



**Politecnico  
di Torino**

**POLITECNICO DI TORINO**

MASTER DEGREE IN  
MECHATRONIC ENGINEERING

# Collision-free path planning for industrial robot applications

**Supervisor:**

Prof. Marina INDRI

**Candidate:**

Ivan AIZA

**Supervisors at COMAU:**

Ing. Simone PANICUCCI

Ing. Antonio VENEZIA

December 2023

*A Stefano, mio padre  
per sempre inciso nella mia anima*

---

## Abstract

Nowadays, robotics is growing rapidly thanks to its huge flexibility and potential in many industrial applications. One of the recurring problems is the collision-free path planning that allows the robot to move from an initial pose to a final pose safely, avoiding all objects in the scene.

In COMAU, common planners belonging to the sampling-based category are used for collision-free path planning. To do this, the MoveIt framework is used in which is present the OMPL library that contains many of the sampling-based planners. These planners are fast and effective even in complex environments, however they generate geometric paths that are not optimized and do not consider kinematics constraints. Indeed a second step is needed, performed by the C5G controller on board robot, which also considers kinematics constraints and adds a timing law to the collision-free geometric path in order to obtain the final robot trajectory.

In this thesis work, many types of path planners present in the state of the art are analyzed. Then it is decided to develop a comparison between the performances of OMPL in MoveIt currently used in COMAU and TrajOpt which is an innovative optimized-based planner already present in the Tesseract framework. TrajOpt plans collision-free trajectories with an optimized path. In this way the final paths are much better and smooth, and already respect the kinematics constraints. On the other hand, the higher computational complexity increases the planning times required.

Thanks to this thesis work, it is possible to enhance the comparisons about the performances of TrajOpt and the OMPL planners in typical industrial robotics scenarios.

# Contents

<b>1</b>	<b>Introduction</b>	<b>5</b>
<b>2</b>	<b>Background on industrial robotics</b>	<b>7</b>
2.1	General notions on kinematics . . . . .	9
2.2	Direct kinematics . . . . .	11
2.3	Inverse kinematics . . . . .	12
2.4	Differential kinematics . . . . .	13
<b>3</b>	<b>Motion Planning</b>	<b>15</b>
3.1	Configuration space . . . . .	16
3.2	Collision-free motion planning . . . . .	17
<b>4</b>	<b>Motion planning algorithms</b>	<b>19</b>
4.1	Graph search-based algorithms . . . . .	20
4.2	Sampling-based algorithms . . . . .	22
4.2.1	RRT algorithm . . . . .	23
4.2.2	RRT connect and bi-directional RRT algorithms . . . .	25
4.2.3	PRM algorithm . . . . .	26
4.2.4	RRT* and PRM* algorithms . . . . .	28
4.2.5	SBL algorithm . . . . .	29
4.2.6	Deterministic Chekhov algorithm . . . . .	30
4.3	Optimized-based algorithms . . . . .	31
4.3.1	CHOMP algorithm . . . . .	32
4.3.2	STOMP algorithm . . . . .	33
4.3.3	TrajOpt algorithm . . . . .	34
4.3.4	LCQP algorithm . . . . .	37
4.3.5	B-spline algorithm . . . . .	38
4.4	Post-processing algorithms . . . . .	39
<b>5</b>	<b>Simulation framework</b>	<b>40</b>
5.1	ROS . . . . .	40
5.2	ROS nodes and topics . . . . .	41

## CONTENTS

---

5.3	ROS services . . . . .	42
5.4	ROS actions . . . . .	43
5.5	ROS RViz . . . . .	44
5.6	Roboshop . . . . .	45
<b>6</b>	<b>MoveIt</b>	<b>46</b>
6.1	Motion planning . . . . .	48
6.1.1	OMPL . . . . .	48
6.2	Planning scene . . . . .	49
6.2.1	3D perception . . . . .	50
6.3	Collision checking . . . . .	50
<b>7</b>	<b>Tesseract</b>	<b>51</b>
7.1	Motion planning . . . . .	52
7.2	Planning scene . . . . .	52
7.3	Collision checking . . . . .	53
<b>8</b>	<b>Experimental procedure</b>	<b>54</b>
8.1	Racer5-0-80 . . . . .	55
8.2	MoveIt pipeline . . . . .	56
8.3	Tesseract pipeline . . . . .	58
8.3.1	Add_link service . . . . .	59
8.3.2	Tesseract_planning service . . . . .	60
<b>9</b>	<b>Benchmark 1: table avoidance</b>	<b>61</b>
9.1	Results . . . . .	62
<b>10</b>	<b>Benchmark 2: Pick &amp; Place</b>	<b>63</b>
10.1	Results . . . . .	64
<b>11</b>	<b>Benchmark 3: Pick &amp; Place with obstacle avoidance</b>	<b>66</b>
11.1	Results . . . . .	67
<b>12</b>	<b>Benchmark 4: TrajOpt safe distance</b>	<b>68</b>
12.1	Results . . . . .	69
<b>13</b>	<b>Gripper change management</b>	<b>70</b>
13.1	NJ-370-2.7 . . . . .	70
13.2	Gripper change management pipeline . . . . .	71

## CONTENTS

---

<b>14 Conclusions</b>	<b>74</b>
<b>15 Future works</b>	<b>75</b>
<b>16 Acknowledgements</b>	<b>76</b>
<b>17 Acronyms</b>	<b>77</b>
<b>List of Figures</b>	<b>80</b>
<b>List of Tables</b>	<b>81</b>
<b>Bibliography</b>	<b>82</b>

# Chapter 1

## Introduction

This thesis work is developed within the Innovation Hub at COMAU SpA in Turin and it deals with collision-free path planning which is a relevant topic in industrial robotics. Collision-free path generation is a deeply studied and analyzed branch worldwide. This is because robotics is growing in many industrial and non-industrial fields and at present there are still no flawless planners.

To understand better the relevance of robotics, think about how the production lines are organized, where one or more robotic arms are involved in the assembly of products (Figure 1.1), or in automated warehouses where products are sorted by robots. Other industrial applications are robotic welding, quality inspection of production lines up to collaborative robotics where robots and workers share the same workspace. Having said that, the importance of moving the robots safely, accurately and efficiently, thus avoiding any kind of collision with objects in the environment (and, in the case of collaborative robots, workers) becomes intuitive.

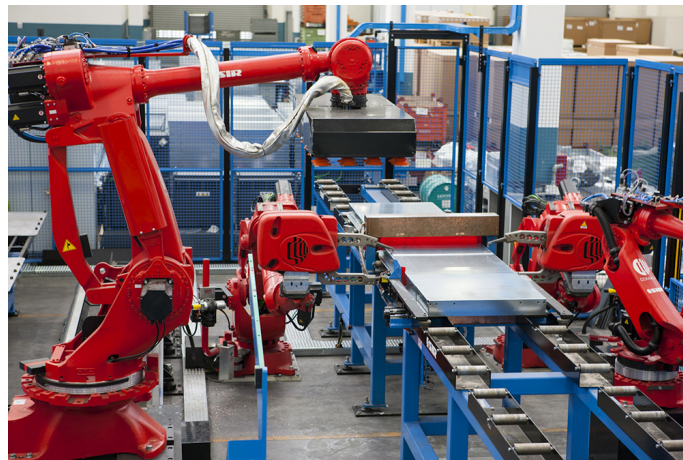


Figure 1.1: COMAU industrial robots working in a production line [35].

As it will be discussed in Chapter 4 about the state of the art, there are many types of motion planners. The most widespread are the sampling-based planners that are fast and quite effective, but they generate only geometric and not optimal paths. On the other hand, there are the optimized-based planners that optimize the geometric path and are able to generate a trajectory, also considering timing law and kinematics constraints, but with relatively long planning times. Finally, there are the innovative trajectory post-processing planners, which are a hybrid way of the two types explained above. The choice of the appropriate planner depends on multiple factors. First selections can be made based on the environment, which can be structured or unstructured, static or quasi-static or dynamic. In addition, the planner can be used online or offline.

The planners used in COMAU are sampling-based because of their short planning time and high success rate in typical industrial robotics problems. However, these planners do not provide optimized paths, thus resulting in robot movements that are not always necessary. In particular, the MoveIt environment is used to generate collision-free paths, thanks to the OMPL library that contains various sampling-based planners.

In this thesis, a comparison is made between the performance of the newer TrajOpt planner and the OMPLs currently used.

TrajOpt turns out to be an optimized-based planner that generates optimized paths with high success rates, but at the expense of planning time, which can be longer. For this reason, it is interesting to compare it with current planners in typical industrial robotics environments in order to evaluate its real performance.

Going into the details about the structure of this thesis, there is a first part where an introduction to industrial robotics (Chapter 2) and motion planning problems (Chapter 3) is provided. Chapter 4 reports an analysis of the current state of the art regarding motion planning. In Chapters 5-7 there is an in-depth look at the simulation environment used, particularly ROS, MoveIt, and Tesseract. After such more theoretical part, there are the experimental chapters, which illustrate the experimental procedure (Chapter 8) and also the benchmarks and tests performed (Chapters 9-13). Finally, in Chapter 14 some conclusions are drawn from the data obtained in the experimental part and then, in Chapter 15, possible future developments are proposed to further investigate the comparison between OMPL and TrajOpt.



# Chapter 2

## Background on industrial robotics

Industrial robotics is an important branch of robotics that deals with the design, construction, and operation of robotic arms (called manipulators) in the industrial setting. Industrial robots are usually designed to perform the most repetitive and dangerous tasks in different fields such as manufacturing, automotive, electronic manufacturing, and so on. The main goal of industrial robotics is to increase the efficiency, precision, and safety of industrial processes.

In detail, an industrial robot generally consists of the following parts:

- manipulator: it is the robotic arm that moves according to the instructions it receives.
- end-effector: it is the end of the manipulator or the gripper or the tool that is mounted on it. In the case of motion planning, it is the point that is considered for planning.
- sensors: these can be visual sensors, torque sensors, position sensors, etc., that allow the robot to sense the environment and adapt its movements accordingly.
- control unit: this is the controller that processes instructions and controls the movements of the robot. In the COMAU case, this is the C5G on board robot controller.
- programming environment: industrial robots are programmed through specific languages or through direct interfaces such as teach pendants. In COMAU case, this is the TP5.

Focusing on the industrial manipulators, they are composed of a chain of rigid bodies called links connected together by actuators called joints [25]. The robots are moved by mechanisms such as pneumatic systems or electric motors placed on the joints. These joints can be of two types: prismatic joints and revolute joints. The former allow only translation of the two connected links, while the latter allow only rotation of the connected links. Each joint corresponds to a degree of freedom (DOF) of the manipulator and this determines the robot's dexterity.

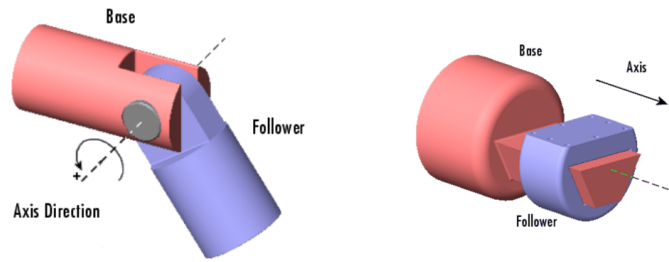


Figure 2.1: Revolute joint (left) and prismatic joint (right) [25].

Industrial robots consist of an arm and a wrist: the arm is composed of the first three links that are responsible for positioning the end-effector in the workspace, while the wrist consists of the other links that are responsible for the end-effector orientation. In this thesis, a 6 DOF robot is used: the arm is composed of three revolute joints, while the wrist is of spherical type, i.e., it is formed by three revolute joints in which the axes intersect at one point.

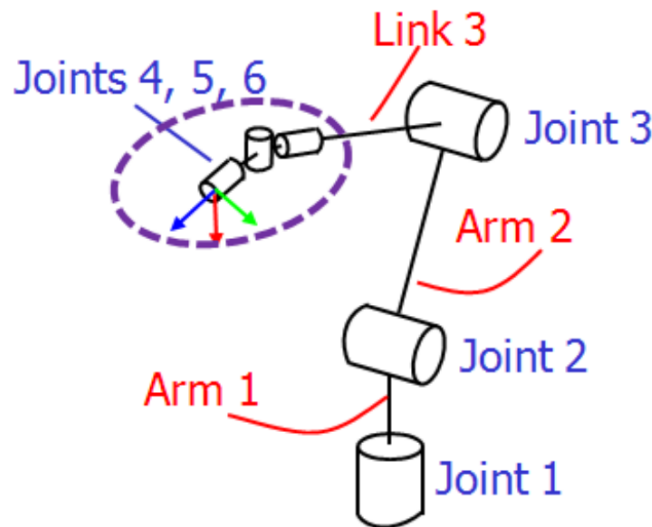


Figure 2.2: Sketch of an industrial robot with 6DOF and a spherical wrist [25].

## 2.1 General notions on kinematics

A manipulator can be mechanically represented as a kinematic chain of links connected by prismatic or revolute joints. This implies that the motion of the end-effector can be obtained as a composition of elementary motions of each link with respect to the previous one.

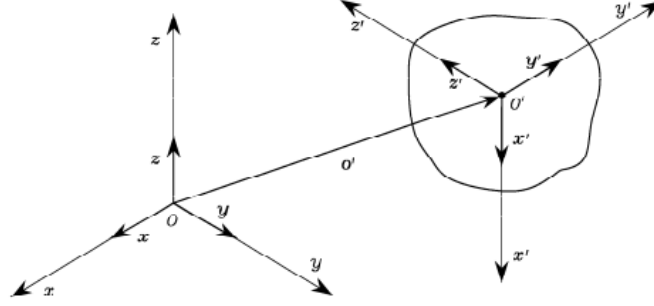


Figure 2.3: Position and orientation of a rigid body [33].

To fully describe the pose of a rigid body, it is necessary to know its position and orientation with respect to a reference frame [33]. Figure 2.3 represents O-xyz as orthonormal reference frame with  $\mathbf{x}$ ,  $\mathbf{y}$  and  $\mathbf{z}$  as reference frame axes, while O'-x'y'z' is the orthonormal body frame with  $\mathbf{x}'$ ,  $\mathbf{y}'$  and  $\mathbf{z}'$  as body frame axes. The position of the point O' on the rigid body with respect to the reference frame O-xyz is expressed by:

$$\mathbf{o}' = o'_x \mathbf{x} + o'_y \mathbf{y} + o'_z \mathbf{z} \quad (2.1)$$

In this equation  $o'_x$ ,  $o'_y$ ,  $o'_z$  are the components of the vector  $\mathbf{o}'$  along the reference frame. It can be written compactly:

$$\mathbf{o}' = \begin{bmatrix} o'_x \\ o'_y \\ o'_z \end{bmatrix}$$

To describe the orientation of the rigid body, the O'-x'y'z' body frame is used with origin in O' and  $\mathbf{x}'$ ,  $\mathbf{y}'$  and  $\mathbf{z}'$  are unit vectors of the body frame. These vectors expressed in the reference frame become:

$$\mathbf{x}' = x'_x \mathbf{x} + x'_y \mathbf{y} + x'_z \mathbf{z} \quad (2.2)$$

$$\mathbf{y}' = y'_x \mathbf{x} + y'_y \mathbf{y} + y'_z \mathbf{z} \quad (2.3)$$

$$\mathbf{z}' = z'_x \mathbf{x} + z'_y \mathbf{y} + z'_z \mathbf{z} \quad (2.4)$$

From here it is possible to obtain the rotation matrix of the body frame with respect to the reference frame:

$$\mathbf{R} = [\mathbf{x}' \quad \mathbf{y}' \quad \mathbf{z}'] = \begin{bmatrix} x'_x & y'_x & z'_x \\ x'_y & y'_y & z'_y \\ x'_z & y'_z & z'_z \end{bmatrix}$$

Rotation matrix  $\mathbf{R}$  is essential for the kinematics of the robot as it represents the orientation of the body frame with respect to the reference frame. The columns of  $\mathbf{R}$  are mutually orthogonal since they represent the unit vectors of the orthonormal frame. It follows that its transpose matrix is equivalent to the inverse matrix and thus the product between the transpose and rotation matrix is equivalent to the identity one. The determinant of the rotation matrix is 1 if the frame is right-handed, and -1 if it is left-handed.

$$\mathbf{R}^T \mathbf{R} = \mathbf{I}_3 \quad (2.5)$$

$$\mathbf{R}^T = \mathbf{R}^{-1} \quad (2.6)$$

It is possible to represent the rigid object pose by means of the single homogeneous transformation matrix  $\mathbf{T}$ .

$$\mathbf{T}_1^0 = \begin{bmatrix} \mathbf{R}_1^0 & \mathbf{o}_1^0 \\ \mathbf{0}^T & 1 \end{bmatrix}$$

$\mathbf{T}_1^0$  is the homogeneous matrix that completely describes the pose of rigid body 1 with respect to reference frame 0.  $\mathbf{R}_1^0$  is the rotation matrix of frame 1 with respect to frame 0,  $\mathbf{o}_1^0$  is the position of the origin of frame 1 with respect to frame 0 and finally the last row is formed by a vector of three 0's below the rotation matrix and a 1 below the position vector.

## 2.2 Direct kinematics

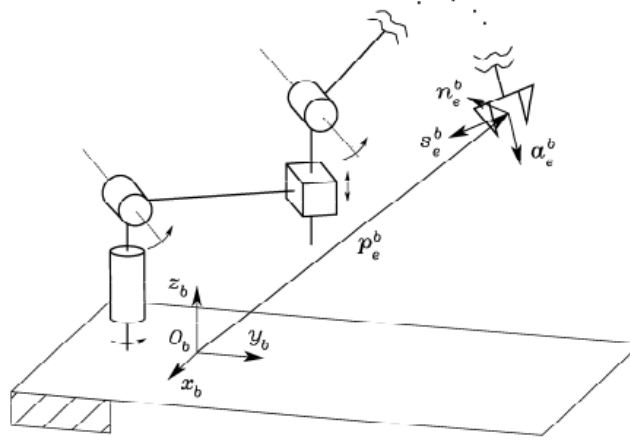


Figure 2.4: Position and orientation of the end-effector frame with respect to the base frame [33].

A DOF is associated to each joint of the robot corresponding to a joint variable. The goal of direct kinematics is to compute the pose of the robot's end-effector as a function of the joint variables [33]. Then by exploiting the concept of homogeneous transformation matrix, introduced in the previous section, a reference frame is fixed at the base of the robot, as in Figure 2.4, and the pose of end-effector can be obtained as:

$$\mathbf{T}_e^b(\mathbf{q}) = \begin{bmatrix} \mathbf{n}_e^b(\mathbf{q}) & \mathbf{s}_e^b(\mathbf{q}) & \mathbf{a}_e^b(\mathbf{q}) & \mathbf{p}_e^b(\mathbf{q}) \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

In this matrix,  $\mathbf{q}$  is the vector of joint variables ( $n \times 1$ ),  $\mathbf{n}_e$ ,  $\mathbf{s}_e$ ,  $\mathbf{a}_e$  are the unit vectors of the end-effector frame, and  $\mathbf{p}_e$  is the position of the end-effector with respect to the base frame.

To simplify the direct kinematics problem is used the Denavit-Hartenberg convention. Following this convention, homogeneous transformation matrices between one link and the previous one are computed, and at the end through the product of the  $n$  homogeneous matrices, the matrix describing the pose of the end-effector can be obtained with respect to the base frame. For example in a robot with 6 DOFs as in the case of this thesis work, the homogeneous transformation matrix between the end-effector and the base frame is:

$$\mathbf{T}_6^0(\mathbf{q}) = \mathbf{T}_1^0 \mathbf{T}_2^1 \mathbf{T}_3^2 \mathbf{T}_4^3 \mathbf{T}_5^4 \mathbf{T}_6^5 \quad (2.7)$$

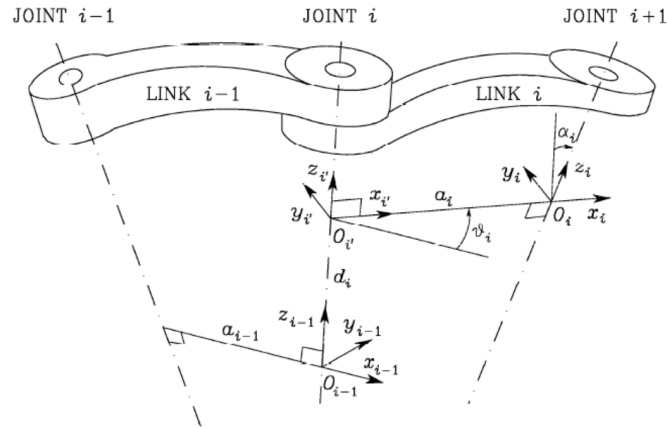


Figure 2.5: Denavit-Hartenberg convention [33].

## 2.3 Inverse kinematics

The problem of inverse kinematics is to find the values of the joint variables knowing the pose of the end-effector [33]. This problem is crucial, since it allows to transform the motion planning assigned to the pose of the end-effector in the operational space into values for the joint variables in the joint space.

While direct kinematics is a relatively simple problem with a unique solution, this one is much more complex. In fact, inverse kinematics usually requires solving nonlinear equations, so it is not always possible to find closed-form solutions. There could be infinite solutions, multiple admissible solutions or even no solution. The existence of the solution is guaranteed only if the given end-effector pose belongs to the manipulator's dexterous workspace. In the simplest cases the problem can be solved by algebraic or geometric intuitions, in others numerical techniques are used, which can be applied in any kinematic structures but do not compute all possible solutions.

In Figure 2.6 it is possible to see an example in which for the same end-effector pose there are two admissible joint poses.

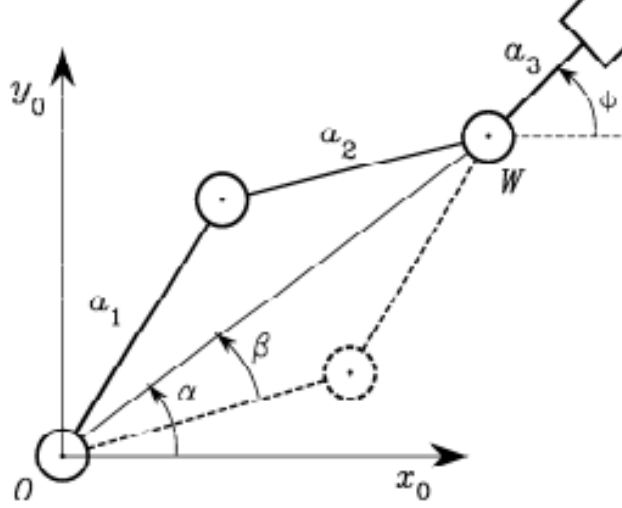


Figure 2.6: Two admissible solutions for the inverse kinematics problem for a two-link planar arm [33].

## 2.4 Differential kinematics

Differential kinematics characterises the mapping between joint velocities and the corresponding end-effector linear and angular velocities [33]. This mapping is described by a matrix called Jacobian. There are two types of Jacobians: if the end-effector pose is expressed by a homogeneous transformation matrix, it is used the so-called geometric Jacobian; while if the end-effector pose is expressed by a minimal representation (by the position-orientation vector), it is used the analytical one.

In the first case, the initial homogeneous transformation matrix is:

$$\mathbf{T}(\mathbf{q}) = \begin{bmatrix} \mathbf{R}(\mathbf{q}) & \mathbf{p}(\mathbf{q}) \\ \mathbf{0}^T & 1 \end{bmatrix}$$

It is possible to compute the end-effector velocities from the joint velocities thanks to the geometric Jacobian of the manipulator:

$$\mathbf{v} = \begin{bmatrix} \dot{\mathbf{p}} \\ \omega \end{bmatrix} = \begin{bmatrix} \mathbf{J}_p(\mathbf{q}) \dot{\mathbf{q}} \\ \mathbf{J}_o(\mathbf{q}) \dot{\mathbf{q}} \end{bmatrix} = \mathbf{J}(\mathbf{q}) \dot{\mathbf{q}}$$

where  $\mathbf{J}_p$  is the  $(3 \times n)$  matrix that refers to how joint velocities affect the linear velocities of the end-effector, while  $\mathbf{J}_o$  is the  $(3 \times n)$  matrix that refers to how joint velocities affect the angular velocities of end-effector.

Instead, in the second case in which is present a minimal representation of the end-effector pose:

$$\mathbf{x} = \begin{bmatrix} \mathbf{p}(\mathbf{q}) \\ \phi(\mathbf{q}) \end{bmatrix}$$

the speed of the end-effector is obtained through the analytical Jacobian matrix:

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{\mathbf{p}} \\ \dot{\phi} \end{bmatrix} = \begin{bmatrix} \frac{\partial \mathbf{p}}{\partial \mathbf{q}} & \dot{\mathbf{q}} \\ \frac{\partial \phi}{\partial \mathbf{q}} & \dot{\mathbf{q}} \end{bmatrix} = \begin{bmatrix} \mathbf{J}_{\mathbf{p}}(\mathbf{q}) & \dot{\mathbf{q}} \\ \mathbf{J}_{\phi}(\mathbf{q}) & \dot{\mathbf{q}} \end{bmatrix} = J_A(q) \dot{\mathbf{q}}$$

In general, the derivative of  $\phi$  with respect to time does not coincide with  $\omega$  defined previously. In any case, it is possible to compute the relationship between the geometric Jacobian and the analytic one that depends on the configuration.



# Chapter 3

## Motion Planning

Motion planning is a branch of robotics that deals with collision-free path planning starting from an initial state and a defined goal state. For this reason, the main goal of the planner is to generate paths that safely avoid the obstacles present. Important parameters in the generated paths are the planning time, the path length and the path smoothness. Usually, very good path lengths and smoothness coincide with long planning times, so an appropriate trade-off is required.

Before going into more detail with current planners, it is important to make a distinction between the concepts of path and trajectory [33]. In fact, the path is the geometric locus of points from the initial state to the final state. Trajectory, on the other hand, is composed of both the geometric path generated and the timing law required to execute it, while respecting all the constraints of velocity, acceleration, jerk, and torque.

In the state of the art, there are many types of planners, each with its advantages and disadvantages, which will be discussed in the subsequent chapters. The main issue in each planner is the trade-off between computational complexity and the quality of the final path. The more precise the paths and with higher quality, the greater their computational complexity and consequently the time required to generate them. An important parameter in this regard is the resolution of the initial scene: the more accurate the scene, the heavier will be the computation of the collision-free path.

Motion planning can be divided into two macro-categories: offline motion planning and online motion planning [26].

In offline motion planning, the planner does not consider dynamic changes in the environment during the execution of the path. As a first step, it detects the scene and obstacles by loading CAD models or by visual sensors that usually generate point clouds or depth images and then they are converted into octomaps. Then the collision-free path is planned considering the scene and

collision objects, and it is executed. This way of proceeding is also called scan-plan-execute, since the executed path considers only the environment scanned in the first step. Possible changes in the scene that occur after the path has been planned, will not be taken into account, causing possible collisions. Intuitively, they are suitable planners for static or quasi-static environments, as the scene is no longer updated after the initial scan.

On the other hand, in online planners, the scene is scanned continuously at every step during the path execution. This is a significant advantage where the scene is dynamic, since the planner can update the path in each scan. However, this procedure is computationally heavy, since at each step the planner has to remake all the computations thus causing significantly longer planning times. To reduce them, it is possible to update the scene at multiple one-step intervals, losing in path update rate but gaining in computation time.

Another common distinction in the motion planning world is between deterministic and nondeterministic planners. The former generate the path uniquely, so given the same initial input conditions (initial state, goal state, and collision objects in the scene) the planned path will always be the same. The latter generate it randomly, so the planned paths may differ even with the same input.

### 3.1 Configuration space

Configuration space is a fundamental concept in the world of motion planning. It represents the set of all possible configurations or poses that the robot can assume in the workspace [26], [33].

The dimension of the configuration space is given by the number of degrees of freedom of the system or the minimum number of parameters required to specify the configuration. It follows that if a robotic arm is moving in a two-dimensional plane then the configuration space dimension is two, if it is in a three-dimensional space then it is three, and so on.

The use of configuration space greatly simplifies the problem of planning a collision-free path; in fact, it reduces the complexity of the problem as only this space is considered during planning instead of the entire geometry of the workspace.

Generalized coordinates are used to identify the robot's pose in the configuration space: six coordinates in which the first three represent the body's

position in space and they are called cartesian coordinates, while the second three are angular coordinates and represent the body's orientation. These six coordinates together completely and uniquely define the robot's pose in space.

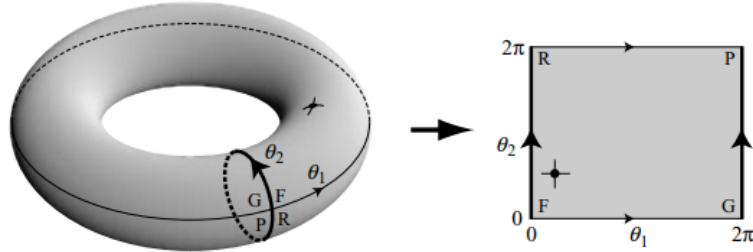


Figure 3.1: Configuration space of a two-joint manipulator: on the left a representation topologically correct as a torus 2D, on the right a representation locally valid as subset  $R^2$  [26].

## 3.2 Collision-free motion planning

The configuration space can be subdivided into two additional complementary spaces essential for motion planning: the free space  $C_{\text{free}}$  and the obstacle space  $C_{\text{obst}}$  [22], [26], [33].

The first,  $C_{\text{free}}$ , is the set of all points in the configuration space that are not in collision with the environment, so it is a subspace of the configuration space in which all points are collision-free. The second, instead, is the set of all points in collision, so it represents the union of obstacles and bodies in the scene. The two sub-spaces are complementary and together form the entire configuration space.

Considering the configuration space of robot  $A$ , where  $A(q)$  corresponds to  $A$  with the configuration of robot  $q$  and  $B$  to the geometric representation of obstacles in the scene:

$$C_{\text{free}} = \{q : A(q) \cap B = \emptyset\} \quad (3.1)$$

$$C_{\text{obst}} = \{q : A(q) \cap B \neq \emptyset\} \quad (3.2)$$

Having defined these sub-spaces in the configuration space, it is possible to define the goal of the motion planner as finding a path that joins the initial state and the goal state in the absence of collisions, that is, finding a path such that all its points belong to the free space  $C_{\text{free}}$ .

As can be seen from Figure 3.2, mapping the obstacles from the workspace to the configuration space is a non-trivial operation, particularly in cases where there are present high-dimensional state space or when the obstacles cannot be reconducted to simple shapes.

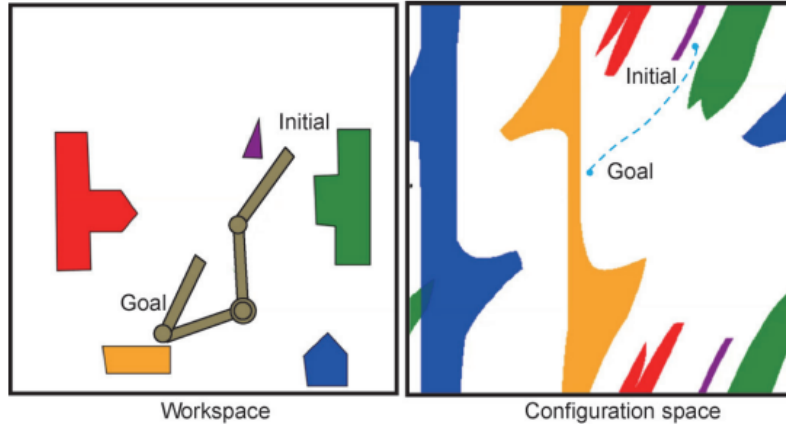


Figure 3.2: The right figure shows the configuration space corresponding to the workspace on the left. A two-link robot is used and each obstacle has a different colour to simplify the mapping [22].

# Chapter 4

## Motion planning algorithms

In the state of the art are present multiple types of motion planning algorithms. Focusing on the logic by which the planner researches for the collision-free path, the main subdivision is among graph search-based, sampling-based and optimized-based planners. The first two planners generate a collision-free path without considering kinematic constraints such as path smoothness or velocity, acceleration and jerk profiles [8]. In other words, they plan a path without a time dependence, so they require a second step in which is defined a timing law and so the final trajectory. In the case of COMAU it is defined by the C5G controller on board of the robot. Although this controller is highly optimized, this procedure could generate kinematic profiles with unnecessary components, so at the end the collision-free paths are still non-optimal. This is due to the fact that the timing law is optimized by the controller, but the initial path generated by the planner is not.

On the other hand, optimized-based algorithms plan also the timing law of the final trajectory, optimizing its kinematic parameters. This causes the trajectory to have the main parameters considered optimal (path length, execution time, smoothness, kinematics profiles, so on) with, however, generally higher computational costs and failure rates [24].

Furthermore, in recent years much research and efforts have focused on post-processing type algorithms: these algorithms provide a hybrid solution between sampling-based and optimized-based algorithms. In this type of planner, there is the generation of an initial collision-free path by a sampling-based planner and then the path is optimized thanks to an optimized-based planner. In this way, optimal collision-free trajectories can be obtained, improving some of the main disadvantages of optimized-based planners such as high failure rate [18].

Another important parameter for a motion planner algorithm is completeness [26]. An algorithm is considered complete if it is guaranteed to find a solution for every instance of the problem (if it exists). This means that if there is at least one path that satisfies the constraints of the problem, the algorithm will find it.

There are cases in which the algorithm is called probabilistically complete: this happens when the probability of finding the solution tends to one if the planning time tends to infinity. This situation is typical of sampling-based algorithms, in which there is random sampling of the configuration space. Probabilistic completeness can be a guarantee of completeness, but of course it depends on the amount of time required: this time depends on both the computational complexity of the algorithm and the complexity of the environment (e.g., number and shape of obstacles, dynamic obstacles, tight spaces, resolution used and so on).

## 4.1 Graph search-based algorithms

Graph search-based planning uses graph search algorithms to compute discrete paths in the robot's state space [26].

This type of planning can be divided into two different main problems: the first is how to transform the initial problem into a graph, the second is how to find the graph corresponding to the best possible solution. In order to represent the space of states with a graph, all possible states are discretized, so that there is a finite number of poses in which it could stay.

Graph search algorithms have been extensively studied and vary from the simplest and most classical ones, to more complex and with better performance. In the latter, a cost is associated to each node and each arc of the search tree, then based on this cost it is decided whether (and how) to expand the tree or whether it is better to stop and then find the optimal path. An example of this is Dijkstra's algorithm, which associates a cost with each node based on its distance from the final or initial node (depending on whether the backward or forward variant of the algorithm is used).

Furthermore, there are two macro-categories of graph search-based planners: depth-first searches and breadth-first searches. The former start at the root, choose a child node and continue to deepen it. If they encounter a leaf, they go back one level and restart with an unvisited child. On the other hand,

the second ones start from the root node as before and next they visit all the children. Once all the children have been visited, the algorithm proceeds by delving into all the grandchildren. The latter is more complex, but it also generates the shortest path between the start node and the goal node.

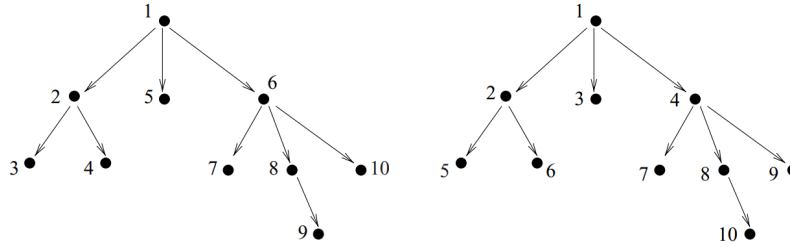


Figure 4.1: Comparison of depth-first (left) and breadth-first (right) planners. The number near the nodes is the order in the expansion of the tree [26].

The main advantage of these planners is completeness. If the graph representing the space of states is constructed correctly, then these planners are able to find a solution, assuming it exists. Moreover, among the various solutions found, the final one is also the optimal one based on the path length.

On the other hand, the discretization of the space limits its final accuracy and thus even the path considered optimal by the planner may be too approximate.

Moreover, having to transform the entire space of states into a graph in order to find the best solution, the algorithm is computationally heavy, particularly in environments that are complex or that require a large number of graph nodes. This results in high and, especially, unnecessary computation time: these planners build the graph over the entire space of states, thus also in areas that are not needed for path planning purposes. This results in a waste of planning time for the purposes of the required problem.

The lost and unnecessary computational cost is the main difference with sampling-based planners. The latter in fact randomly sample the space of states, so they lose in the completeness of the solution (they still guarantee probabilistic completeness), but greatly shorten the planning time by being computationally significantly lighter. For this reason, graph search-based planners are no longer used in industrial robotics in favour of sampling-based planners.

## 4.2 Sampling-based algorithms

Sampling-based planners are the most common planners, because they are very quick and quite effective even in complex environments [3], [8], [26], [33]. In this type of algorithm, the configuration space is randomly sampled (i.e., following a certain probability distribution, such as Gaussian). For each sampling point, it is checked whether the robot's configuration is collision-free; if it is, it adds this configuration to the graph of possible configurations and connects it, if possible, to other configurations already in the graph. Once the graph is constructed, the algorithm determines a collision-free path that connects the initial pose to the final one.

This way of proceeding is probabilistically complete: in fact, if it is assumed that is present a number of configurations in the graph tending to infinity (and thus also the time it takes to find them); if a collision-free solution exists, it surely belongs to the constructed graph.

Sampling-based planners turn out to be fast even in high-dimensional configuration space. In fact, unlike graph search-based, their computational cost does not depend exponentially on the size of the state space. The success rate of these planners depends strongly on the complexity of the scene and the maximum planning time set. Since they are probabilistically complete planners, in some cases they may not have time to find the collision-free path, even if it exists. This speed of computation combined with a good success rate even in complex environments has made them currently the most popular in the industrial world.

An important distinction of this type of random-sampling planners is the one between single-query planners and multiple-query planners.

In single-query planners, the important concept is the speed in computing a single path to be executed. If another path is then to be planned, it restarts from the beginning and it is necessary to re-initialize everything, since nothing is stored of the previously planned paths.

In multiple-query planners, on the other hand, it is assumed in principle that there will be many motion planning problems to be solved in the same environment, so it is essential to save the information of each path in order to speed up subsequent computations, having no need to re-initialize the algorithm.



In the sampling-based planners, the main disadvantage is the non-optimality of the planned path: in fact, they do not generate also a timing law, but only a collision-free geometric path. This means that in these planners there is no optimization either on the smoothness of the path or on the kinematic profiles of velocity, acceleration and final jerk. This results in higher consumption for the robot since, as also highlighted for the graph search-based planners, there are usually unnecessary kinematic profiles present. In addition, the final path may have abrupt changes in direction (because it is not smooth) that are not safe for the robot and the environment around it, and generally resulting in higher energy consumption.

All these problems are avoided in optimized-based planners, as they optimize the trajectory in terms of length, execution time, smoothness during its planning and they also consider the kinematic constraints about velocity, accelerations and jerk, thus obtaining smoother trajectories free (or almost free) of unnecessary movements and actions.

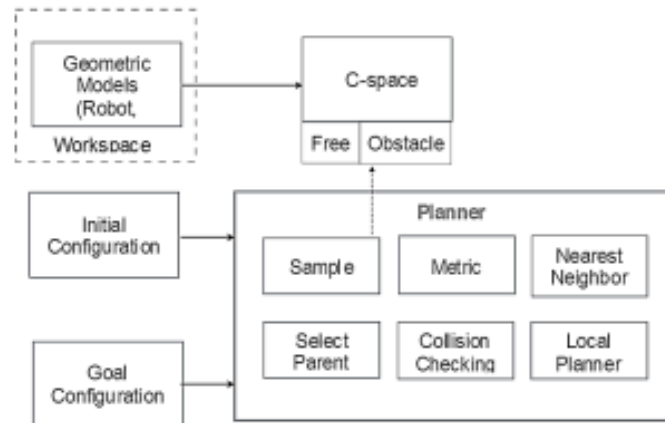


Figure 4.2: General structure of a sampling-based planner: the inputs are the initial state, the final state and the configuration space. Thanks to the sample procedure and the collision checks, it looks for a collision-free path. [8].

### 4.2.1 RRT algorithm

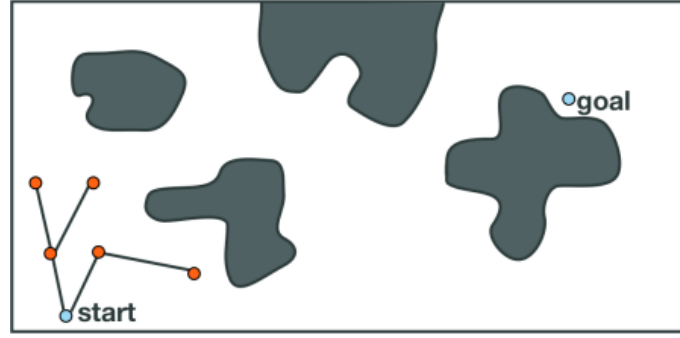
Rapidly-Random Tree (RRT) is one of the first single-query sampling-based planners with simple logic that is effective even in complex environments [2], [3], [8], [12], [21], [26]. In this planner, a tree is constructed so that it is expanded from the initial point of the path until the required end point is reached.

In more detail, the steps to obtain the final collision-free path are:

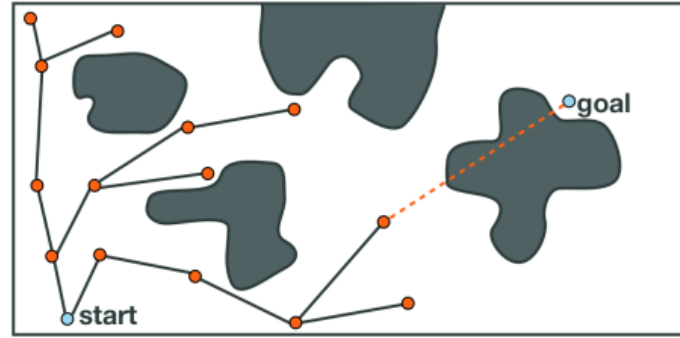
- the initial point of the path  $q_{\text{start}}$  coincides with the initial configuration of the algorithm.
- a random state is selected from the configuration space following the sample procedure.
- a collision check is performed. If the state is in collision it is discarded and restarted from the previous step.
- On the other hand if the state is not in collision, using the Nearest Neighbor (NN) it is found the  $q_{\text{near}}$  according to metric.
- $q_{\text{rand}}$  and  $q_{\text{near}}$  are connected thus giving a new configuration to the tree within a certain distance  $\delta$  from  $q_{\text{near}}$ . If it does not meet the requirement,  $q_{\text{new}}$  is discarded.
- another collision check is performed at the path between  $q_{\text{near}}$  and  $q_{\text{new}}$ . If it is not satisfied,  $q_{\text{new}}$  is discarded and the procedure is restarted.
- everything is repeated in a loop until  $q_{\text{new}}$  corresponds to the final state of the path  $q_{\text{goal}}$ .

The overall logic of the RRT algorithm is quite simple; moreover, it turns out to be probabilistically complete. In fact, if ideally the planning time is infinite, it would surely find the solution, assuming it exists. However, as mentioned before, being a sampling-based planner it does not consider kinematic constraints and therefore the final path is not optimized neither as smoothness nor as velocity, acceleration and jerk profiles (since only the geometric profile is planned without the corresponding timing law).

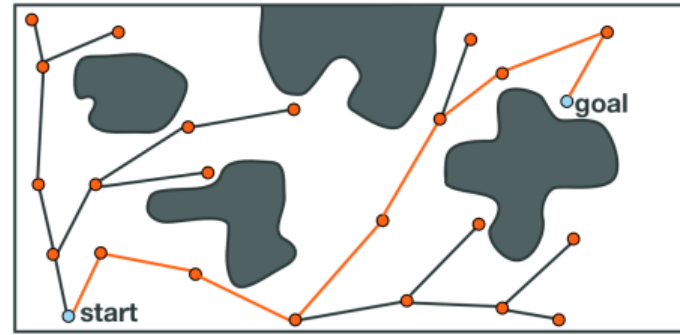
This is one of the earliest designed planners, at the state of the art there are several improvements of it such as RRT\* or RRT connect [6] which use this planner as a basement and then they optimize the steps to achieve better final planned path performance (as will be explained in more detail in the next sections).



(a) Starting the tree from  $q_{start}$ .



(b) Expanding the tree to reach  $q_{goal}$ .



(c) The path joining  $q_{start}$  and  $q_{goal}$  exist and it is found.

Figure 4.3: Planning the collision-free path through RRT algorithm [21].

### 4.2.2 RRT connect and bi-directional RRT algorithms

RRT connect represents one of the enhancements made to the simpler RRT algorithm to speed up its path generation time [3], [6], [14].

In this algorithm two trees are constructed instead of one: the first starts from the initial state  $q_{start}$ , while the second starts from the final state  $q_{goal}$  of the path. Then the two trees are expanded simultaneously with the goal of finding a common state in order to connect and find a collision-free path.

During the expansion the steps are the same as in the RRT algorithm, so there are collision checks performed step-by-step.

This algorithm retains the advantages of the simpler RRT, indeed it exploits the same tree development logic (so it is still probabilistically complete, computationally quick and quite effective even in complex environments). The notable advantage is that by developing two trees in parallel, the planning time is significantly reduced.

Another variant of the RRT, very similar to the RRT connect, is the bi-directional RRT [26], [33], [36]. Again two trees are developed simultaneously, one starting from the initial state  $q_{\text{start}}$  and one starting from the final one  $q_{\text{goal}}$  of the required path. The main difference is that while in the connect RRT the two trees interact with each other and each step seeks to join, in the bi-directional RRT the two trees are totally independent while seeking for a common configuration.

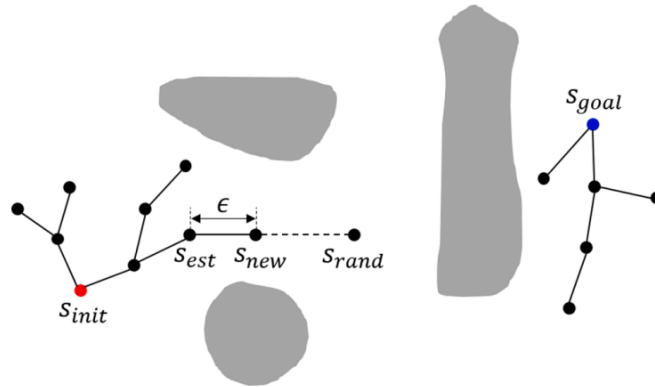


Figure 4.4: Two trees development in the bi-directional RRT algorithm [36].

### 4.2.3 PRM algorithm

The Probabilistic Roadmap Method (PRM), unlike the RRT algorithm, is a multiple-query sampling-based motion planner [2], [3], [8], [12], [13], [21], [26], [33].

For this reason, the main advantages of the PRM planner are present when there are multiple paths to be generated in the same scene. In fact, since it is a multiple-query planner, the data of the planned paths will be stored from time to time, thus resulting in reduced planning times for subsequent paths, since the computations already performed do not have to be repeated.

In the PRM planner, the configuration space is randomly sampled in order to build a roadmap of the free space  $C_{\text{free}}$  and then a collision-free path can be found within it.

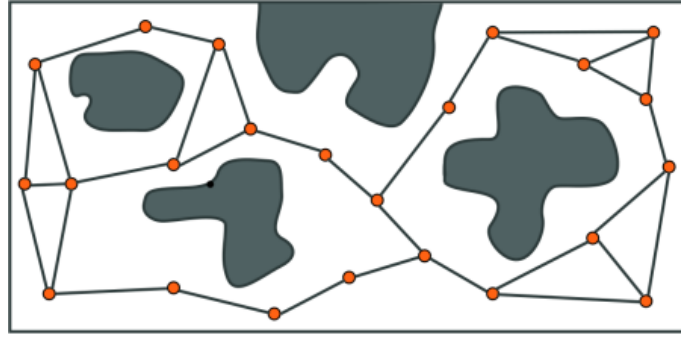
More in deep, the steps to build the roadmap for this algorithm are:

- following the sample procedure, a  $q_{\text{rand}}$  is selected in the configuration space.
- a collision check on  $q_{\text{rand}}$  is done. If the configuration is collision-free, it is added to the roadmap, otherwise it is discarded and the algorithm goes back to the previous step.
- if  $q_{\text{rand}}$  is collision-free, then all the possible configurations  $q$  in a specific range  $\delta$  from  $q_{\text{rand}}$  are searched.
- all these new configurations are connected to  $q_{\text{rand}}$  and then a collision check is done to each one. The colliding paths are disconnected from  $q_{\text{rand}}$ , while the collision-free ones are maintained into the roadmap.
- the iteration goes on until a satisfactory number of points in the roadmap is reached. The more points, the better the planner's performance.

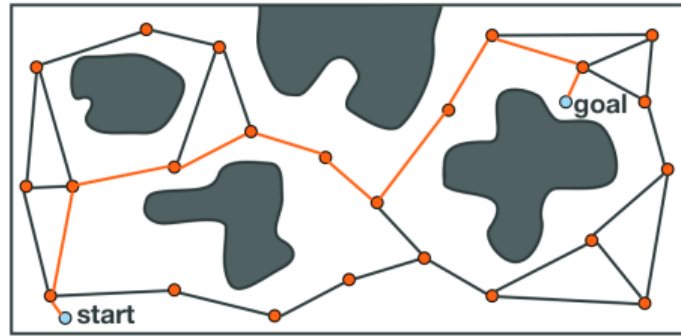
Once the initial roadmap is generated, the shortest collision-free path can be found between the initial pose  $q_{\text{start}}$  and the final pose  $q_{\text{goal}}$  using a graph-search algorithm.

Since it is a multiple-query planner, the roadmap is computed initially and then maintained for all the paths to be planned later. This results in a significant reduction of planning time in subsequent paths.

The PRM planner has a good behaviour also in high dimensional state spaces, it turns out to be probabilistically complete and its accuracy will depend on the amount of configurations that are present in the roadmap. Since this planner is a sampling-based one, it does not plan the trajectory in dependence of time, but only as a geometric path. The denser the initial roadmap, the shorter but also computationally heavier this path will tend to be.



(a) Initial roadmap in the configuration space.



(b) After creating the roadmap, the PRM planner finds the collision-free path.

Figure 4.5: Collision-free path planning using PRM planner [21].

#### 4.2.4 RRT\* and PRM\* algorithms

The RRT and PRM algorithms have proven to be effective and with good performances in even complex environments. In fact, even in complicated scenes the success rate and planning time have generally satisfactory values. The drawback is the non-optimality of the planned path and the lack of a timing law, so it is needed a second step to obtain the final trajectory in dependence of time. Both planners also turn out to be probabilistically complete.

To overcome these limitations, there are a variety of innovative algorithms in the state of the art. Looking always in the class of probabilistically complete algorithms, the most widely used are RRT\* and PRM\*, which compared to RRT and PRM also guarantee asymptotic optimality [6], [8], [12].

In RRT\*, the underlying procedure is the same as in RRT. The main difference is that each node is associated with a cost to be minimized. This makes the planned path much smoother and converges asymptotically to the optimal solution. On the other hand, there is a significant increase in computational times since the weight of each node must also be considered.

Another improvement over RRT is the possibility of tree rewiring. This causes a higher number of collision checks, from which comes an increase in computational cost [11].

Also PRM\* provides enhancements with respect to the basic PRM algorithm. In this case, the radius by which the initial roadmap is created is made variable and the logic by which nodes are added to it is also changed. These variations result in much denser initial roadmaps, providing much more accurate solutions that tend to be optimal as the number of nodes increases. Again, the sharp increase in planner performance causes a lengthening of path planning time.

### 4.2.5 SBL algorithm

The Single-query Bi-directional probabilistic roadmap planner with Lazy collision checking (SBL) is another improvement widely used of the bi-directional RRT and PRM algorithms [1], [26].

In the SBL planner, there are two trees that are built in the free space  $C_{\text{free}}$  in order to find the collision-free path from the initial configuration  $q_{\text{start}}$  to the final  $q_{\text{goal}}$ . Step by step, the planner auto-regulates the size of the radius  $\delta$  based on the area of free space  $C_{\text{free}}$  in which it is located. In the areas that are safer,  $\delta$  has a higher value so that it is computationally faster. In areas closer to obstacles and thus risky, there is a greater need for precision and accuracy and thus the value of  $\delta$  is lower.

Compared to the PRM algorithm, collision checks are minimized by performing "lazy collision checking". This means that collision checks are postponed until they are absolutely necessary, usually meaning that they are performed once a path connecting the two trees is found. Where colliding paths are found, they are recomputed only in that local area in order to avoid the obstacle. This modification is developed because it is seen that in the PRM most of the time is spent performing collision checks, as they are performed at each step during the planning path. This is done even though there are regions of the path where they are practically useless, as they are definitely in free space.

Operating with this logic, it is seen that the generated paths are geometrically similar to those of the PRM planner (comparing parameters such as path lengths and failure rates), with lower planning times.

### 4.2.6 Deterministic Chekhov algorithm

The deterministic Chekhov algorithm [9] is a sampling-based planner that uses a graph-based representation called Tube-based Roadmap (TRM) to find a collision-free path. In this graph, there are nodes representing the pre-grasping poses required during the task and random collision-free poses to avoid obstacles. The arcs of the graph are not single trajectories, as in typical roadmap-based planners, but are flow tubes, i.e., families of possible paths. The set of these trajectories is called Quality Control Plan (QCP) and it considers also the control policies and dynamic constraints. These QCPs are very complex and computationally heavy; for these reasons, they are computed offline initially and stored. Subsequently, an All-Pairs Shortest Path (APSP) algorithm based on the QCP computed previously is used to calculate the collision-free path.

Thanks to this structure, these planners are fast and efficient even in complex environments, and they are also reactive and robust in the case of both strong disturbances (e.g. dynamic environments or goal state changes) and soft disturbances, such as noisy sensor inputs. However, since these algorithms are based on a roadmap that is obtained by sampling the configuration space, the final trajectory is not optimal. Furthermore, this planner requires considerable initial computational effort to generate QCPs and it is also complex to implement in planning frameworks such as MoveIt, due to its integrated architecture and sophisticated control policies.

In recent years, many enhancements have been made to this planner, in particular the probabilistic Chekhov [4], an optimized-based planner that is based on the deterministic Chekhov algorithm. It presents a second step where it optimizes the generated collision-free path based on collision probabilities with both static and dynamic objects in the scene. Moreover, it tries to maximize offline computations in order to result in a planner that is as reactive to changes as possible.



### 4.3 Optimized-based algorithms

Sampling-based planners are fast and quite effective even in complex environments. However, they have major limitations on the final smoothness of the paths and their optimality regarding kinematic constraints such as velocities, accelerations, and jerks. This means the final trajectories that will be executed by the robot are worse, in the sense that they will be longer, less safe and usually with higher execution times. In addition, there is a possibility that there may be unnecessary and abrupt robot movements, thus being less safe and with generally higher energy consumption.

For these reasons, algorithms have been developed that compute the final trajectory as the solution of an optimization problem, considering a cost function with various constraints (e.g., maximize the distance to obstacles in the environment, consider limits in the velocity, acceleration and jerk profiles, and so on).

The main disadvantage in optimized-based planners turns out to be the computational complexity. It grows considerably compared to sampling-based planners, thus resulting in longer planning times and making them unsuitable for online planning. Another major disadvantage present in most optimized-based planners is the dependence on initial conditions. This leads them to trap into local minima rather than global minima, thus causing an increase in failure rates compared to sampling-based planners.

The next sections discuss the two optimized-based motion planners that are discovered first and studied for several years and thus also more widely used (CHOMP and STOMP) and also other recently developed solutions. The latter have provided a significant upgrade to the state of the art in terms of trade-offs path optimality, planning times and failure rates in collision-free path planning. In particular, it is present a complete description of TrajOpt which is an innovative optimized-based planner highly studied in recent years. It is used in this thesis work to develop a comparison with the performance of the current planners used in typical industrial robotics scenes.

### 4.3.1 CHOMP algorithm

The Covariant Hamiltonian Optimization Motion Planner (CHOMP) optimizes the trajectory using a functional gradient-descent technique [24].

In this logic, the trajectory is updated step-by-step taking into account the gradient value of the cost function  $F_{\text{cost}}$ :

$$\tau^{i+1} = \tau^i - \frac{1}{\eta} A^{-1} \nabla F_{\text{cost}}[\tau^i] \quad (4.1)$$

in which  $\eta > 0$  is the step size,  $A = D^T D$  where  $D$  is the finite differentiation matrix,  $\nabla F_{\text{cost}}[\tau^i]$  is the gradient of the cost function and  $\tau^i$  is the trajectory after  $i$  iterations.

This cost function  $F_{\text{cost}}$  to be minimized is given by the sum of two distinct parts:

$$F_{\text{cost}} = F_{\text{smooth}} + F_{\text{obst}} \quad (4.2)$$

- $F_{\text{smooth}}$  is the smoothness functional that is used to penalize too high values of velocity and acceleration. In this way, the final path is smoother.
- $F_{\text{obst}}$  is the obstacle functional which is used to penalize points close to obstacles. For this reason, a threshold distance  $\epsilon$  is fixed and this guarantees to stay sufficiently far away from them.

This algorithm depends on the gradient of the cost function  $F_{\text{cost}}$ , due to this the final trajectory is very sensitive to the initial conditions of the problem and it can trap in local minima causing an increase in the failure rates.

For this reason, it is usually used as path post-processing [16]: by optimizing an initial collision-free geometric path computed by a sampling-based planner, the probability of falling into local minima is greatly reduced thus achieving better performance. This provides optimal trajectories as path length, considering also the kinematic constraints and with lower failure rates. All at the expense of planning time, since there is a cascade of 2 planners.

### 4.3.2 STOMP algorithm

The Stochastic Trajectory Optimization Motion Planner (STOMP) is an optimized-based motion planner that uses a stochastic trajectory optimization framework [7], [10].

This algorithm starts from an initial trajectory, even not collision-free, and at each step generates  $N$  noisy trajectories, as seen in Figure 4.6.

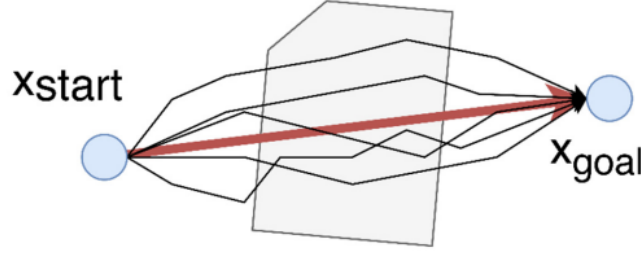


Figure 4.6: Noisy trajectories around an obstacle starting from an arbitrary initial one (in red) [7].

At each iteration, a cost function based on an obstacle cost, smoothness cost and control cost, is minimized, in which  $\tilde{\theta} = \mathcal{N}(\theta, \Sigma)$  is a noisy parameter vector with mean  $\theta$  and variance  $\Sigma$ , and  $q(\tilde{\theta}_i)$  is an arbitrary state-dependent cost function which can include costs, joint constraints and torques.

$$\min_{\tilde{\theta}} \left( \sum_{i=1}^N q(\tilde{\theta}_i) + \frac{1}{2} \tilde{\theta}^T R \tilde{\theta} \right) \quad (4.3)$$

This part is very similar to the one already seen previously for the CHOMP planner, with the crucial difference that here it is not gradient-based, so there is no risk to trap in local minima.

Iterations go on until the cost function  $q(\tilde{\theta})$  converges to a minimum value  $Q$ . This procedure explores the space around the initial trajectory looking for the collision-free one among the  $N$  generated. All these generated noisy trajectories are called rollouts and the chosen one is the collision-free rollout that minimizes the cost function.

In the case where none of the noisy trajectories are collision-free, the algorithm restarts from the trajectory that minimizes the cost function and then generates additional  $N$  noisy trajectories starting from the latter. It proceeds until an optimized collision-free one is obtained, as seen in Figure 4.7.

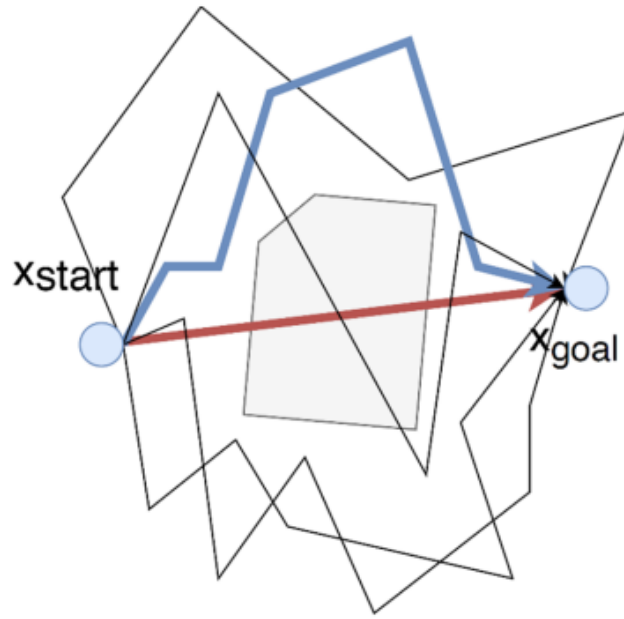


Figure 4.7: Final optimized collision-free trajectory (in blue) starting an arbitrary initial one (in red) [7].

This type of planner has longer planning times and lower precision than CHOMP, especially in cases where there is high noise variance. However, it has higher success rates, thanks to the fact that it is not a gradient-based algorithm.

In recent years, improvements to STOMP planners have also been developed, such as Cartesian Constrained STOMP [16]. In this newer algorithm, it is possible to add constraints in the cartesian space and not only in the joint space as is the case of STOMP. To obtain this flexibility, the algorithm relies on innovative methods with regard to noise generation and cost function computation.

### 4.3.3 TrajOpt algorithm

The TrajOpt planner is a novel approach for robotic motion planning among obstacles. The core of this algorithm is a Sequential Quadratic Program (SQP), a convex optimization procedure which penalizes collisions with a hinge loss and uses an efficient formulation of the collision-free constraints. This procedure allows also continuous-time safety and enables the algorithm to reliably solve problems involving thin, complex and dynamic obstacles [5], [31], [32].

There are two key concepts to optimize collision-free path planning.

The first is the numerical optimization method. TrajOpt uses a sequential convex optimization, with  $\ell_1$  penalties for equalities and inequalities constraints. This solves a series of convex optimization problems that approximate the cost and constraint of the true problem, which is highly non-convex. Among the constraints of the optimization problems, in addition to kinematics constraints such as joint limits, speed limits, end-effector initial and target poses, it is also possible to consider the dynamics of the problem, e.g., joint torques and contact forces. Moreover, there is the safe distance,  $d_{safe}$ , defined by the user and entered as a hard constraint; it is the minimum distance that is maintained from any object in the scene. To speed up the algorithm, a second distance called  $d_{check}$  greater than  $d_{safe}$  is defined. The collision checker is activated only when objects are at a distance lower than  $d_{check}$ . Thanks to this logic, no time is wasted in useless computations, because the robot configuration is surely safe ( $d > d_{check} > d_{safe}$ ). Thanks to the SQP, the algorithm is not gradient-based, resulting in higher success rates. Furthermore, fewer iterations are needed to converge to an optimal solution. On the other hand, however, solving this type of problem results in more time-consuming computations in each iteration.

The second key concept is the collision checking method. To do this, the planner relies on the signed distance, which is computed using convex-convex collision detection. As can be seen in Figure 4.8a, if the signed distance is greater than zero ( $T > 0$ ), the configuration is collision-free; otherwise, the configuration is in collision. With this method, it is also possible to ensure the continuous-time safety of a path by considering the swept-out volume (Figure 4.8b). In this case, the swept-out volume of the dynamic object A is composed (i.e., the volume between two poses  $A(t)$  and  $A(t+1)$ ) and then it is taken into account in the computations of the signed distance with object B. Convex collision checking compared to distance fields, which is used in planners such as CHOMP, guarantees greater accuracy and precision in the computations. Indeed, in the distance field, each link of the robot is approximated as a union of spheres. In this way, collision checking between links and obstacles does not depend on the complexity of the environment, but results in less accurate calculations.

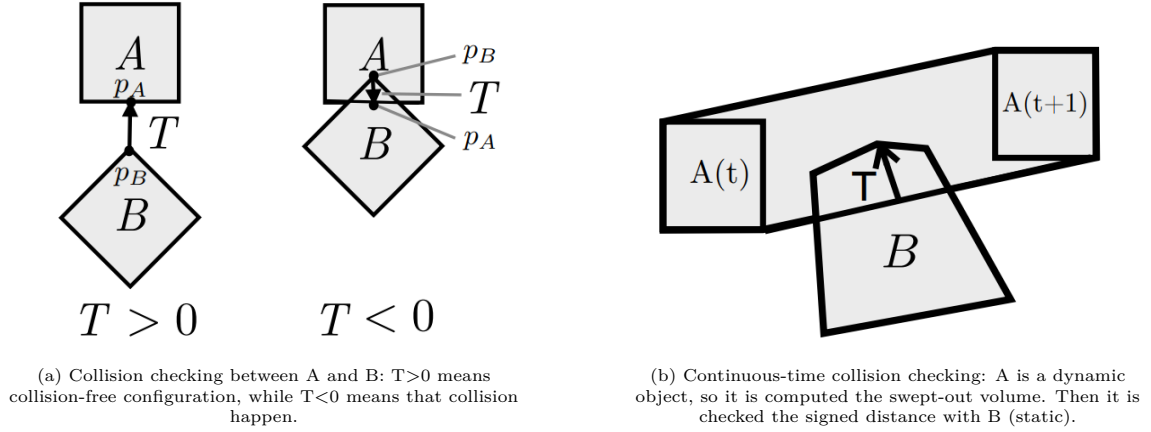


Figure 4.8: Concepts of signed distance and swept-out volume between two objects A and B [31].

The combination of an SQP optimization problem and convex-convex collision detection results in an algorithm that requires more computation per iteration, but with fewer iterations. The final solutions are optimal and effective even in complex environments, albeit in some cases can trap in local minima. On the other hand, due to the convex-convex collision checking, planning times are closely related to the complexity of the scene.

Finally, it is important to point out that this type of planner is already implemented in the Tesseract framework. Tesseract is an open-source planning framework that is very common in the industrial robotics field and will be explained deeply in Chapter 7. This implementation provides a number of advantages, including flexible scene management and the possibility to achieve multiple target poses sequentially in the same path.

### 4.3.4 LCQP algorithm

The Linear Constraint Quadratic Program (LCQP) algorithm solves the problem of collision-free path planning setting it as an optimization one in a similar way to TrajOpt [17].

LCQP planner is usually used to post-process an initial collision-free path, since it presents a first part where it is able to remove the redundant motions and then optimize it. This does not exclude that it can be used as an optimized-based planner, by avoiding the first part.

Assuming the robot has  $n$  joints, the configuration space is  $C \subset R^n$ . Then if the number of path points in the trajectory is  $m$ , the trajectory can be expressed as  $\xi \in R^{mn}$ .

This algorithm treats collision avoidance and task constraints as hard constraints rather than optimization terms. For this reason, the cost function only contains the smooth term  $f_{smooth}(\xi)$  to measure dynamical quantities across the trajectory.  $f_{smooth}(\xi)$  can be computed as a sum of squared derivatives. The problem is then set as:

$$\begin{aligned} \min_{\Delta\xi} \quad & \frac{1}{2} \Delta\xi^T \mathbf{M} \Delta\xi + g^T \Delta\xi \\ & \mathbf{C} \Delta\xi \geq \mathbf{0} \\ & \mathbf{S} \Delta\xi = \mathbf{0} \end{aligned}$$

in which the linear constraints take into account collision avoidance thanks to the matrix  $\mathbf{C}$  and the start and final points of the trajectory thanks to the matrix  $\mathbf{S}$ . It is possible to obtain linear constraints for collision avoidance thanks to a collision backtrack, which converts the collisions into linear constraints in order to ensure that there will be no more collisions at those points.

This optimized-based planner achieves excellent results in terms of trajectory quality based on execution time, final smoothness and task constraints. However, it has significantly longer planning times due to the numerous hard constraints.

### 4.3.5 B-spline algorithm

The B-spline planner [15] is an effective trajectory generation algorithm to shorten and smooth initial jerky paths. The initial path can be anyone, either one in collision with the environment or one already collision-free generated by a sampling-based algorithm.

This planner is divided into two core parts: a two-layer local adjustment of the initial path and a fast trajectory pruning. First of all, several control points are selected on the initial path. If there is a colliding segment between two control points, the two-layer local adjustment is used. In the first layer, the local path is adjusted to obtain a collision-free one. To do this, it is possible to apply a fast "optimist" strategy but not entirely reliable, or a "pessimist" strategy that is reliable but increases the trajectory's roughness. In simple environments, it is recommended to use the former to achieve the best possible smoothness, while in complex environments the latter to ensure greater safety. If this first layer fails to find a local collision-free path, the second layer is used. This second layer allows a split and merge strategy between the control points. By adjusting the control points suitably, collision-free paths can be found. Finally, once the collision-free path has been obtained, there is the final part in which there is a pruning method in order to remove the remaining redundant motions.

This algorithm allows the generation of optimized collision-free trajectories, which respect kinematic constraints and guarantee excellent smoothness. Indeed, thanks to the method of optimization, the final path belongs to the continuity class  $C^2$ , i.e., paths with velocity and acceleration profiles that are continuous. Furthermore, this algorithm does not depend on gradients, so it does not trap in local minima. However, the two-layer structure followed by a pruning method, makes the algorithm complex and therefore increases considerably the planning times and it is closely related to the complexity of the scene. These last features do not allow dynamic scene management and online motion planning.



## 4.4 Post-processing algorithms

Post-processing algorithms are hybrid planners with the purpose of combining the advantages of sampling-based and optimized-based planners, reducing the weight of defects [16], [18].

In these algorithms there is the generation of an initial collision-free path by a sampling-based planner. This collision-free path is then used as the initial condition in an optimized-based planner in order to obtain the final optimized trajectory.

With this procedure, it is possible to plan a final optimized trajectory thanks to the optimized-based planner. Moreover, the success rates of the post-processing algorithms are higher with respect to the optimized-based ones, since the initial input is generated by a sampling-based planner. This input is already collision-free so it is closer to the final optimized trajectory. Thanks to this property, the probability to trap in local minima drops and so the success rates are higher.

Computational times generally turn out to be quite high, since there are two planners in cascade, however with the proper combination of planners they are not too large. This is because the first sampling-based step is usually fast, while the second step takes a shorter time with respect to the single optimized-based, since it starts from an input that is closer to the final optimal solution.

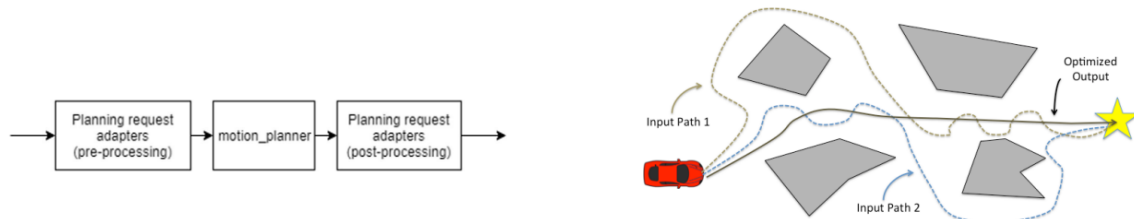


Figure 4.9: Structure of a post-processing planner: cascade of two motion planners (left) [16]; final optimized-trajectory starting from collision-free inputs (right) [18].

# Chapter 5

## Simulation framework

Each of the collision-free motion planning algorithms explained in Chapter 4 requires a planning framework. This planning framework is responsible to execute the algorithm and generate the collision-free path. In this thesis, the focus is on the MoveIt and Tesseract planning frameworks. The former is currently used in COMAU and contains a library called OMPL, in which the main sampling-based planners can be executed. This planning framework turns out to be one of the most widespread in the industrial robotics fields thanks to the features that will be illustrated in Chapter 6. The latter has been considered as the TrajOpt planner is already implemented in this planning framework. This planner is highly innovative and, for this reason, it is decided to develop a comparison between the performance of the current planners used and this one in typical industrial robotics scenes.

Furthermore, it must be underlined that both planning frameworks are ROS-based. The next sections explain in detail what is ROS, how it is structured and why its features make it widely used in robotics.

Then the following chapters are devoted more specifically to the MoveIt and Tesseract planning frameworks and to the experimental part for comparing the planners with the chosen benchmarks.

### 5.1 ROS

Robot Operating System (ROS) is an open-source framework widely used for the development of robotic applications [27]. Although it contains "operating system" in its name, it is not a traditional operating system but is rather a set of libraries, tools and conventions that optimize communication and cooperation between elements of a complex system such as a robotic system.

ROS provides a communication middleware that allows nodes (independent execution units) to exchange data synchronously or asynchronously through publisher/subscriber type mechanisms.

Thanks to this structure ROS has a variety of advantages:

- modularity: each node is launched and executed independently of the others. Thanks to this structure, it is possible to design, develop and test individual nodes. This makes them reusable for other purposes since each node has a specific function.
- portability: this is another inherent feature due to the structure of independent nodes. Each node performing a function on its own can be carried and used for other purposes.
- effective communication: the messaging system of ROS is simple and scalable. Nodes can communicate with each other via topics or services, thus enabling the transmission of sensors, status, control or any other type of information.
- flexibility: this is both a consequence of the previous features and the fact that it supports a variety of programming languages, including the most common Python and C++.
- graphics: ROS supports simulation of environments such as RViz or Gazebo, thus also enabling graphical simulation by loading models such as URDFs (Unified Robot Description Format) that describe the geometry and kinematics of the robot.

## 5.2 ROS nodes and topics

Nodes are the main core of ROS. As explained earlier, they are launched and executed independently of the rest of the system and they can be written in various programming languages including Python and C++. Each node has its own source file where it is defined what it has to do, its publishers and its subscribers. Then there is a launch file that is responsible for executing one or more nodes. Indeed there is the possibility of launching multiple nodes with the same launch file in order to simplify the execution and to speed up time.

Nodes can communicate with each other using publisher-subscriber protocol logic. To do this, a topic is instantiated among the nodes that contain the data to be transferred [30]. The nodes connected to the topic can be of two types: publishers that send messages/data on the topic and subscribers that receive messages/data from the topic. The final structure of the ROS environment thus consists of nodes communicating with each other through topics, as reported in Figure 5.1.

This structure is effective when the environment is relatively small and simple. If the system grows in complexity, services and actions are frequently used to carry out communication between nodes.

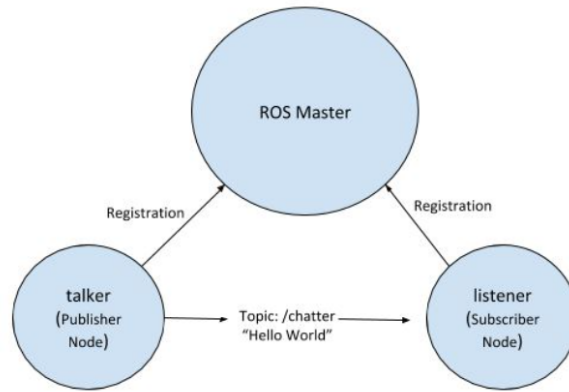


Figure 5.1: ROS topic structure: communication between a publisher node and a subscriber node [30].

### 5.3 ROS services

Communication by topic, thus with a publisher/subscriber structure, is very flexible and allows many-to-many communication. However, it has the limitation of being a one-way communication, since the nodes that send or receive the message are defined in principle at the very beginning.

This way of communication is not always suitable, so there is the possibility of using other communication routes such as services.

Services use a request/reply type of communication between a client and server node and it is based on a message pair [29]. A ROS node offers a service under a string name, another node, denoted as a client, calls the service by sending it a request message and then it waits for a reply message from the service.

This service-client procedure is a synchronous type of communication: the client node makes the service request and then it waits until it receives a response. During this waiting time the node is stationary and it cannot perform other tasks. This makes this type of communication suitable only for occasional tasks, otherwise there is a risk of slowing down the system considerably.

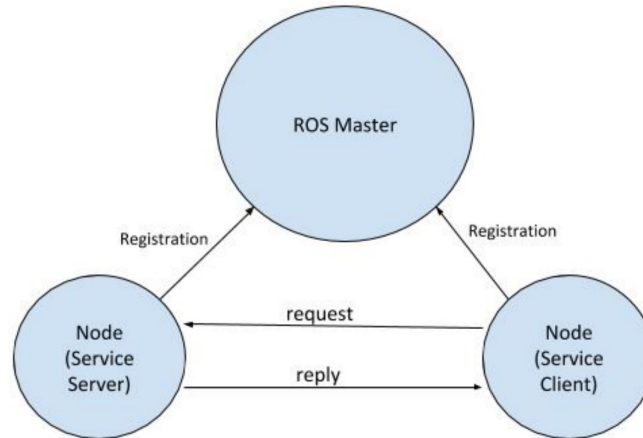


Figure 5.2: ROS service structure: communication between a service client node and a service server node [29].

## 5.4 ROS actions

ROS actions are server-client messages that function much like services: an action client sends a request message to the server for a particular action and it waits for a second response message [28]. The key difference from services is that the actions are asynchronous. This means that once the request is sent to the action server, the client node continues to work and in parallel it waits for the server's response that will arrive in a second time (asynchronous).

The messages that are used in actions have all the same structure:

- **goal:** it is the initial message that sends the action client to the action server. It contains information about the request goal.
- **feedback:** they are various messages that the action server sends to the action client to update it on the progress regarding the achievement of the goal.
- **result:** it is the final message that the action server sends to the action client once the goal is completed.

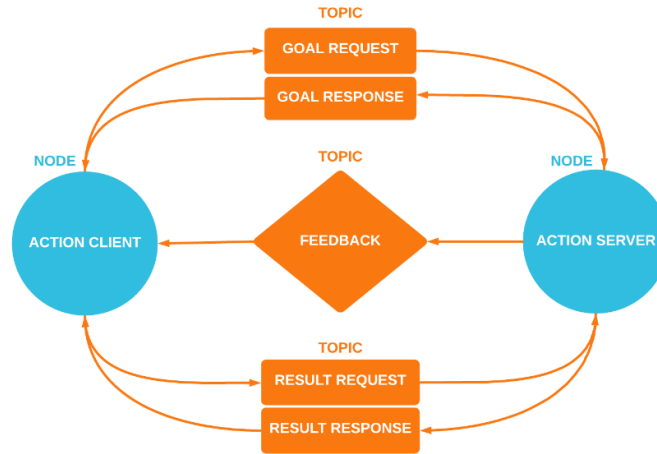


Figure 5.3: ROS action structure: communication between an action client node and an action server node [28].

## 5.5 ROS RViz

The ROS visualization tool RViz is a graphical user interface that allows the user to visualize information about the robot and the evolution of the environment around it.

In this thesis work RViz is subscribed by the topic that contains the values of the robot's joints (called `joint_states`) so as to visualize its pose at every instant. In addition, the user can see the evolution of the scene if it is acquired by visual sensors and thus also keep track of the dynamic changes by appropriate octomaps.

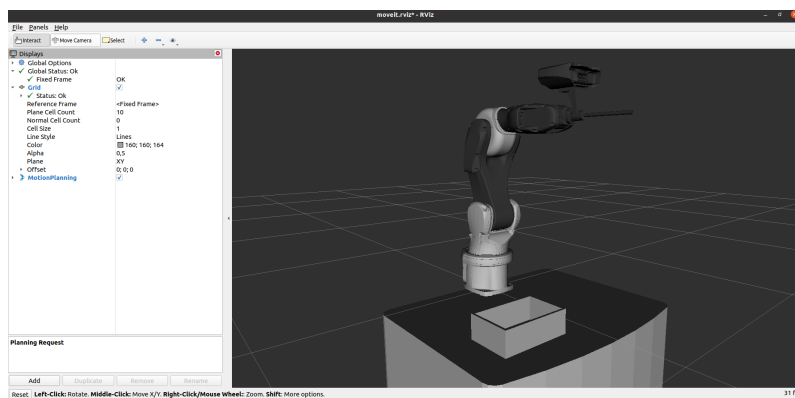


Figure 5.4: ROS RViz graphical interface.

## 5.6 Roboshop

Roboshop is an internal COMAU software for simulating the C5G controller of COMAU robots.

There is a graphical interface that simulates the movements of the robot and the virtual TP5 that simulates the actual teach pendant that is present to control each robot.

Thanks to Roboshop, it is then possible to realistically simulate the interface with the robot. For example, the user can plan paths on MoveIt and simulate them on Roboshop before utilizing the real robot. Being the simulator of the C5G controller hardware on board the robot, with Roboshop it is possible to derive much more accurate values regarding path execution times, velocities, accelerations, and jerks of the robot, since it takes into account more real-world factors than the simpler simulations on MoveIt or Tesseract environments.

On Roboshop it is possible also to control the input and output signals from the robot, so for example it can be assigned true/false values to the digital inputs and/or outputs. These digital outputs are the basis of the change gripper management developed and which will be discussed more in deep in Chapter 13.

In Figure 5.5 it can be seen the virtual interface that faithfully simulates the teach pendant used to control COMAU robots, more in detail it is shown a Racer3-0-63.



Figure 5.5: Racer3-0-63 Roboshop graphical interface.

# Chapter 6

## MoveIt

MoveIt [19] is the most widely used software for manipulation in industrial robotics and it has been used over 150 robots. Furthermore, it is open-source for industrial and research use. MoveIt is recognized as the state of the art platform for mobile manipulation thanks to its main features:

- motion planning: it can generate high degrees of freedom paths through cluttered environments. It is also able to avoid local minimums.
- manipulation: it can analyze and interact with the environment thanks to grasp generation.
- inverse kinematics: it can solve for joint positions for a given pose. It works also for over-actuated arms.
- control: it can execute time-parameterized joint paths to low-level hardware controllers through common interfaces.
- 3D perception: it is able to connect the depth sensors and point clouds thanks to octomaps.
- collision checking: it can avoid obstacles using geometric primitives, meshes or point cloud data.

The main tool in this framework is the MoveIt Setup Assistant. Thanks to the MoveIt Setup Assistant, it is possible to configure the robot and then plan and execute the desired paths.

To get started, is needed to upload the robot model via URDF (Unified Robot Description Format). Then the user has to generate the collision matrix, which is the matrix that considers all possible collisions in the loaded URDF. Next, it is possible to specify the robot's virtual/fixed joints, planning groups, specific poses for the robot's use and other information useful for 3D perception.



Once all the information about the robot has been entered, the MoveIt Setup Assistant generates the SRDF (Semantic Robot Description Format) file and other configuration files that are used in the MoveIt pipeline.

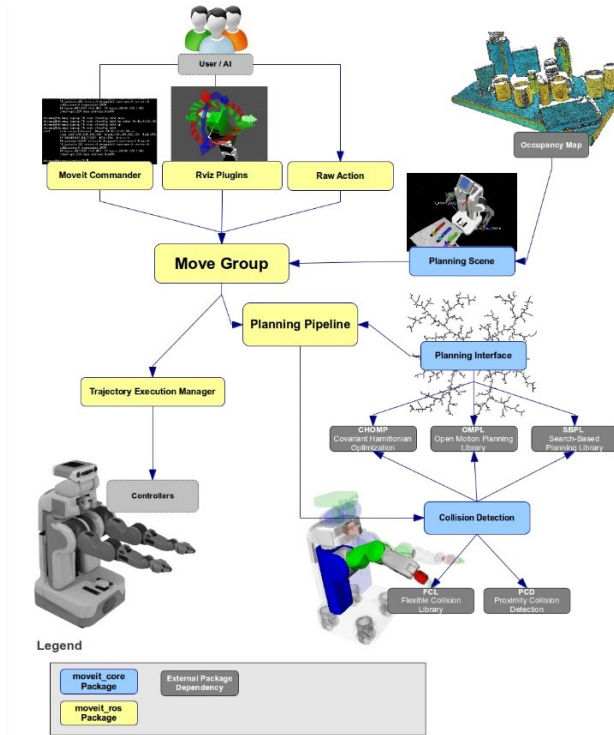


Figure 6.1: MoveIt general system architecture [19].

The primary node provided by MoveIt is the `move_group`. The high-level system architecture for this node is shown in Figure 6.1, while the node structure is shown in Figure 6.2.

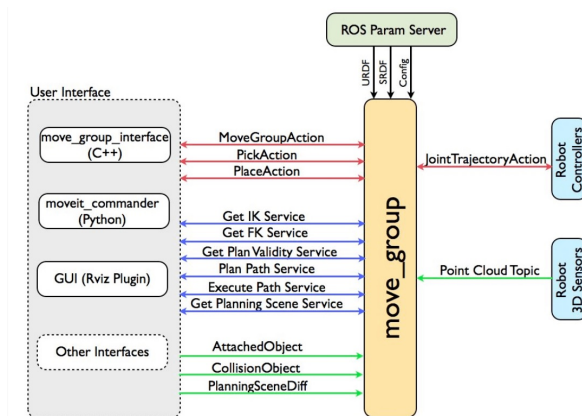


Figure 6.2: MoveIt `move_group` architecture [19].

This node serves as an integrator: pulling all the individual components together to provide a set of ROS actions and services for users to use.

For the user it is very easy to access the actions and services provided by `move_group`, since it can be configured using the ROS param server. From this server, the `move_group` ROS node gets three pieces of information:

- URDF: the robot description parameters.
- SRDF: the robot description semantic parameters. As already said, the SRDF is generated thanks to the MoveIt Setup Assistant.
- MoveIt configuration: other data as joint limits, kinematics, motion planning and perceptions. Also these config files are automatically generated by the MoveIt Setup Assistant and stored in the proper directories.

The `move_group` node talks to the robot through ROS topics and actions. Thanks to them, it is able to get all the current state informations: the joint positions, the point clouds of the scene, the data coming from the other sensors and from the controllers on the robot, and so on.

## 6.1 Motion planning

MoveIt can communicate with many motion planners from different libraries thanks to a plugin interface. This interface to the motion planners is through a ROS action or service offered by the `move_group` node. The interface is initially configured by the MoveIt Setup Assistant automatically.

The motion planners that are available on MoveIt are those belonging to the OMPL library, the Pilz industrial motion planner and CHOMP.

### 6.1.1 OMPL

OMPL (Open Motion Planning Library) is an open-source library containing many types of randomized motion planners, so basically those planners belong to the category of sampling-based motion planners. MoveIt integrates directly with OMPL and all motion planners that belong to this library are set as planners by default. OMPL is abstract, meaning that the planners in this library have no concept of robot description. When MoveIt integrates with OMPL, it configures the library and provides OMPL with the back-end to work with robotics problems.

## 6.2 Planning scene

The planning scene stores the current state of the robot and the world around it. It is stored inside the move\_group node, more specifically by the planning scene monitor.

The architecture of this node is shown in Figure 6.3, particularly the planning scene monitor listens to:

- state information: stored on the joint\_states topic.
- sensor information: thanks to the world geometry monitor. It builds the world geometry using the occupancy map monitor and then it saves a 3D representation of the scene on the planning\_scene topic.
- world geometry information: directly from the user on the planning\_scene topic.

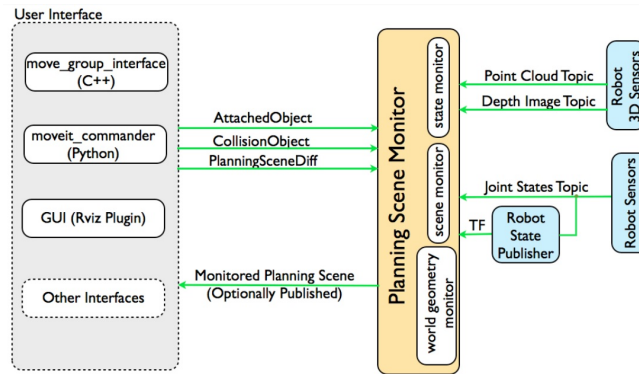


Figure 6.3: Planning scene monitor [19].

Specifically, the planning scene can be built in different ways: it is possible to upload the 3D CAD model of the object or through the 3D perception supported by vision sensors. In this second way, the sensors generate a point cloud or a depth image and then MoveIt converts it into a 3D occupancy grid mapping by using octomap.

### 6.2.1 3D perception

3D perception in MoveIt is handled by the occupancy map monitor. This occupancy map monitor can manage different types of sensor inputs thanks to its plugin architecture. It is possible to add manually any type of updaters for the occupancy map monitor. The two main kinds of inputs are the point clouds and the depth images, which are already built and supported in MoveIt. Finally, these inputs are converted into octomaps thanks to a dedicated ROS node.

## 6.3 Collision checking

Collision checking is the most expensive part during the path generation, which accounts for almost 90% of the planning time. Fortunately, MoveIt is setup in order to do it automatically, so the user never really has to worry about how it is happening.

To speed up the collision checking, it is present the ACM (Allowed Collision Matrix). In this matrix are written all the allowed collisions between two bodies, so the ones in which the check is not needed.

The collision check is done using the FCL (Flexible Collision Library) package. This package supports different types of objects including:

- meshes: the ones added through 3D CAD model.
- primitives shapes: e.g. boxes, cylinders, spheres, cones, and so on.
- octomaps: obtained after conversion from the sensors (e.g. from point clouds or depth images).

# Chapter 7

## Tesseract

Tesseract [34] is another common open-source software in the industrial robotics field used for manipulating robots.

Its operations and architecture are very similar to that of MoveIt, but with some differences present within it. As MoveIt has the MoveIt Setup Assistant to automatically generate all the packages needed to solve problems with the related robot, here there is the Tesseract Setup Wizard. Starting with the URDF model of the robot, the user can set the Allowed Collision Matrix (ACM), kinematics groups, virtual/fixed joints and then generate the SRDF model and other packages needed into the Tesseract pipeline.

In Tesseract there are basically the same features as in MoveIt (motion planning, scene planning, collision checking, and so on), but structured in a different way. The next sections will explain such differences with respect to the MoveIt environment in more detail, analyzing their possible advantages and disadvantages.

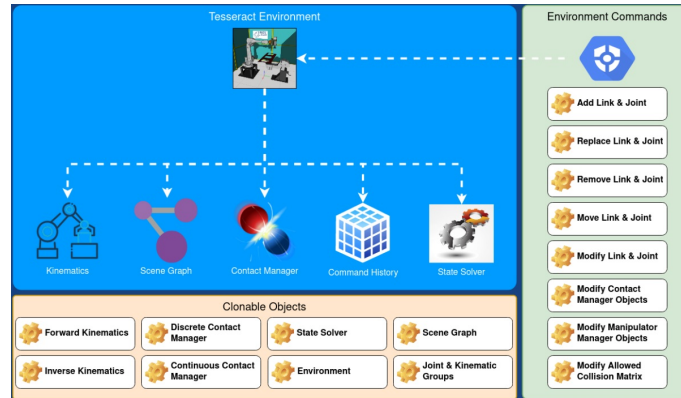


Figure 7.1: Tesseract general system architecture [34].

## 7.1 Motion planning

In the open-source Tesseract framework, it is possible to use all the motion planners belonging to the OMPL library that are available on MoveIt. In addition there are also the Descartes, STOMP and TrajOpt planners. In particular, TrajOpt has several advantages over the simpler sampling-based planners belonging to OMPL. In Tesseract it is possible to cascade motion planners to achieve path post-processing. In this case there is some wasted time due to switching between planners, so the planning time gets longer.

The fact that TrajOpt is already integrated within Tesseract is what prompted the development of this thesis work, so as to compare its performance with popular OMPL planners and understand its merits and shortcomings.

## 7.2 Planning scene

Scene management in Tesseract is more flexible than in MoveIt, this is due to the presence of the Command History (Figure 7.1). This makes possible to modify almost the entire scene, thus adding complexity to the framework. In particular, this eliminates the major constraint present in MoveIt: in this platform everything depends on the initial URDF and SRDF and they are loaded only once at the beginning of the problem. This does not allow subsequent modification in quick and flexible ways. On Tesseract, instead, URDF and SRDF are stored in the Command History and then the scene can be modified more easily (Figure 7.2).

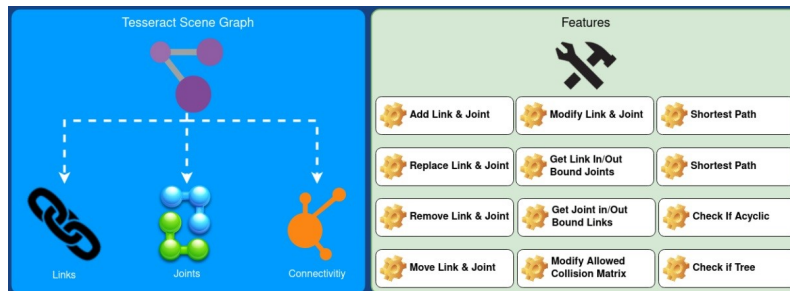


Figure 7.2: Tesseract scene handler [34].

## 7.3 Collision checking

As already explained in the chapters about the MoveIt framework, collision checking is the computationally heaviest and most time-consuming part in the problem. In Tesseract, the FCL library is not used as it is on MoveIt and the collision checking can be done both continuously and discretely thanks to the signed distance and the convex-convex collision detection used (more details are available in subsection 4.3.3). Even here it is possible to speed up the checking using the ACM.

# Chapter 8

## Experimental procedure

The objective of the experimental part developed during this thesis work is the comparison between the performance of the OMPL library on MoveIt currently in use at COMAU and TrajOpt in the Tesseract framework.

To do this each benchmark is tested 40 times: in this way all the data obtained equals the average of the values obtained in the 40 simulations in order to be more reliable.

Initially, the objective of the experimental procedure was to compare the OMPL library planners with TrajOpt in both cases using the Tesseract platform. Immediately it was seen that the OMPL library does not perform satisfactorily on Tesseract, probably because it is less optimized with respect to the one on MoveIt that is used in COMAU. To compare the performances of OMPL on MoveIt and Tesseract, it is used a simple scene shown in Figure 8.1. With this scene ten attempts are launched for both frameworks and the data obtained are shown in Table 8.1.

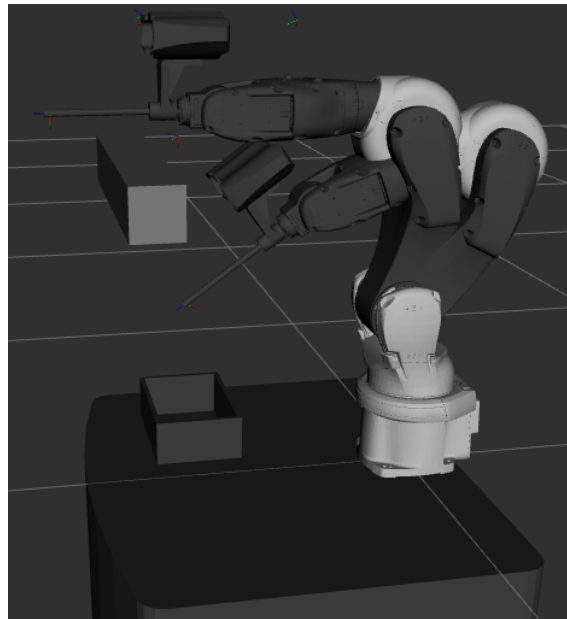


Figure 8.1: Box avoidance to compare OMPL performances on MoveIt and Tesseract. Both the initial and final poses of the Racer5-0-80 are shown.



	OMPL (MoveIt)	OMPL (Tesseract)
Planning time 1 [s]	0.189	6.320
Planning time 2 [s]	0.113	6.467
Planning time 3 [s]	0.087	8.826
Planning time 4 [s]	0.112	7.678
Planning time 5 [s]	0.124	7.128
Planning time 6 [s]	0.244	8.730
Planning time 7 [s]	0.221	7.820
Planning time 8 [s]	0.100	7.787
Planning time 9 [s]	0.078	6.273
Planning time 10 [s]	0.136	7.935

Table 8.1: Box avoidance: OMPL on MoveIt and Tesseract comparisons.

In the Tesseract framework there are significantly longer planning times than in MoveIt and that is why the comparison between TrajOpt on Tesseract and OMPL on MoveIt is chosen. In fact, the main goal is to understand the potential of TrajOpt compared to the current state of the art, so it would not have made sense to compare it with planners that have much worse performance than those currently in use.

The benchmarks are chosen in order to analyse the advantages and disadvantages of TrajOpt compared to OMPL planners on MoveIt. To do this, scenes are created with various features: paths with a single goal state, paths having multiple goal states to be reached sequentially and finally the handling of a gripper change.

For each benchmark the data considered to make the comparison are the success rate and the minimum, average and maximum of both planning time and joint path length. In this way, it is possible to compare not only the planning time required but also the optimality of the final path.

## 8.1 Racer5-0-80

The COMAU Racer5-0-80 robot is used in the benchmarks created for this thesis work [23].

This robot weights about 32 kg and it has 6 working axes. The wrist load capacity is 5 kg and the maximum horizontal is 809 mm; these are the two values from which the robot derives its name. It has an excellent repeatability being only 0.03 mm and it has IP54 certified protection.



Figure 8.2: Robot Racer5-0-80 COMAU [23].

## 8.2 MoveIt pipeline

The pipeline used in the MoveIt framework is divided into 3 main steps. First a ROS launch file called `dexter_ros_startup.launch` is started and then, through the `ros_handler` service, the robot model is decided and whether to use it in simulation or not. This takes the decided robot model (in this case the Racer5-0-80), it initializes the whole MoveIt platform (especially the `move_group` which is the key part) and it also starts the RViz in order to show the robot and the scene around it. In the `ros_handler` service there is also a boolean parameter called `simulation` and a possible `IP_address`. If in simulation is inserted the boolean value 1, it means that all paths are generated and only simulated on the RViz graphical interface. This is what happens in the case of the benchmarks used to compare OMPL with TrajOpt. On the other hand, if in simulation is inserted the value 0, it means that the paths are also executed either by the real robot or by the COMAU Roboshop robot simulator. In this case, it is also necessary to specify the `IP_address` of the robot (it can be easily found in the robot's teach pendant). This is what is done in the case of gripper change management, since for its simulation it is necessary to connect with the robot or Roboshop to obtain the values of the digital outputs (details are available in Chapter 13).

Once the robot model, the MoveIt environment and RViz have been initialized (and, if it is needed, connected to Roboshop or the real robot), the next step is to send the target to be reached so that the planner can look for a collision-free path, if it exists. To carry out this step, there are two different actions: the `move_to_pose` action and the `move_to_joints` action.

In the first case the final pose expressed by the generalized coordinates in the robot space is sent, so the action sends the three cartesian coordinates plus the three angular coordinates in order to uniquely define the final pose. In the second case, the six final values of the robot's joints are sent. In this project the `move_to_pose` action is used, since in these cases it is more practical to work with the final pose in the workspace of the robot. For example, in the Pick & Place problem explained in Chapter 10, it is well defined the final position in which to place the object, so it is very immediate to know the final generalized coordinates of the robot. If the `move_to_joints` action is used, the place position should be converted into joint values, so it presents an additional step. Through these actions, the user can also decide which motion planner to use and whether to execute or just simulate the generated path. Specifically for this thesis, the SBL planner is chosen, as it turns out to be the best-performing planner in the OMPL library for these applications in industrial robotics.

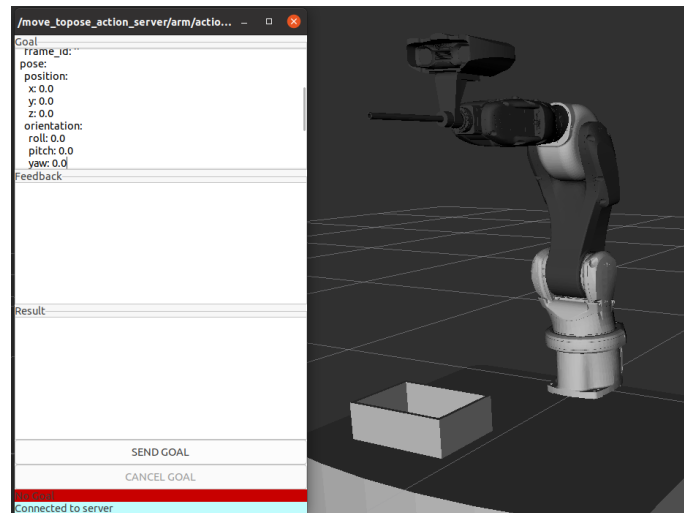


Figure 8.3: MoveIt pipeline: the framework is waiting for the goal state in order to plan the collision-free path.

Once the path is generated and executed, the three parameters that are used for the comparisons are obtained from the terminal:

- execution completed: this is the succeeded/failed response to find the collision-free path from the starting pose to the target pose. After the 40 launches, it is possible to compute the success rate:

$$Success\ rate = \frac{succeeded}{succeeded + failed} \quad (8.1)$$

- time elapsed: this is the time occurred to plan the path. It is computed thanks to a time-watch.
- path length: this is the path length of the joint path. It is computed as the Euclidean length into the joint path space.

$$\Delta\theta = \sqrt{\sum_{i=1}^{N-1} (\theta_{i+1} - \theta_i)^2} \quad (8.2)$$

The value  $\Delta\theta$  is relative to a single joint; it is then necessary to compute the total joint path length as:

$$path\ length = \sum_{j=1}^6 \Delta\theta_j \quad (8.3)$$

### 8.3 Tesseract pipeline

In the Tesseract framework, the pipeline is much more flexible than the one explained in the previous section regarding the MoveIt environment. The first step is always to execute the launch file, in this case it is called `co-mau_picker.launch`. Then the procedure depends on how the code of this launch file is structured. In fact, it is possible to insert directly into the code the objects in the scene, the initial pose and the target pose, the choice of the motion planner to be used, and, in case the TrajOpt planner is used, the value of the safe distance. This is convenient when everything is kept the same and nothing needs to be changed between simulation launches. On the other hand, where more flexibility is required for the environment and the path, two proper services are created. The first one is the `add_link` service, which is very useful for managing the scene around the robot, while the second one is

the `tesseract_planning` service, which is useful to set the start state, the goal state and to choose the motion planner to be used among the possible ones (so OMPL, Descartes, TrajOpt planners and, if desired, a path post-processing). These services are explained in more detail in the following two subsections. Once the path has been generated, as it happens in the MoveIt pipeline, the user can get the data useful for path comparison from the terminal: the succeeded/failed response in order to find the success rate, the planning time through the time-watch and finally the joint path length.

### 8.3.1 Add\_link service

The `add_link` service is a service that is used to manage the scene around the robot as shown in Figure 8.4.

Topic	Type	Expression
/comau/add_link	comau_msgs/AddLinkRequest	0
shape	comau_msgs/Box	0
link_name	string	..
parent_link	string	..
width	float64	0.0
height	float64	0.0
length	float64	0.0
pose	comau_msgs/Pose	0.0
x	float64	0.0
y	float64	0.0
z	float64	0.0
rx	float64	0.0
ry	float64	0.0
rz	float64	0.0
cylinder	comau_msgs/Cylinder	0.0
link_name	string	..
parent_link	string	..
radius	float64	0.0
length	float64	0.0
pose	comau_msgs/Pose	0.0
x	float64	0.0
y	float64	0.0
z	float64	0.0
rx	float64	0.0
ry	float64	0.0
rz	float64	0.0
replace allowed	bool	False

Figure 8.4: Add\_link service structure.

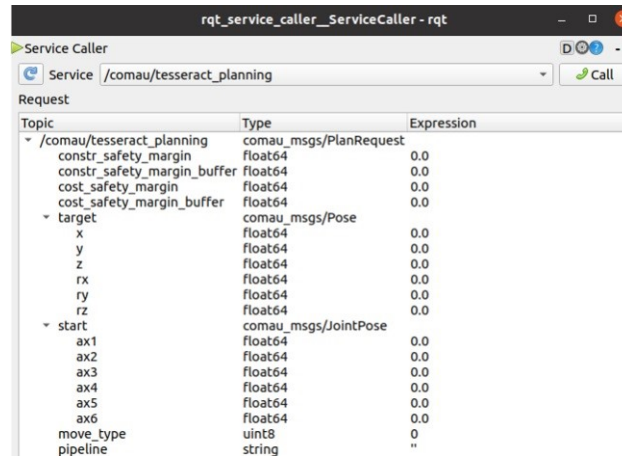
This service allows four types of objects to be added to the scene:

- **box:** it adds a box to the scene by specifying the dimensions of the three sides (width, height, length).
- **cylinder:** it adds a cylinder to the scene by specifying its radius and length.
- **sphere:** it adds a sphere to the scene by specifying its radius.
- **mesh:** it adds a mesh to the scene by specifying the path in which the desired CAD model is located. In addition to the object mesh, the collision mesh must also be entered: this is usually a simpler mesh in which the actual model is enlarged. In this way the computations for collision checking are faster and they are safely approximated by enlarging the dimensions of the real object.

For each object, in addition to the needed dimensions already explained, the pose in space must be inserted: it is necessary to specify which is the reference system and, with respect to it, the values of the generalized coordinates.

### 8.3.2 Tesseract\_planning service

The tesseract\_planning service is needed when the user needs more flexibility in the choice of start pose, target pose and motion planner.



Topic	Type	Expression
comau/tesseract_planning	comau_msgs/PlanRequest	
constr_safety_margin	float64	0.0
constr_safety_margin_buffer	float64	0.0
cost_safety_margin	float64	0.0
cost_safety_margin_buffer	float64	0.0
target	comau_msgs/Pose	
x	float64	0.0
y	float64	0.0
z	float64	0.0
rx	float64	0.0
ry	float64	0.0
rz	float64	0.0
start	comau_msgs/JointPose	
ax1	float64	0.0
ax2	float64	0.0
ax3	float64	0.0
ax4	float64	0.0
ax5	float64	0.0
ax6	float64	0.0
move_type	uint8	0
pipeline	string	"

Figure 8.5: Tesseract\_planning service structure.

For the start state and the goal state it is required to put the six values needed to define them completely:

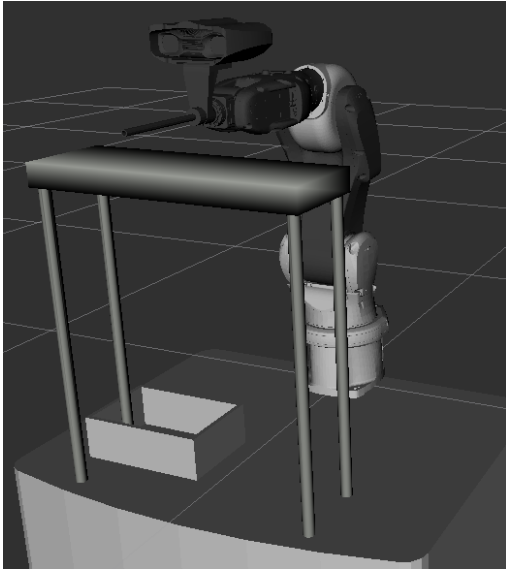
- start state: in this case the values of the six joints are entered. This choice is made in order to simplify the entry of the initial pose being always (0 0 -1.57 0 0 0) radians in joint space, while it is more complex in cartesian space.
- target state: in the goal state instead the six values of generalized coordinates are inserted since it is easier to obtain the goal state into the robot space.

For the choice of the motion planner (called pipeline in Figure 8.5), the user needs to just enter the name of the desired one. In case the OMPL is entered, all planners belonging to the library are launched in parallel and then the final result is taken from the best resulting one.

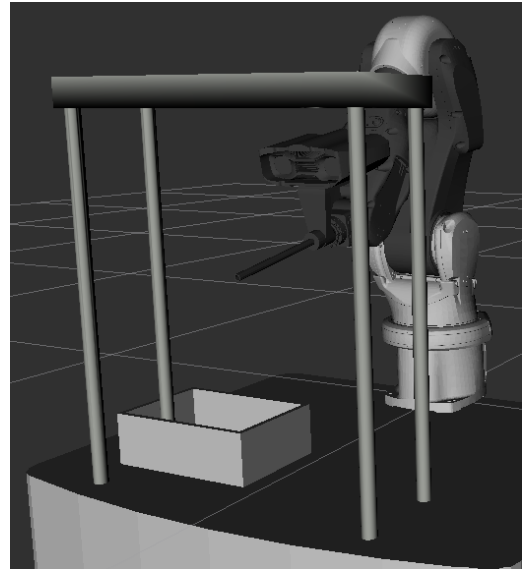
As anticipated at the beginning of the chapter, in this thesis only TrajOpt is used as the motion planner in the Tesseract planning framework. This is because the OMPL library has poor results compared to the same library in the MoveIt environment.

# Chapter 9

## Benchmark 1: table avoidance



(a) Racer5-0-80 initial pose over the table.



(b) Racer5-0-80 final pose under the table.

Figure 9.1: Start and goal states for the table avoidance.

In this first benchmark, the task required to the robot is to move from above to below the table while avoiding collisions with it (Figure 9.1).

In this test, both `add_link` and `tesseract_planning` services in the Tesseract framework are exploited. More in detail, to add the table to the scene, both the CAD model of the object and a more simplified collision CAD are inserted to compute its collisions. In this way the part concerning collision checking is sped up.

## 9.1 Results

	OMPL (MoveIt)	TrajOpt (Tesseract)
Success Rate [%]	100	100
Minimum Planning Time [s]	0.111	21.612
Average Planning Time [s]	0.378	22.090
Maximum Planning Time [s]	1.953	22.576
Minimum Path Length [m]	2.886	2.216
Average Path Length [m]	4.928	2.241
Maximum Path Length [m]	8.554	2.244

Table 9.1: Table avoidance: OMPL and TrajOpt comparisons.

As shown in Table 9.1, in both frameworks the success rate is 100%, i.e., no failures in the 40 launches performed.

It is important to note that in the case of the TrajOpt planner the times are substantially longer; this is due to the table mesh, which increases the load during the research and optimization of the collision-free path. On the other hand it can be seen, as expected, that the joint path is optimized and so it is shorter with respect to the one obtained with OMPL. Another consideration that can be made by looking at the table is the variability of the data obtained. Indeed, in the case of TrajOpt the path found is practically always the same, with joint path length and planning time very similar. In the case of OMPL on MoveIt these data change a lot, since the path is generated randomly and it is not optimized.

In conclusion TrajOpt finds much better paths, however with much longer times. This makes it severely limited in uses like this where there is only one target point and with a quite complex scene.



# Chapter 10

## Benchmark 2: Pick & Place

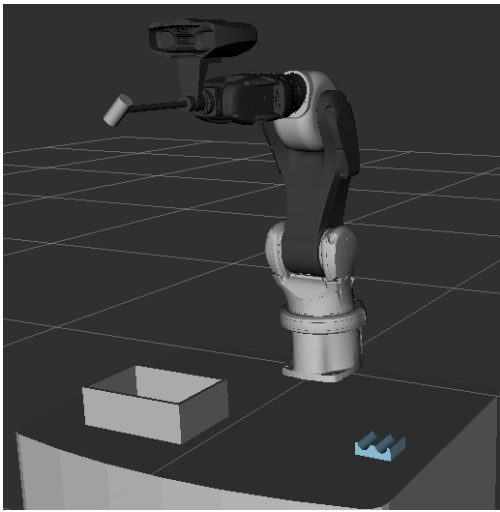


Figure 10.1: Initial pose.

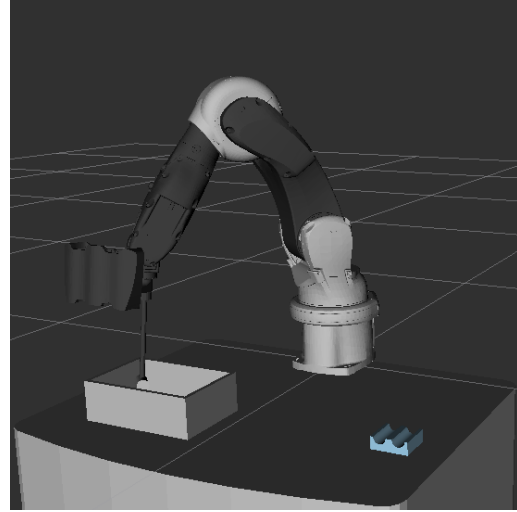


Figure 10.2: Pick pose.

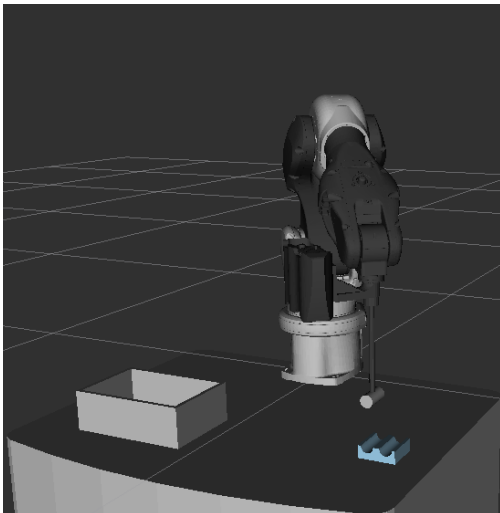


Figure 10.3: Pose 0.01m above the final one.

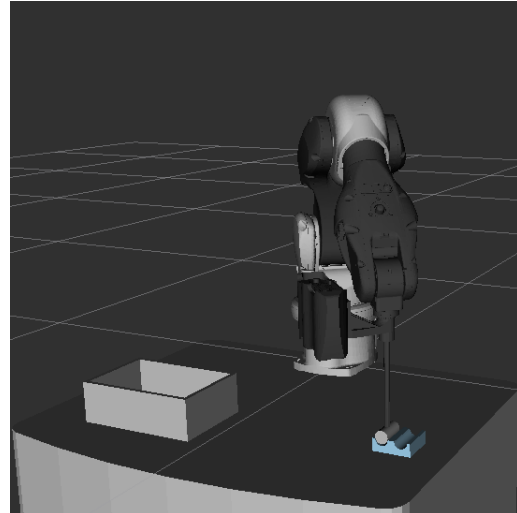


Figure 10.4: Final pose or Place pose.

Figure 10.5: Four main Racer5-0-80 poses for the Pick & Place.

In this benchmark there is a typical Pick & Place scene. In this case the Racer5-0-80 starts from the initial pose reported in Figure 10.1. Then it goes and it picks up a cylindrical object in the box to the left of the figure at the position defined as "Pick" (Figure 10.2), next it goes to the position 0.01m above the final position of the place (Figure 10.3). Finally, it descends vertically and it goes to place the object in the final support defined as "Place" position (Figure 10.4). Pick & Place is a real and very typical scene in the world of industrial robotics, this is why this scene is chosen for the comparison between TrajOpt on Tesseract and OMPL on MoveIt. To take full advantage of the potential of TrajOpt, it is decided to send the three target points directly from code. In this way, the planner reaches them sequentially generating a collision-free path.

For simplicity, the cylinder to be picked up in the box and placed in the final support is connected to the robot's gripper at the beginning of the scene. In reality, this is not present initially and it is taken into the box, so it is only present after reaching the target in Figure 10.2.

## 10.1 Results

	OMPL (MoveIt)	TrajOpt (Tesseract)
Success Rate [%]	100	100
Minimum Planning Time [s]	0.112	0.923
Average Planning Time [s]	0.274	0.975
Maximum Planning Time [s]	0.409	1.038
Minimum Path Length [m]	4.437	4.250
Average Path Length [m]	4.895	4.250
Maximum Path Length [m]	6.381	4.250

Table 10.1: Pick&Place: OMPL and TrajOpt comparisons.

As in the first benchmark, the success rates are 100% in both cases confirming how effective both planners are in these types of problems.

In this benchmark the planning times using TrajOpt are highly competitive: they are roughly four times longer than OMPL ones, however they have the huge advantage of being computed once. In fact, in the case of TrajOpt the user executes the code containing all the targets in the scene and the planner reaches them sequentially, resulting in the time that one reads in Table 10.1 with a single launch. In the case of OMPL on MoveIt, that time is equivalent

to the sum of the generation of three separate paths. The first to reach the Pick pose, the second to stand over the Place pose and finally the third to go into the final pose. After all, the sum of these three planning times is shorter than the planning time using TrajOpt, but it has to be considered the need to perform a computation for each target, since it is not possible to reach them sequentially as in the previous case.

Moreover, even in this case the joint path turns out to be optimized and therefore shorter and constant, instead in the OMPL case it turns out to be longer and variable.

In conclusion, it can be seen that TrajOpt in this type of scenes provides numerous advantages: the collision-free path turns out to be effective and optimized with only one planning for all the desired target points in the scene to be reached.

# Chapter 11

## Benchmark 3: Pick & Place with obstacle avoidance

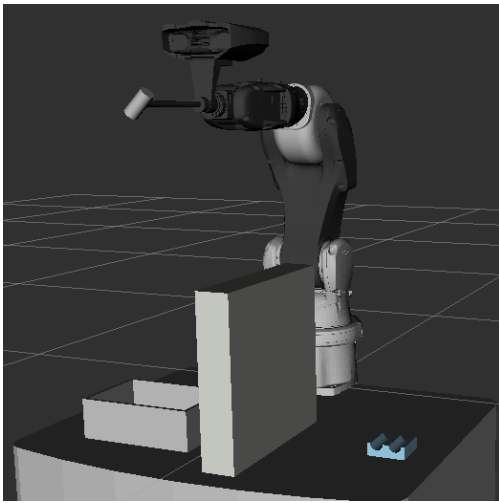


Figure 11.1: Initial pose.

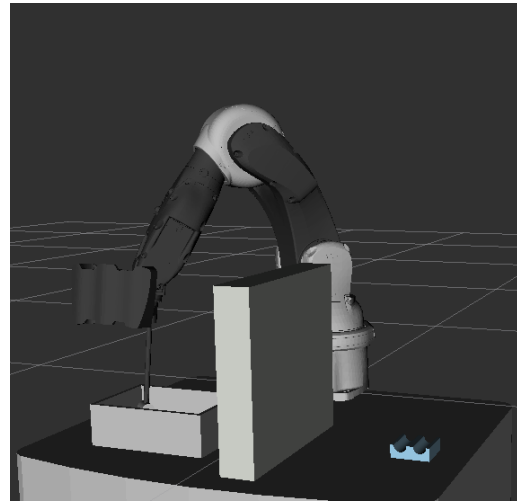


Figure 11.2: Pick pose.

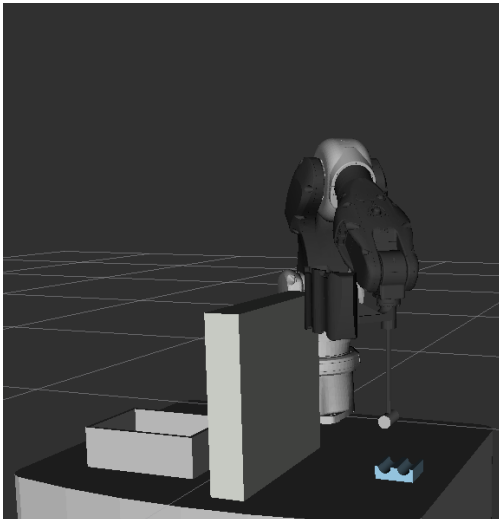


Figure 11.3: Pose 0.01m above the final one.

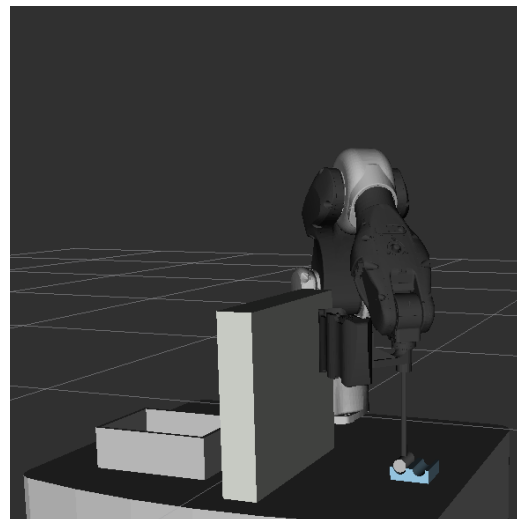


Figure 11.4: Final pose or Place pose.

Figure 11.5: Four main Racer5-0-80 poses for the Pick & Place with obstacle avoidance.

## CHAPTER 11. BENCHMARK 3: PICK & PLACE WITH OBSTACLE AVOIDANCE

Here it is used the same Pick & Place scene as in the previous benchmark, but with a "wall" obstacle to avoid in the middle between the Pick pose and the Place pose. This is also a common scene in industrial robotics, so it is interesting to analyse the behaviour in both cases for further comparisons. For convenience, the collision object in the middle is added from the code and not with the `add_link` service. This is because the target points are inserted from the code in order to reach them in only one launch, so it is more convenient to insert also the scene from the code rather than via service.

### 11.1 Results

	OMPL (MoveIt)	TrajOpt (Tesseract)
Success Rate [%]	100	100
Minimum Planning Time [s]	0.261	2.926
Average Planning Time [s]	0.499	3.033
Maximum Planning Time [s]	0.923	3.214
Minimum Path Length [m]	4.721	4.595
Average Path Length [m]	5.278	4.595
Maximum Path Length [m]	5.889	4.595

Table 11.1: Pick&Place with obstacle avoidance: OMPL and TrajOpt comparisons.

The considerations that can arise from this benchmark are very similar to the previous ones.

Again both solutions prove the effectiveness of the planners, with success rates of 100%. The planning times using TrajOpt are proportionally longer than those using OMPL. While the former are more or less tripled, the latter are doubled. This is mainly due to the fact that the scene is more complex than the previous one and, as already mentioned, the TrajOpt performance is highly dependent on the complexity of the problem. The joint path lengths always turn out to be fixed and optimized. Again there is the advantage of using TrajOpt in the Tesseract platform to launch once to reach all three target points sequentially.

# Chapter 12

## Benchmark 4: TrajOpt safe distance

In this last developed benchmark, the same scene as Pick & Place with obstacle avoidance is tested. Here instead of focusing on an OMPL-TrajOpt comparison, it is analyzed how the safe distance on TrajOpt affects the final collision-free path.

As already explained in subsection 4.3.3, there are two main parameters regarding the distances in the TrajOpt planner. The first is the safe distance  $d_{safe}$ , i.e., the minimum guaranteed distance to each object in the scene. The second is the check distance  $d_{check}$ , greater than the previous one, from which the collision checking is activated in order to guarantee the safe distance constraint definitely. Thanks to this procedure, the algorithm does not waste time in computations where the objects are for sure safe, since in these situations the distance is such that  $d > d_{check} > d_{safe}$ .

Higher safe distances mean longer planning times since the constraints are more strict. In other words, the larger these values are, the more poses the robot cannot reach to guarantee the safe distance constraint. However, higher safe distances also correspond to greater safety as the risk of collision with objects in the scene is reduced. In conclusion, the user must be careful not to exceed it on both sides, finding the proper trade-off between constraint stiffness and safety.

## 12.1 Results

	TrajOpt (Tesseract)	TrajOpt (Tesseract)
Safe Distance [m]	0.01	0.001
Success Rate [%]	100	100
Minimum Planning Time [s]	2.926	1.903
Average Planning Time [s]	3.033	2.048
Maximum Planning Time [s]	3.214	2.282
Minimum Path Length [m]	4.595	4.490
Average Path Length [m]	4.595	4.490
Maximum Path Length [m]	4.595	4.490

Table 12.1: TrajOpt safe distances comparison.

As can be seen, in Table 12.1 a comparison is made between a safe distance of 0.01m and 0.001m. The check distance is twice the safe distance at all times. As expected, with a smaller safe distance (and hence also a smaller check distance) the performance is definitely better both as time planning required and as joint path length. Obviously this also means less safety, as it can also be seen in Figure 12.1, in which with a safe distance of 0.001m the robot's arm passes significantly closer to the obstacle, with has higher probability to collide with the object.

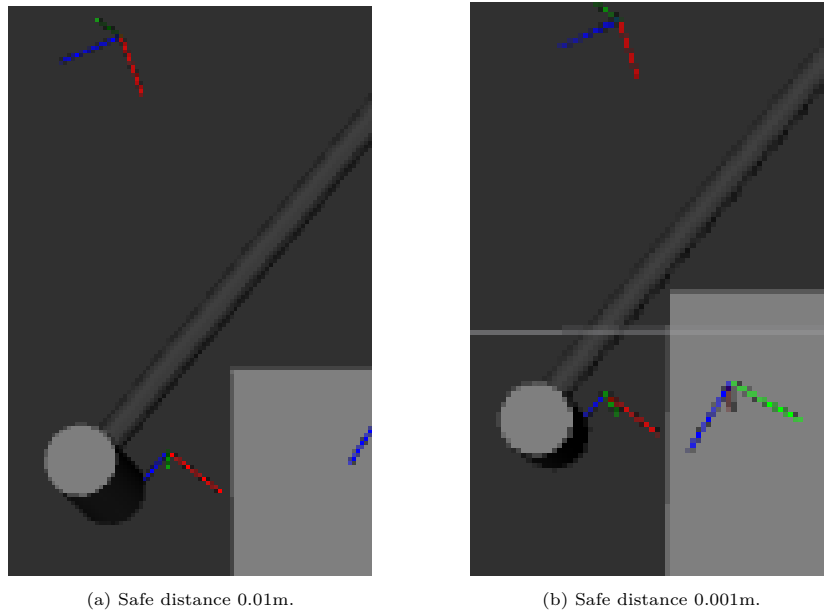


Figure 12.1: Safe distances comparison.

# Chapter 13

## Gripper change management

The gripper change management is a typical problem in industrial robotics where great flexibility in scene management is required. In this thesis work, a ROS node is developed for gripper change management in an NJ-370-2.7 robot on the MoveIt framework.

### 13.1 NJ-370-2.7

In the last work developed in this thesis, the gripper change management, the robot used is no longer the Racer5-0-80 explained in Section 8.1, but it is the NJ-370-2.7 [20]. This robot is used for gripper change management because it already has an industrial application with 3 grippers, specifically in COMAU's Flex-BD project.

This robot has a weight of about 2100 kg and it has 6 working axes. The wrist load capacity is 370 kg and the maximum horizontal is 2703 mm; these are the two values from which the robot derives its name. It also supports an additional forearm load of 50 kg. Being a significantly larger robot than the Racer5-0-80 it has a worse repeatability, more precisely 0.15 mm, and it has IP65/IP67 certified protection.





Figure 13.1: Robot NJ-370-2.7 COMAU [20].

## 13.2 Gripper change management pipeline

The robot considered, NJ-370-2.7 [20], for the required application has three different grippers: each of them has a specific task and, when required, a gripper change is needed to be handled in simulation as well.

In the MoveIt environment, all the objects in the scene must be initially loaded into the URDF of the robot. This forces the user to manage the gripper change in a rough way: all three grippers are loaded into the scene at the very beginning and placed in a position far away from the robot, thus they are not seen in the simulation environment. When the robot uses a gripper, in the simulation the position of that gripper is changed in order to attach it to the flange of the robot resulting in a working position. In this way, only the proper gripper is shown in the scene attached to the robot when it is used, while the others remain far away so it is not possible to see them in the actual scene.

To do this, the created node is subscribed by the topic `joint_states` and the digital outputs of the robot, and it goes to publish on a new topic called `joint_states_arm`. More in detail, the topic `joint_states` contains the values of the six joints of the robot. The digital outputs of the robot are boolean values that in this case are 0 if the gripper is not present, and 1 otherwise. So, since there are three possible grippers, if the first gripper is present, the first digital output has the value 1 while the others are 0; if the second gripper is present, the second digital output has the value 1 while the others are 0; if the third gripper is present, the third digital output has the value 1 while the others are 0; otherwise, all the digital outputs values are 0. Finally, the topic `joint_states_arm` consists of nine values: the six values of the robot joints taken from the topic `joint_states` and the three values of the positions of the three grippers with respect to the flange of the robot. These three values are huge if the gripper is not present (in this way the gripper results far away, so it is not possible to see it in the scene), while zero if the gripper is used and so it is attached to the flange of the robot. During the execution of the gripper change management node, it is implemented a check that there are not two or more digital outputs set to 1, otherwise, it would mean that there is more than one active gripper on the robot and this is an evident error.

As it can be guessed, in the MoveIt framework gripper change management is not trivial and in any case it is not even accurate, since the user has to load all three grippers into the scene even though this is not what happens in reality. On Tesseract this operation is much quicker and more flexible. Indeed in this framework it is possible to load only the URDF of the robot at the beginning and then, when requested, it can be added and removed a gripper from the scene. This way of proceeding is cleaner and more realistic, as inactive grippers are not present into the scene. Practical issues are also present on Tesseract, in fact in the case of complex grippers CAD models, the final planning time is much higher as the problem becomes much heavier.

## CHAPTER 13. GRIPPER CHANGE MANAGEMENT

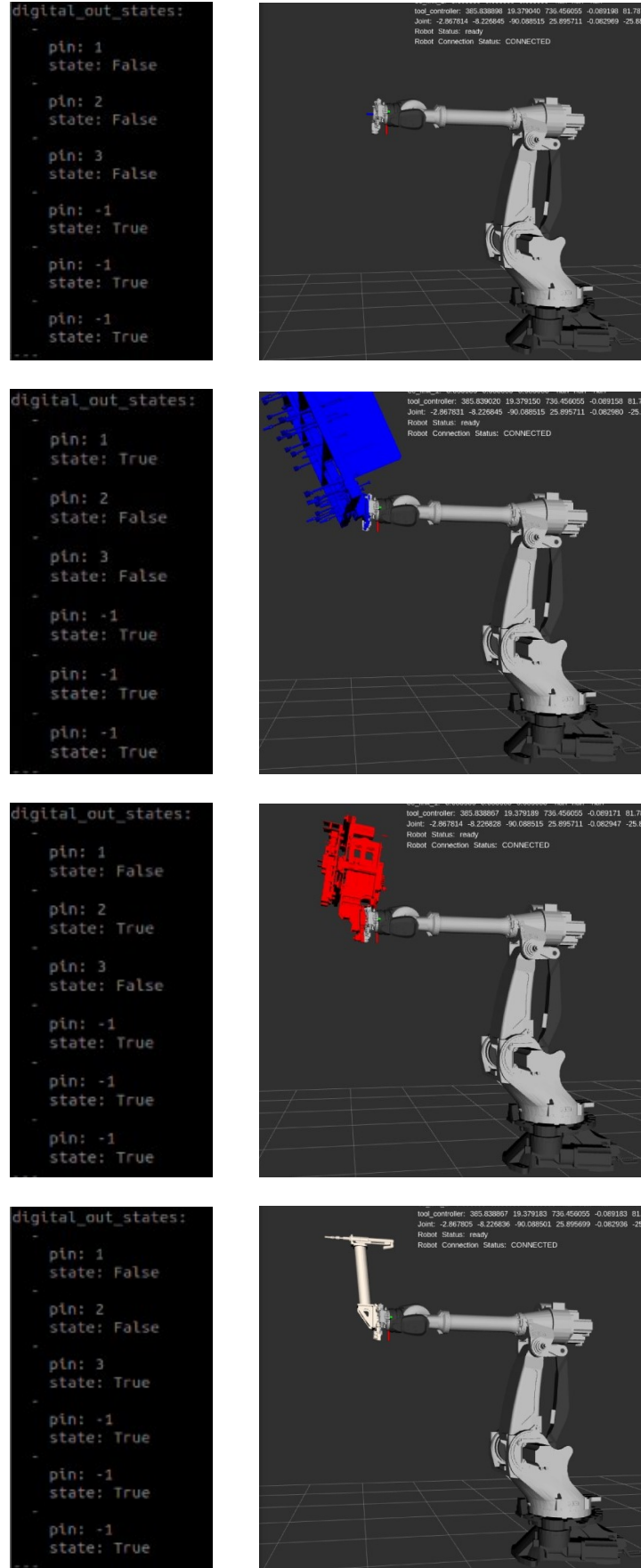


Figure 13.2: Robot digital outputs and relative gripper change management.

# Chapter 14

## Conclusions

Thanks to the scenes created and the several tests performed, very useful considerations can be drawn.

First of all, it can be said that TrajOpt is generally slower than the sampling-based planners, even if with optimized collision-free paths and therefore shorter and smoother ones. This does not make it applicable in cases where online path planning is used, because it is too slow, but it is suitable for situations in which it is necessary to have an optimal and precise path.

Moreover, it is important to note that TrajOpt, being an optimized-based planner, obtains an optimal path and therefore, launching the same problem several times, the collision-free path generated is practically always the same. This is not the case of the planners in the OMPL library; indeed, they generate the collision-free path by random sampling the configuration space, so they are not deterministic, and the final paths have a higher variability.

Other important factors to be considered in the analysis of the performance of TrajOpt are the complexity of the scene, its management, and the number of target goals required. The performance of TrajOpt turns out to be strictly correlated to the complexity of the scene. In the table avoidance benchmark with a heavy mesh (Chapter 9), the TrajOpt planning algorithm is slowed down a lot. On the other hand, however, TrajOpt in the Tesseract framework allows a much more flexible scene management, thus making it usable in cases like gripper change management (Chapter 13). In problems such as Pick & Place, where there are multiple targets to be reached (Chapters 10 and 11), the ability to reach them sequentially with a single launch results in a considerable advantage over planners belonging to the OMPL library on MoveIt.

Finally, it arises that the safe distance parameter is very important to be tuned properly. Indeed, with lower safe distances the TrajOpt performances increase a lot, but the probability of collisions is much higher. In this sense, a suitable trade-off must be found for each situation (Chapter 12).

# Chapter 15

## Future works

Thanks to this thesis work, good progress has been made in the analysis of the performance of the TrajOpt planner in typical industrial robotics problems. Obviously, there are still many improvements that can be made to enhance this analysis.

First, it would be very useful to implement a hardware interface between the Tesseract framework and the COMAU robot. This interface is already created for the MoveIt environment and allows the robot to execute the generated collision-free paths. This would make it possible to compare not only the success rates, planning times and joint path lengths, but also the execution times of the path and the energy consumption of the robot.

It would be also interesting to expand the comparison to other planners. Now the comparison is between the TrajOpt planner and the sampling-based in the OMPL library. It could be added the performance of some innovative post-processing planners, as it is done in paper [5], where the authors focus on the deterministic Chekhov+TrajOpt planner.

Finally, it is possible to test a couple of potentially very useful properties present in Tesseract, such as collision checks performed on the continuous paths, instead on the discrete points (exploiting the concept of swept-out volume reported in subsection 4.3.3) and the dynamic scene management. For this last point, it would be useful to implement also a ROS package that acquires the scene through a laser scanner.

# Chapter 16

## Acknowledgements

Questo elaborato sancisce la fine del percorso di laurea magistrale e, complessivamente, del mio percorso accademico. Per questo mi sembra doveroso ringraziare chi c'è stato.

In primis un grazie va alla mia relatrice Prof. Marina Indri e ai miei tutor aziendali Ing. Simone Panicucci e Ing. Antonio Venezia per avermi seguito ed indirizzato durante il lavoro di tesi. Ringrazio anche la COMAU e tutti i colleghi conosciuti durante questo periodo di tesi in azienda per ogni momento condiviso.

Un enorme grazie va a tutta la mia famiglia per non avermi mai fatto mancare nulla ed avermi sempre supportato nei miei piani. In particolare ai miei genitori Eva e Stefano, a mia sorella Sara, ai nonni, allo zio Loris e agli zii Federico e Lina.

Un grazie di cuore ai miei amici del Friuli e di Torino che mi son stati vicino in questi due anni piuttosto difficili per me. Una menzione speciale è per Enrico, Mirko, Giada e Davide che son stati delle spalle importanti su cui poggiarmi per raggiungere questo traguardo.

Da sempre, ogni volta che si parlava di questo momento, al papà ed ai nonni si illuminavano gli occhi di gioia e iniziavano a raccontare della festa grande che si sarebbe fatta. Ora non siete qui, ma son sicuro che mi state vedendo e state festeggiando con noi questo traguardo che vi stava così a cuore.

# Chapter 17

## Acronyms

ACM - Allowed Collision Matrix

APSP - All-Pairs Shortest Path

CAD - Computer Aided Design

CHOMP - Covariant Hamiltonian Optimization Motion Planner

DOF - Degree Of Freedom

FCL - Flexible Collision Library

LCQP - Linear Constraint Quadratic Programme

OMPL - Open Motion Planning Library

NN - Nearest Neighbor

PRM - Probabilistic Roadmap Method

QCP - Quality Control Plan

ROS - Robot Operating System

RRT - Rapidly-Random Tree

SBL - Single-query Bi-directional probabilistic roadmap planner with Lazy collision checking

SQP - Sequential Quadratic Program

SRDF - Semantic Robot Description Format

STOMP - Stochastic Trajectory Optimization Motion Planner

TRM - Tube-based RoadMap

URDF - Unified Robot Description Format

# List of Figures

1.1	COMAU industrial robots working in a production line [35]. . . . .	5
2.1	Revolute joint (left) and prismatic joint (right) [25]. . . . .	8
2.2	Sketch of an industrial robot with 6DOF and a spherical wrist [25]. . . . .	8
2.3	Position and orientation of a rigid body [33]. . . . .	9
2.4	Position and orientation of the end-effector frame with respect to the base frame [33]. . . . .	11
2.5	Denavit-Hartenberg convention [33]. . . . .	12
2.6	Two admissible solutions for the inverse kinematics problem for a two-link planar arm [33]. . . . .	13
3.1	Configuration space of a two-joint manipulator: on the left a representation topologically correct as a torus 2D, on the right a representation locally valid as subset $R^2$ [26]. . . . .	17
3.2	The right figure shows the configuration space corresponding to the workspace on the left. A two-link robot is used and each obstacle has a different colour to simplify the mapping [22]. . . . .	18
4.1	Comparison of depth-first (left) and breadth-first (right) planners. The number near the nodes is the order in the expansion of the tree [26]. . . . .	21
4.2	General structure of a sampling-based planner: the inputs are the initial state, the final state and the configuration space. Thanks to the sample procedure and the collision checks, it looks for a collision-free path. [8]. . . . .	23
4.3	Planning the collision-free path through RRT algorithm [21]. . . . .	25
4.4	Two trees development in the bi-directional RRT algorithm [36]. . . . .	26
4.5	Collision-free path planning using PRM planner [21]. . . . .	28
4.6	Noisy trajectories around an obstacle starting from an arbitrary initial one (in red) [7]. . . . .	33
4.7	Final optimized collision-free trajectory (in blue) starting an arbitrary initial one (in red) [7]. . . . .	34



## LIST OF FIGURES

---

4.8	Concepts of signed distance and swept-out volume between two objects A and B [31]. . . . .	36
4.9	Structure of a post-processing planner: cascade of two motion planners (left) [16]; final optimized-trajectory starting from collision-free inputs (right) [18]. . . . .	39
5.1	ROS topic structure: communication between a publisher node and a subscriber node [30]. . . . .	42
5.2	ROS service structure: communication between a service client node and a service server node [29]. . . . .	43
5.3	ROS action structure: communication between an action client node and an action server node [28]. . . . .	44
5.4	ROS RViz graphical interface. . . . .	44
5.5	Racer3-0-63 Roboshop graphical interface. . . . .	45
6.1	MoveIt general system architecture [19]. . . . .	47
6.2	MoveIt move_group architecture [19]. . . . .	47
6.3	Planning scene monitor [19]. . . . .	49
7.1	Tesseract general system architecture [34]. . . . .	51
7.2	Tesseract scene handler [34]. . . . .	52
8.1	Box avoidance to compare OMPL performances on MoveIt and Tesseract. Both the initial and final poses of the Racer5-0-80 are shown. . . . .	54
8.2	Robot Racer5-0-80 COMAU [23]. . . . .	56
8.3	MoveIt pipeline: the framework is waiting for the goal state in order to plan the collision-free path. . . . .	57
8.4	Add_link service structure. . . . .	59
8.5	Tesseract_planning service structure. . . . .	60
9.1	Start and goal states for the table avoidance. . . . .	61
10.1	Initial pose. . . . .	63
10.2	Pick pose. . . . .	63
10.3	Pose 0.01m above the final one. . . . .	63
10.4	Final pose or Place pose. . . . .	63
10.5	Four main Racer5-0-80 poses for the Pick & Place. . . . .	63

## LIST OF FIGURES

---

11.1	Initial pose. . . . .	66
11.2	Pick pose. . . . .	66
11.3	Pose 0.01m above the final one. . . . .	66
11.4	Final pose or Place pose. . . . .	66
11.5	Four main Racer5-0-80 poses for the Pick & Place with obstacle avoidance. . . . .	66
12.1	Safe distances comparison. . . . .	69
13.1	Robot NJ-370-2.7 COMAU [20]. . . . .	71
13.2	Robot digital outputs and relative gripper change management.	73

# List of Tables

8.1	Box avoidance: OMPL on MoveIt and Tesseract comparisons.	55
9.1	Table avoidance: OMPL and TrajOpt comparisons. . . . .	62
10.1	Pick&Place: OMPL and TrajOpt comparisons. . . . .	64
11.1	Pick&Place with obstacle avoidance: OMPL and TrajOpt comparisons. . . . .	67
12.1	TrajOpt safe distances comparison. . . . .	69

# Bibliography

- [1] R. Bohlin and L.E Kavraki. “Path Planning Using Lazy PRM”. In: *IEEE, International Conference on Robotics and Automation* (2000) (cit. on p. 29).
- [2] H. Choset and et al. “*Probabilistic Roadmap Path Planning*”. URL: <http://www.cs.columbia.edu/~allen/F15/NOTES/Probabilisticpath.pdf> (cit. on pp. 23, 26).
- [3] H. Choset and et al. “*Robotic Motion Planning: Sample-Based Motion Planning*”. URL: [https://www.cs.cmu.edu/~motionplanning/lecture/Chap7-Prob-Planning\\_howie.pdf](https://www.cs.cmu.edu/~motionplanning/lecture/Chap7-Prob-Planning_howie.pdf) (cit. on pp. 22, 23, 25, 26).
- [4] S. Dai and et al. “Fast-reactive probabilistic motion planning for high-dimensional robots”. In: *SAGE* (2019) (cit. on p. 30).
- [5] S. Dai and et al. “Improving Trajectory Optimization Using a Roadmap Framework”. In: *MIT Open Access Articles* (2018) (cit. on pp. 34, 75).
- [6] J. Ding and et al. “An improved RRT\* algorithm for robot path planning based on path expansion heuristic sampling”. In: *Elsevier* (2023) (cit. on pp. 24, 25, 28).
- [7] M. Dobis and et al. “Cartesian Constrained Stochastic Trajectory Optimization for Motion Planning”. In: *Applied Sciences, MDPI* (2021) (cit. on pp. 33, 34).
- [8] M. Elbanhawi and M. Simic. “Sampling-Based Robot Motion Planning: A Review”. In: *IEEE Access, practical innovations* (2014) (cit. on pp. 19, 22, 23, 26, 28).
- [9] A. Hofmann and et al. “Reactive Integrated Motion Planning and Execution”. In: *MIT Open Access Articles, AAAI Press* (2015) (cit. on p. 30).
- [10] M. Kalakrishnan and et al. “STOMP: Stochastic Trajectory Optimization for Motion Planning”. In: *IEEE International Conference on Robotics and Automation* (2011) (cit. on p. 33).
- [11] S. Karaman and et al. “Anytime Motion Planning using the RRT\*”. In: *MIT Open Access Articles* (2011) (cit. on p. 29).

- [12] S. Karaman and E. Frazzoli. “Sampling-based algorithms for optimal motion planning”. In: *ijrr, The International Journal of Robotics Research* (2011) (cit. on pp. 23, 26, 28).
- [13] L.E. Kavraki and et al. “Probabilistic Roadmaps for Path Planning in High-Dimensional Configurational Spaces”. In: *IEEE, Transactions on Robotics and Automation* (1996) (cit. on p. 26).
- [14] J. Kuffner and S. LaValle. “RRT-connect: An Efficient Approach to Single-Query Path Planning”. In: *IEEE, International Conference on Robotics and Automation* (2000) (cit. on p. 25).
- [15] X. Li and et al. “Smooth and collision-free trajectory generation in cluttered environments using cubic B-spline form”. In: *Elsevier* (2022) (cit. on p. 38).
- [16] S. Liu and et al. “Benchmarking and optimization of robot motion planning with motion planning pipeline”. In: *The International Journal of Advanced Manufacturing Technology* (2022) (cit. on pp. 32, 34, 39).
- [17] Y. Liu and et al. “Creating Better Collision-Free Trajectory for Robot Motion Planning by Linearly Constrained Quadratic Programming”. In: *frontiers in Neurorobotics* (2021) (cit. on p. 37).
- [18] R. Luna and et al. “Anytime Solution Optimization for Sampling-Based Motion Planning”. In: *IEEE International Conference on Robotics and Automation* (2013) (cit. on pp. 19, 39).
- [19] *MoveIt!* <https://moveit.ros.org/> (cit. on pp. 46, 47, 49).
- [20] *NJ-370-2.7 COMAU*. <https://www.comau.com/it/competencies/robotics-automation/robot-team/nj-370-2-7/> (cit. on pp. 70, 71).
- [21] “Open Motion Planning Library: A Primer”. In: *Kavraki Lab, Rice University* (2021) (cit. on pp. 23, 25, 26, 28).
- [22] J. Pan and D. Manocha. “Efficient Configuration Space Construction and Optimization for Motion Planning”. In: *Engineering, Volume 1* (2015) (cit. on pp. 17, 18).
- [23] *Racer5-0-80 COMAU*. <https://www.comau.com/it/competencies/robotics-automation/robot-team/racer-5-0-80/> (cit. on pp. 55, 56).
- [24] N. Ratliff and et al. “CHOMP: Gradient Optimization Techniques for Efficient Motion Planning”. In: *IEEE International Conference on Robotics and Automation* (2009) (cit. on pp. 19, 32).

- [25] A. Rizzo. “Kinematic chains”. In: *Robotics course, Mechatronics Engineering, Politecnico di Torino* (2021/2022) (cit. on p. 8).
- [26] C. Ronald. *"Principles of Robot Motion"*. The MIT Press, Cambridge, Massachusetts, 2005 (cit. on pp. 15–17, 20–23, 26, 29).
- [27] ROS. <https://www.ros.org/> (cit. on p. 40).
- [28] ROS Action. <https://foxglove.dev/blog/creating-ros1-actions> (cit. on pp. 43, 44).
- [29] ROS Service. <https://automaticaddison.com/how-to-create-a-service-in-ros-noetic/> (cit. on pp. 42, 43).
- [30] ROS Topic. <https://automaticaddison.com/create-a-hello-world-project-in-ros/> (cit. on p. 42).
- [31] J. Schulman and et al. “Finding Locally Optimal, Collision-Free Trajectories with Sequential Convex Optimization”. In: *The International Journal of Advanced Manufacturing Technology* (2013) (cit. on pp. 34, 36).
- [32] J. Schulman and et al. “Motion planning with sequential convex optimization and convex collision checking”. In: *UC Berkeley and the International Journal of Robotics Research*, 33(9) (2014) (cit. on p. 34).
- [33] B. Siciliano and et al. *"Robotics: Modelling, Planning and Control"*. Springer, 2009 (cit. on pp. 9, 11–13, 15–17, 22, 26).
- [34] Tesseract. [https://tesseract-docs.readthedocs.io/\\_/downloads/en/latest/pdf/](https://tesseract-docs.readthedocs.io/_/downloads/en/latest/pdf/) (cit. on pp. 51, 52).
- [35] F. Villon. *"I robot Comau quali acceleratori di competitività presso Modula"*. URL: <https://www.tecnelab.it/approfondimenti/speciali/i-robot-comau-quali-acceleratori-di-competitivita-presso-modula> (cit. on p. 5).
- [36] B. Wang, J. Li, and M. Meng. “Kinematic Constrained Bi-directional RRT with Efficient Branch Pruning for robot path planning”. In: *Elsevier* (2021) (cit. on p. 26).