem, in all simulation the number of tries was exceeded for the picking of the first to objects, not allowing the testing of the others. However there is a enormous disparity from the previous shown results, PRM* was not able to finish any of the simulations, although it has an 100% success rate for picking the first two objects it was never capable of placing them. Likewise KPIECE also behaved more poorly, being that the only case where it was able to place objects 1 and 2 was not able to place objects 5 and 6, thus falling in all simulations. As for RRTConnect, PRM and EST few of the simulation were carried out until the end, however the success rate dropped drastically.

Concerning the planning times for the simultaneous movement Table 6.4 indicates that once again PRM needs about 5 seconds to find a solution - the same for PRM* but it is not as relevant since we only have the result for one of the six wanted paths. RRTConnect, KPIECE and EST take more time to plan each path, which is obviously explained by the problem's arduousness.

After all this it is worth point out the influence that non free obstacle environments have in both the success rate and the planning time, specially in the simultaneous movement of the arms. Even in a simple scenario some planners proved to be completely ineffective and the ones that were not so revealed far greater planning times when compared to the experiments discussed in Chapter 5.3.

During the simulations it was noticed that several times the planner would output failure due

Planner	Picking Object 1	Picking Object 2	Placing Object 1	Placing Object 2	Picking Object 3	Picking Object 4
RRTConnect	0,13	0,15	1,74	1,86	0,14	0,14
RRT	1,80	1,24	4,45	3,05	1,99	1,75
KPIECE	0,52	0,88	4,05	2,06	0,31	0,37
PRM	5,02	5,03	5,52	5,88	5,03	5,03
PRMStar	5,05	5,04	5,71	5,77	5,02	5,04
EST	0,40	0,68	4,69	3,15	0,74	0,30

Planner	Placing Object 3	Placing Object 4	Picking Object 5	Picking Object 6	Placing Object 5	Placing Object 6
RRTConnect	1,76	1,51	0,20	0,19	1,17	3,67
RRT	2,21	0,44	1,57	0,38	5,08	3,05
KPIECE	3,16	4,11	1,23	1,33	3,28	5,39
PRM	5,71	5,65	5,03	5,03	5,58	5,76
PRMStar	5,84	5,47	5,03	5,04	5,39	6,00
EST	2,83	3,51	0,81	1,17	2,04	3,12

Table 6.2: Scenario A - Single arm movement average planning time (seconds) to compute a path for picking and placing of each object

Planner	Picking Object 1 and 2		Placing Object 1 and 2		Picking Object 3 and 4		Placing Object 3 and 4		Picking Object 5 and 6		Placing Object 5 and 6		Overall
	SR (%)	NoT	SR (%)	NoT	SR (%)	NoT	SR (%)	NoT	SR (%)	NoT	SR (%)	NoT	SR (%)
RRTConnect	100,00%	1,00	60,00%	5,33	100,00%	1,00	100,00%	1,00	100,00%	1,00	50,00%	3,33	30,00%
RRT	0,00%	-	-	-	-	-	-	-	-	-	-	-	0,00%
KPIECE	100,00%	1,00	10,00%	7,00	100,00%	1,00	100,00%	2,00	100,00%	3,00	0,00%		0,00%
PRM	100,00%	1,20	30,00%	6,67	100,00%	1,67	100,00%	2,00	100,00%	1,33	66,67%	8,50	20,00%
PRMStar	100,00%	1,00	0,00%	-	-	-	-	-	-	-	-	-	0,00%
EST	100,00%	1,10	30,00%	6,33	100,00%	1,00	100,00%	1,00	100,00%	4,67	33,33%	1,00	10,00%

Table 6.3: Scenario A - Simultaneous arm movement Success Rate (SR) and average Number of Tries (NoT) it took to compute a path for picking and placing of each pair of objects

Planner	Picking Objects 1 and 2	Placing Objects 1 and 2	Picking Objects 3 and 4	Placing Objects 3 and 4	Picking Objects 5 and 6	Placing Objects 5 and 6
RRTConnect	0,258	2,834	0,574	3,850	1,115	2,563
RRT	-	-	-	-	-	-
KPIECE	1,438	0,489	3,006	5,148	0,879	-
PRM	5,030	6,057	5,022	5,884	5,019	5,874
PRMStar	5,167	-	-	-	-	-
EST	1,357	3,817	1,117	3,394	3,086	4,901

Table 6.4: Scenario A - Simultaneous arm movement average planning time (seconds) to compute a path for picking and placing of each pair of objects

Planner	Picking Object 1 and 2		Placing Object 1 and 2		Picking Object 3 and 4		Placing Object 3 and 4		Picking Object 5 and 6		Placing Object 5 and 6		Overall
	SR (%)	NoT	SR (%)	NoT	SR (%)	NoT	SR (%)	NoT	SR (%)	NoT	SR (%)	NoT	SR (%)
RRTConnect	100,00%	1,00	60,00%	5,00	100,00%	1,00	100,00%	1,00	100,00%	1,17	100,00%	3,17	60,00%
RRT	20,00%	5,50	0,00%	-	-	-	-	-	-	-	-	-	0,00%
KPIECE	100,00%	1,40	70,00%	2,71	100,00%	1,00	100,00%	1,00	100,00%	1,86	57,14%	1,25	40,00%
PRM	90,00%	2,33	44,44%	4,50	100,00%	1,50	100,00%	1,25	100,00%	2,00	50,00%	2,00	20,00%
PRMStar	100,00%	3,40	20,00%	7,00	100,00%	1,00	100,00%	1,00	100,00%	2,00	100,00%	2,00	20,00%
EST	100,00%	1,30	60,00%	3,67	100,00%	1,00	100,00%	1,00	100,00%	1,83	66,67%	2,50	40,00%

Table 6.5: Scenario A - Simultaneous arm movement Success Rate (SR) and average Number of Tries (NoT) it took to compute a path for picking and placing of each pair of objects, with adapted parameters

to a contact - collision - between different robot's links. Although the planners already incorporate collision check for each configuration chosen there is a parameter that defines the maximum distance the robot can travel without the collision checker is called. This parameter was changed from 5 cm to 2 cm. This change entails an increase in the planning time since collision checking is a time consuming process. Therefor, and also due to the probabilistic completeness of the planners, the planning time was increased to 10 minutes. All the simulation were once again tested and the results acquired are depicted in Table 6.5. There we can see that there is a more effective performance for all planners. RRT-Connect double its success rate, KPIECE and PRM* were now able to carry out 4 and 2 whole simulations, respectively, as for EST the overall success rate is 4 times higher. Although RRT was not able to complete any of the simulations it was now able to pick object 1 and 2 two times. It can also be noticed a decreased in the average number of tries it took for each planner to compute a path, this is due to the fact that there were no failed tries due to collision.

As it was mentioned before it is expected that this set of tests, with adapted parameters, present greater planning times. This is corroborated by the values in Table 6.6 that represent the average planning time it took each planner to compute a path for picking or placing each pair of objects. Besides the picking of objects 5 and 6 RRT-Connect always presents an average planning time lower then 30 seconds, making it the faster of all planners. It is also interesting to notice that PRM and PRM* will always use the maximum amount of time possible, thus before taking circa 5 seconds and now the whole 10 minutes.

However the yielded results are not yet satisfactory, thus a new set of tests where executed.

Planner	Picking Objects 1 and 2	Placing Objects 1 and 2	Picking Objects 3 and 4	Placing Objects 3 and 4	Picking Objects 5 and 6	Placing Objects 5 and 6
RRTConnect	4,776	15,362	1,075	9,334	257,082	26,794
RRT	218,885	-	-	-	-	-
KPIECE	188,251	74,171	6,108	73,010	44,371	146,258
PRM	600,076	601,639	600,062	602,464	600,107	602,195
PRMStar	600,145	602,927	600,036	604,496	600,098	602,429
EST	170,272	84,707	3,469	21,265	101,566	69,806

Table 6.6: Scenario A - Simultaneous arm movement average planning time (seconds) to compute a path for picking and placing of each pair of objects, with adapted parameters

Planner	Picking Object 1 and 2		Placing Object 1 and 2		Picking Object 3 and 4		Placing Object 3 and 4		Picking Object 5 and 6		Placing Object 5 and 6		Overall
	SR (%)	NoT	SR (%)	NoT	SR (%)	NoT	SR (%)	NoT	SR (%)	NoT	SR (%)	NoT	SR (%)
RRTConnect	100,00%	1,00	100,00%	1,00	100,00%	1,00	100,00%	1,00	100,00%	1,00	100,00%	1,00	100,00%
RRT	90,00%	4,33	100,00%	1,56	0,00%	-	-	-	-	-	-	-	0,00%
KPIECE	100,00%	1,00	100,00%	1,00	100,00%	1,00	100,00%	1,00	100,00%	2,00	70,00%	2,43	70,00%
PRM	100,00%	1,70	100,00%	1,00	100,00%	1,00	100,00%	1,00	100,00%	2,10	40,00%	3,00	40,00%
PRMStar	100,00%	1,00	100,00%	1,00	100,00%	1,00	100,00%	1,00	100,00%	1,90	90,00%	2,44	90,00%
EST	100,00%	1,00	100,00%	1,00	100,00%	1,00	100,00%	1,00	100,00%	2,00	100,00%	1,90	100,00%

Table 6.7: Scenario A - Simultaneous arm movement Success Rate (SR) and average Number of Tries (NoT) it took to compute a path for picking and placing of each pair of objects, with adapted parameters and joint space goals

In these, the previous set up was maintained, i.e., the longest traveled path without performing collision check is still 2 cm and the maximum planning time is 10 minute. The only difference introduced was that, instead of feeding each planner the Cartesian picking or placing position the value of each joint was passed. The transformation from Cartesian goals to joint ones can be performed through the equations exposed in Chapter 2. The resulting success rates and average number of tries can be consulted in Table 6.7.

The new obtained results reveal, without a doubt, a far more reliable behaviour. RRT-Connect was able to perform every path and on the first try, making it the most effective and trustworthy planner. RRT, on the other hand, still displayed a faulty result, albeit was able to place objects 1 and 2 in all cases - which is a further step than on the previous tests. It is also at the utmost importance to underline the efficiency of EST and PRM*, being that the former had a perfect success rate and the latter an almost perfect one.

As it has been the method before we will now look into the average planning time it took each planner to compute a path, this values are replicated in Table 6.8. In these tests some surprising results can be observed, for instance, for RRT-Connect there is a tremendous decreased in the average planning time of almost every path. Another noticeable aspect is the times registered for PRM*, for the first time in the all simulation it did not took the maximum given planning time, since it was able to find the path before that. For both KPIECE and EST there is an overall decline in the planning times, however they do not achieve valuables as low as RRT-Connect. Finally there is still another occurrence that should be pointed out, the unanimous increase of the planning time for the placing of object 5 and 6, this might be due to a poor choosing of the joint values fed to the planners.

Reaching the end of this scenarios it is crucial to rewind some of the steps taken, there were 3 different test in total, the first one had the same conditions as in the experiments - maximum planning time of 5 seconds, a discretization for collision checking of 5 cm and a Cartesian position as goal -, the second the time was increased to 10 minutes and the discretization lowered to 2 cm and, on the last one, besides that the goal position was in the joint space. For each tests the results attained were gradually better, being that in the end more than one planner was able to achieve an 100% success rate. We will now analyze the behaviour of the planners for another scenario, being that the first results presented are with the same parameters as in this scenario's first test,

Planner	Picking Objects 1 and 2	Placing Objects 1 and 2	Picking Objects 3 and 4	Placing Objects 3 and 4	Picking Objects 5 and 6	Placing Objects 5 and 6
RRTConnect	0,446	0,430	0,549	5,945	123,571	42,676
RRT	246,409	91,552	-	-	-	-
KPIECE	16,532	3,242	6,319	35,293	31,478	248,685
PRM	600,098	600,108	600,089	602,354	600,089	601,724
PRMStar	421,320	421,796	422,083	442,131	442,279	440,092
EST	4,604	8,298	5,839	59,137	45,575	306,496

Table 6.8: Scenario A - Simultaneous arm movement average planning time (seconds) to compute a path for picking and placing of each pair of objects, with adapted parameters and joint space goals

i.e., maximum planning time of 5 seconds, a discretization for collision checking of 5 cm and a Cartesian position as goal.

6.2.2 Scenario B

So far we have concluded that all planners can handle manipulators with high DoF, however when obstacles are present and need to be manipulated the outcomes are not as satisfactory, specially when one requests that the arms move simultaneously. Scenario B aims to subject the planners to more complex and various objects, requiring different grasping and placing poses.

Table 6.9 allows to inspect each planners efficiency for this scenario when each arm moves at a time. Surprisingly all planners present positive results, being that this time RRT succeeded in 9 out of the 10 simulations, this may be the consequence of two factors: the existence of fewer objects lead to 4 fewer paths that could have decreased the overall success of the simulation and a better sampling might have occurred, it is not possible to pin point exactly what is the cause to this event. There can be noticed a slightly better performance for all planners, however, considering both the success rate and the average number of tries RRTConnect proves, once again, to be the most efficient. Another aspect that is highlighted when comparing the number of tries in this scenario to Scenario A - Table 6.1 - is that in this case fewer attempts were made, which indicates that the paths were easier to compute, possibly due to the decline in the number of obstacles present in the surroundings.

Simpler paths usually mean shorter planning times, looking at Table 6.10 that lists the planning times for each path and planner it can be seen that the overall planning time for each planner decreased, although some fluctuation are noted compared to Scenario A. There are three consistent outcomes when it comes to planning times, the first one is that, in both the experiments and real life scenarios simulation both PRM and PRM* yield similar times, always around 5 seconds; second EST - that uses the roadmap concept though sampled towards the areas of interest of the problem to solve. - presents faster results than PRM or PRM*; third RRTConnect, as it is expected, confirms to need a shorter amount of time to compute a path.

When both arms move at the same time it is already known that the problem's level of complexity is elevated, therefor it is expected that the planners will not output such good results. This is corroborated by Table 6.11, that displays this scenario's success rates and number of tries for

Planner	Picking Object 1		Picking Object 2		Placing Object 1		Placing Object 2	
	SR (%)	NoT	SR (%)	NoT	SR (%)	NoT	SR (%)	NoT
RRTConnect	100,00%	1,00	100,00%	1,00	100,00%	1,20	100,00%	1,20
RRT	100,00%	1,90	100,00%	2,00	100,00%	2,10	90,00%	2,22
KPIECE	100,00%	1,50	100,00%	1,60	100,00%	1,60	100,00%	1,30
PRM	100,00%	1,00	100,00%	1,10	100,00%	1,20	100,00%	1,40
PRMStar	100,00%	1,00	100,00%	1,10	100,00%	1,10	100,00%	1,20
EST	100,00%	2,10	100,00%	2,20	100,00%	1,40	90,00%	1,00

Planner	Picking Object 3		Picking Object 4		Placing Object 3		Placing Object 4		Overall
	SR (%)	NoT	SR (%)	NoT	SR (%)	NoT	SR (%)	NoT	SR (%)
RRTConnect	100,00%	1,00	100,00%	1,00	100,00%	1,00	100,00%	1,00	100,00%
RRT	100,00%	1,22	100,00%	1,56	100,00%	1,11	100,00%	1,56	90,00%
KPIECE	100,00%	1,10	110,00%	1,20	100,00%	1,00	100,00%	1,00	100,00%
PRM	90,00%	2,56	100,00%	2,56	100,00%	1,00	100,00%	1,00	90,00%
PRMStar	90,00%	3,33	100,00%	2,89	100,00%	1,00	100,00%	1,00	90,00%
EST	100,00%	1,00	100,00%	1,00	100,00%	1,00	100,00%	1,00	90,00%

Table 6.9: Scenario B - Single arm movement Success Rate (SR) and average Number of Tries (NoT) it took to compute a path for picking and placing of each object

Planner	Picking Object 1	Picking Object 2	Placing Object 1	Placing Object 2	Picking Object 3	Picking Object 4	Placing Object 3	Placing Object 4
RRTConnect	0,262	0,352	0,301	0,275	1,933	0,920	0,340	0,243
RRT	1,686	1,732	1,625	1,189	2,091	2,455	1,145	1,816
KPIECE	2,999	2,120	1,393	1,800	2,141	1,740	1,208	1,139
PRM	5,033	5,022	5,047	5,027	5,030	5,025	5,030	5,024
PRMStar	5,043	5,047	5,041	5,046	5,039	5,054	5,025	5,051
EST	2,830	2,323	1,718	0,725	2,498	3,031	1,157	1,251

Table 6.10: Scenario B - Single arm movement average planning time (seconds) to compute a path for picking and placing of each object

Planner	Picking Object 1 and 2		Placing Object 1 and 2		Picking Object 3 and 4		Placing Object 3 and 4		Overall
	SR (%)	NoT	SR (%)	NoT	SR (%)	NoT	SR (%)	NoT	SR (%)
RRTConnect	100,00%	1	100,00%	1,8	50,00%	5,2	100,00%	1,2	50,00%
RRT	40,00%	5,75	50,00%	5,5	50,00%	3	100,00%	7	10,00%
KPIECE	30,00%	5	100,00%	4	66,67%	1	100,00%	1,5	20,00%
PRM	60,00%	5,167	83,33%	3,6	0,00%	-	-	-	0,00%
PRMStar	100,00%	2,7	90,00%	4	0,00%	-	-	-	0,00%
EST	0,00%	-	-	-	-	-	-	-	0,00%

Table 6.11: Scenario B - Simultaneous arm movement Success Rate (SR) and average Number of Tries (NoT) it took to compute a path for picking and placing of each pair of objects

each path and planner. It is clear that neither PRM, PRM* or EST are able to succeed in any of the simulations. Compared to the last scenario - Table 6.3 - PRM and EST have a decline of 20% and 10% respectively. So far PRM* has not been able to perform in any of the real life scenarios tested. On the other hand, this time RRT was able to finish one of the simulations and KPIECE also had an increase on the number of succeed test, from 0 to 2. It is of the utmost importance to state that RRTConnect has been repeatedly been proven to be the most efficient of all planners chosen

Since PRM, PRM* and EST could only plan 1 or 2 of the wanted paths there is not much information regarding the planning time, nevertheless in Table 6.12 it can be verified that both PRM and PRM* take the usual 5 seconds to compute a path. As for the rest of the planners - RRTConnect, RRT and KPIECE - like it happened for the single arm movement case, the planning time declined compared to scenario A and like it happened in Scenario A the planning time increased when compared to the single arm movement.

As it occurred in scenario A, for the simultaneous arm movement the results are not at all satisfactory, thus the same parameters were changed. Table 6.13 shows the success rate and average number of tries for this scenario when the longest traveled distance without collision checking is decreased to 2 cm, the planning time is incremented to 10 minutes and the final position is set in the joint space.

As it is expected the newly obtained results are far better than the previous ones. RRT-Connect,

Planner	Picking Object 1 and 2	Placing Object 1 and 2	Picking Object 3 and 4	Placing Object 3 and 4
RRTConnect	1,263	0,968	3,512	0,676
RRT	3,143	1,558	4,156	1,675
KPIECE	4,471	1,879	2,546	0,528
PRM	5,047	5,061	-	-
PRMStar	5,052	5,100	-	-
EST	-	-	-	-

Table 6.12: Scenario B - Simultaneous arm movement average planning time (seconds) to compute a path for picking and placing of each pair of objects

Planner	Picking Object 1 and 2		Placing Object 1 and 2		Picking Object 3 and 4		Placing Object 3 and 4		Overall
	SR (%)	NoT	SR (%)	NoT	SR (%)	NoT	SR (%)	NoT	SR (%)
RRTConnect	100,00%	1,00	100,00%	1,00	100,00%	1,20	100,00%	1,00	100,00%
RRT	60,00%	4,50	100,00%	1,67	100,00%	3,17	100,00%	1,50	60,00%
KPIECE	100,00%	1,90	100,00%	1,00	100,00%	1,00	100,00%	1,00	100,00%
PRM	70,00%	2,43	100,00%	1,14	28,57%	5,50	100,00%	1,00	20,00%
PRMStar	100,00%	1,30	100,00%	1,00	40,00%	6,25	100,00%	1,00	40,00%
EST	100,00%	1,80	100,00%	1,00	100,00%	1,00	100,00%	1,00	100,00%

Table 6.13: Scenario B - Simultaneous arm movement Success Rate (SR) and average Number of Tries (NoT) it took to compute a path for picking and placing of each pair of objects, with adapted parameters and joint space goals

as it happened in scenario A, presents a perfect performance, being able to complete every simulation almost always on the first try. EST also got some impressive results, this indicates that the expansion of two trees - one starting at the initial and other at the final configuration - yields effective results. In this case KPIECE was able to plan for all paths in all simulations and the average number of tries is close to the ones necessary for RRT-Connect. Although PRM and PRM* performed poorly they still behaved better than on the previous tests - Table 6.11.

When it comes to the average planning time of the new set of tests - Table 6.14 - we can see that there is a uniformity with the previous ones, the computational time for the picking of objects 3 and 4 is far greater than the others, indicating that it is a difficult path to plan. As it was stated before both PRM and PRM* - planners based solely on roadmaps - take the full 10 minutes to compute a path. Also RRT-Connect reveals to be the fastest one in yielding a response.

6.3 Summary

Having reached the end of this chapter and after completing the simulation of two pick and place operations we can safely say that the planning for the cases where each arm moves separately the success rates are satisfactory, and in both scenarios RRT-Connect could easily plan all wanted

Planner	Picking Object 1 and 2	Placing Object 1 and 2	Picking Object 3 and 4	Placing Object 3 and 4
RRTConnect	8,576	1,899	100,564	1,493
RRT	58,658	121,888	197,636	24,085
KPIECE	178,305	9,321	58,828	5,574
PRM	600,099	600,084	600,095	600,140
PRMStar	600,185	600,261	600,166	600,190
EST	253,662	13,284	97,749	6,372

Table 6.14: Scenario B - Simultaneous arm movement average planning time (seconds) to compute a path for picking and placing of each pair of objects, with adapted parameters and joint space goals

paths. However the main focus is on the simultaneous arm movement, in these cases the results gotten with the same set up as for the single arm movement are not acceptable. Which leads to the need to tune some parameters, such as, decrease the maximum traveling distance without performing collision checking, increasing the planning time and setting the final goal on the joint space. These are the most important conclusions withdrawn from these tests. After executing these changes the results yielded improved significantly, being that RRT-Connect and EST presented, for both scenarios, an 100% success rate. As for RRT it was observed that its performance is usually poor, this is due to the expansion of a single tree starting at the initial configuration, thus the computed path after the maximum time is usually an approximated solution since it could not reach the goal, it only got close to it. It was also clear that the planner based solely on roadmaps - PRM and PRM* -, although can sometimes perform, are not reliable.

As for the average planning time it was important to assess that PRM and PRM* usually take the maximum possible planning time to output a result. KPIECE also revealed itself to compute a faster outcome on path that took usually faster planners more time to compute - i.e. more difficult paths. RRT-Connect and EST though presenting similar success rates do it in different times, being that RRT-Connect is usually faster.

Chapter 7

Conclusions

Throughout this thesis we have reviewed several motion planning algorithms, we have seen some work that has already been done in the field of motion planning of dual arm manipulators, we have chosen six different planner to analyzed and tested them in a two arm manipulator in both collision free and non free environments. In the former we concluded that all planners yielded similar results concerning the success rates, however the planners based solely in roadmaps - PRM and PRM* - needed more time to compute a plan. It was also clear - and expected - that planning and moving each arm separetly results in a better performance, due to the simplicity - when compared to the simultaneous movement - of the problem. In the end of all the experiments we can safely affirm that all planners can handle dual arm manipulators.

Following this conclusion simple pick and place operations were simulated in order to mimic real life situations, these were simulated with several objects with different forms that required various picking and placing poses. Here the increased complexity of the simultaneous arm planning was corroborated by the decline in the planners success rate, being that for the separate movement the outcomes were overall satisfactory. However it is important to keep in mind that the main goal is to find a system capable of moving both arms, simultaneously, to the goal positions without colliding with anything else. This was achieved by reducing the maximum distance each arm was allowed to move without the collision checker was called, by increasing the planning time and by setting the goal positions in the joint space instead in the Cartesian one. In these tests the success rate for the simultaneous movement improved drastically. However, if for the object free environments the planners performance is quite similar, in a non obstacle free environment it is not. In the real life scenarios simulations it is clear that the planners solely based on roadmaps - PRM and PRM* - do not yield the wanted results and take a great deal of time to compute a successful path, which makes them unreliable planners for dual arm manipulators. RRT although it has been designed to handle nonholonomic constraints and high degrees of freedom falls short when planning for two arms, this is due to the expansion of only one tree, being that in most cases the solution found was approximated, i.e. it was close to the goal but did not reach it, leading to a failed outcome. Possibly a longer planning time will allow it to reach the wanted goal, however there are faster and more effective planners that can perform the wanted task, thus excluding RRT

as the preferential choice. KPIECE presents a different approach and it is not, to our knowledge, an extensively studied or used planner, nevertheless yielded interesting results by being able to complete most of the simulations and by being faster in computing paths that took usually faster planners to compute. Despite all that, it still failed in some of the tasks given making it less trustworthy. Finally there are two planners left to discuss: RRT-Connect and EST. These two planners revealed similar and perfect success rates in the cases tested and there parity in creating two motion trees leads us to conclude that this is an advantageous feature in dealing with a dual arm manipulator. However, the average planning time cannot be overlooked and in this parameter RRT-Connect displayed overall much faster times, making it the ideal planner to solve our problem. The final goal has been reached, a smart collision avoidance system for a dual arm manipulator has been found.

To sum up, we conclude that the simultaneous collision free arm movement, though more complex, is possible to achieve using RRT-Connect. However it is important to underline the need to set the final pose in the joint space, otherwise the performance of the planner can drop drastically.

7.1 Future Work

After the end of this work there is still some further studies that can be carried out. The first of them is the execution of tests on a real dual arm manipulator to evaluate if the planner can actually perform with accuracy when having to deal with the inherent situations of real world testing. Secondly it is important to analyze the planners behaviour when facing more difficult scenarios, with more objects and with more complex forms. Adjacent to this it would also be interesting to assess if the planner can handle path constraints, for example both arms having to pick up a tray and carry it without turning it. Finally it would be fruitful to redo this work but taking in consideration not only the success rate and the average planning time but also other parameters, for example the length of the traveled path.

References

[1] Seth Hutchinson Mark W. Spong and M. Vidyasagar. *Robot Modeling and Control*. John Wiley & Sons, First edition, 2005.

[2] R. S.Hartenberg J. Denavit. A kinematic notation for lower-pair mechanisms based on matrices. *Trans. of the ASME. Journal of Applied Mechanics*, 22:215–221, 1955.

[3] Jr James J. Kuffner. *Autonomous agents for real-time animation*. Ph.d. dissertation, Department of Computer Science, Stanford University, 1999.

[4] Jean-Claude Latombe. Motion planning: A journey of robots, molecules, digital actors, and other artifacts. *International Journal of Robotics Research*, 18:1119–1128, 1999.

[5] Yoshiaki Kuwata, Sertac Karaman, Justin Teo, Emilio Frazzoli, Jonathan P. How, and Gaston A. Fiore. Real-time motion planning with applications to autonomous urban driving. *IEEE Trans. Contr. Sys. Techn.*, 17(5):1105–1118, 2009.

[6] Goldberg K Alterovitz R, Branicky M. Motion planning under uncertainty for image-guided medical needle steering. *The International journal of robotics research*, 27(11-12):1361–1374, 2008.

[7] Siddhartha S. Srinivasa, Dave Ferguson, Casey J. Helfrich, Dmitry Berenson, Alvaro Collet, Rosen Diankov, Garratt Gallagher, Geoffrey Hollinger, James Kuffner, and Michael Vande Weghe. Herb: a home exploring robotic butler. *Autonomous Robots*, 28(1):5, Nov 2009.

[8] Lars-Peter Ellekilde and Henrik Gordon Petersen. Motion planning efficient trajectories for industrial bin-picking. *The International Journal of Robotics Research*, 32(9-10):991–1004, 2013.

[9] Howie Choset, Kevin M. Lynch, Seth Hutchinson, George A. Kantor, Wolfram Burgard, Lydia E. Kavraki, and Sebastian Thrun. *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, Cambridge, MA, June 2005.

[10] Tomas Lozano-Perez. Spatial planning: A configuration space approach. *IEEE Transactions on Computers*, C-32:108–120, 1983.

[11] Wei Chun Tsai Ting-Hsiang Lin Gene Eu Jan, Chi-Chia Sun. An o(nlogn) shortest path algorithm based on delaunay triangulation. *IEEE/ASME Transactions on Mechatronics*, 19(2):660 – 666, April 2014.

[12] Mohamed Elbanhawi, Milan Simic, and Reza Jazar. Autonomous robot path planning: An adaptive roadmap approach. 373-375:9, 08 2013.

[13] Hirohisa Hirukawa and Yves Papegay. Motion planning of objects in contact by the silhouette algorithm. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 1, pages 722–729. IEEE, 2000.

[14] Roland Siegwart and Illah R. Nourbakhsh. *Introduction to Autonomous Mobile Robots*. Bradford Company, Scituate, MA, USA, 2004.

[15] Franz Aurenhammer. Voronoi diagrams - a survey of a fundamental geometric data structure. *ACM Comput. Surv.*, 23(3):345–405, September 1991.

[16] Jean-Claude Latombe. *Approximate Cell Decomposition*, pages 248–294. Springer US, Boston, MA, 1991.

[17] Bruce R. Donald. A search algorithm for motion planning with six degrees of freedom. *Artificial Intelligence*, 31(3):295 – 353, 1987.

[18] Lydia Kavraki, Petr Svestka, Jean-Claude Latombe, and Mark Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

[19] Mohamed Elbanhawi and Milan Simic. Sampling-based robot motion planning: A review. *Ieee access*, 2:56–77, 2014.

[20] Steven M. Lavalle. Rapidly-exploring random trees: A new tool for path planning. Technical report, 1998.

[21] J. J. Kuffner and S. M. LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, volume 2, pages 995–1001 vol.2, 2000. doi:10.1109/ROBOT.2000.844730.

[22] Ioan Alexandru Sucan and Lydia E. Kavraki. Kinodynamic motion planning by interior-exterior cell exploration. In *WAFR*, 2008.

[23] D. Hsu, J. C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. In *Proceedings of International Conference on Robotics and Automation*, volume 3, pages 2719–2726 vol.3, Apr 1997. doi:10.1109/ROBOT.1997.619371.

[24] Y. Koga and J. C. Latombe. Experiments in dual-arm manipulation planning. In *Proceedings 1992 IEEE International Conference on Robotics and Automation*, May 1992.

[25] Benjamin Cohen, Sachin Chitta, and Maxim Likhachev. Single-and dual-arm motion planning with heuristic search. *The International Journal of Robotics Research*, 33(2):305–320, 2014.

[26] N. Vahrenkamp, D. Berenson, T. Asfour, J. Kuffner, and R. Dillmann. Humanoid motion planning for dual-arm manipulation and re-grasping tasks. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2464–2470, Oct 2009.

[27] Y. C. Tsai and H. P. Huang. Motion planning of a dual-arm mobile robot in the configuration-time space. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2458–2463, Oct 2009.

[28] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. Russell, and P. Abbeel. Combined task and motion planning through an extensible planner-independent interface layer. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 639–646, May 2014.

[29] J. P. Saut, M. Gharbi, J. Cortés, D. Sidobre, and T. Siméon. Planning pick-and-place tasks with two-hand regrasping. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4528–4533, Oct 2010.

[30] Jun Kurosu, Ayanori Yorozu, and Masaki Takahashi. Simultaneous dual-arm motion planning for minimizing operation time. *Applied Sciences*, 7(12):1210, 2017.

[31] ROS. Ros introduction. URL: http://wiki.ros.org/ROS/Introduction.

[32] Open Source Robotics Foundation. Gazebo website. URL: http://gazebosim.org/.

[33] Ioan A. Sucan and Sachin Chitta. Moveit! URL: hhttp://moveit.ros.org.

[34] Ioan A. Şucan, Mark Moll, and Lydia E. Kavraki. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, December 2012. http://ompl.kavrakilab.org. doi:10.1109/MRA.2012.2205651.