Intelligent Collision Avoidance System for Industrial Manipulators

Thadeu Vinícius de Brito

Dissertação apresentada à Escola Superior de Tecnologia e Gestão do Instituto Politécnico de Bragança para obtenção do grau de Mestre em Engenharia Industrial ramo de Engenharia Eletrotécnica.

Trabalho orientado por:

Prof. Dr. José Luís Sousa de Magalhães Lima

Prof. Dr. Roberto Ribeiro Neli

Prof. Dr. Pedro Gomes Costa

Bragança

2018

Intelligent Collision Avoidance System for Industrial Manipulators

Thadeu Vinícius de Brito

Dissertação apresentada à Escola Superior de Tecnologia e Gestão do Instituto Politécnico de Bragança para obtenção do grau de Mestre em Engenharia Industrial ramo de Engenharia Eletrotécnica.

Trabalho orientado por:

Prof. Dr. José Luís Sousa de Magalhães Lima

Prof. Dr. Roberto Ribeiro Neli

Prof. Dr. Pedro Gomes Costa

Bragança

2018

"Quando penso que cheguei ao meu limite, descubro que tenho forças para ir além."

Ayrton Senna.

Dedication

I dedicate this work to my mother, Mariza Cristina Silvia de Brito. That by the tireless way she created me and educated, accompanied and helped, made me the person that I am. This work is also hers, because without my mother this work would not have been possible to get here. Thank you mother for being the foundation of my life!

Acknowledgments

The *Escola Superior de Tecnologia e Gestão* (ESTiG) of Polytechnic Institute of Bragança (IPB), for all the conditions it gave me. To the *Módulo de Investigación Cibernética* (MIC) of *Universidad de León* (ULE), which likewise provided all the conditions for performing this work. And also to Federal University of Technology - Paraná (UTFPR), who fostered my academic interest. Thanks to these three institutions, it was possible to meet great professionals and carry out the whole project of this work presented.

To professor José Luís Lima, supervisor of this master's thesis. And to the co-supervisors, the professors Pedro Gomes da Costa and Roberto Ribeiro Neli. Also to teachers Vicente Matellán and Ángel Manuel Guerrero. I want to express my sincere thanks for the incentives, guidance, support, patience and friendship.

Finally, I would like to express to all the friends that life has given me during these 5 years of graduation. Luckily the list of these friends is great and I can not list them, thank you all for the hours of leisure, reflections, good and bad memories, moments of studies and experiences that I will never forget.

Abstract

The new paradigm of Industry 4.0 demand the collaboration between robot and humans. They could help (human and robot) and collaborate each other without any additional security, unlike other conventional manipulators. For this, the robot should have the ability of acquire the environment and plan (or re-plan) on-the-fly the movement avoiding the obstacles and people.

This work proposes a system that acquires the space of the environment, based on a Kinect sensor, verifies the free spaces generated by a Point Cloud and executes the trajectory of manipulators in these free spaces. The simulation system should perform the path planning of a UR5 manipulator for pick-and-place tasks, while avoiding the objects around it, based on the point cloud from Kinect. And due to the results obtained in the simulation, it was possible to apply this system in real situations.

The basic structure of the system is the ROS software, which facilitates robotic applications with a powerful set of libraries and tools. The MoveIt! and Rviz are examples of these tools, with them it was possible to carry out simulations and obtain planning results. The results are reported through `logs` files, indicating whether the robot motion plain was successful and how many manipulator poses were needed to create the final movement. This last step, allows to validate the proposed system, through the use of the RRT and PRM algorithms. Which were chosen because they are most used in the field of robot path planning.

Keywords: Collaborative robots; manipulator path planning; collision avoidance; rgb-d sensors; Point Cloud.

Resumo

Os novos paradigmas da Indústria 4.0 exigem a colaboração entre robôs e seres humanos. Estes podem ajudar e colaborar entre si sem qualquer segurança adicional, ao contrário de outros manipuladores convencionais. Para isto, o robô deve ter a capacidade de adquirir o meio ambiente e planear (ou re-planear) *on-the-fly* o movimento evitando obstáculos e pessoas.

Este trabalho propõe um sistema que adquire o espaço do ambiente através do sensor Kinect. O sistema deve executar o planeamento do caminho de manipuladores que possuem movimentos de um ponto a outro (ponto inicial e final), evitando os objetos ao seu redor, com base na nuvem de pontos gerada pelo Kinect. E devido aos resultados obtidos na simulação, foi possível aplicar este sistema em situações reais.

A estrutura base do sistema é o *software* ROS, que facilita aplicações robóticas com um poderoso conjunto de bibliotecas e ferramentas. O MoveIt! e Rviz são exemplos destas ferramentas, com elas foi possível realizar simulações e conseguir os resultados de planeamento livre de colisões.

Os resultados são informados por meio de arquivos `logs`, indicando se o movimento do UR5 foi realizado com sucesso e quantas poses do manipulador foram necessárias criar para atingir o movimento final. Este último passo, permite validar o sistema proposto, através do uso dos algoritmos RRT e PRM. Que foram escolhidos por serem mais utilizados no ramo de planeamento de trajetória para robôs.

Palavras-chave: Robô colaborativos; planeamento de manipulador; evitar obstáculos; sensor RGB-D; Nuvem de Pontos.

Contents

List of Tables

List of Figures

Acronyms

ABB Asea Brown Boveri.

BSD Berkeley Software Distribution.

CMT Cold Metal Transfer.

CNC Computer Numerical Command.

CPS Cyber-Physical Systems.

DOF Degree Of Freedom.

ESTiG *Escola Superior de Tecnologia e Gestão.*

FCL Flexible Collision Library.

IFR International Federation of Robotics.

IPB Polytechnic Institute of Bragança.

ISO International Organization for Standardization.

MIC *Módulo de Investigación Cibernética.*

MIG Metal Inert Gas.

OMPL Open Motion Planning Library.

OROCOS Open Robot Control Software.

PCL Point Cloud Library.

PR Personal Robots.

PRM Probabilistic Roadmap Method.

RDS Robotics Developer Studio.

RGB Red, Green and Blue.

ROS Robotic Operating System.

RRT Rapidly-exploring Random Trees.

SCARA Selective Compliance Assembly Robot Arm.

SDF Simulation Description Format.

SDK Software Development Kit.

STAIR STanford AI Robot.

ULE *Universidad de León.*

URDF Unified Robot Description Format.

UTFPR Federal University of Technology - Paraná.

V-REP Virtual Robot Experimentation Platform.

VPL Visual Programming Language.

Xacro XML Macros.

Chapter 1

Introduction

The ability to estimate and avoid collisions for a robotic manipulator is just one of many other perspectives of Industry 4.0. Collaborative robotics is a topic addressed in Industry 4.0 where humans and robots can share and help each other cooperatively. The collaborative robot can be used without any additional security, unlike other manipulators. This portrays that the robot must have the ability to acquire the working environment and plan the movement avoiding obstacles and people.

1.1 Theoretical framework

The task of cooperation between humans and robots requires that the robot can re-plan the trajectory in which it reaches the target position in a way that avoids the collision with obstacles and humans in real time, which means that this will happen in on-the-fly (while the robotic arm is in motion). This process can be called Dynamic Collision Avoidance.

To aid in the ability to acquire the working environment, RGB-D sensors can be ordered [1]. Helping in the acquisition and perception of this environment so that the system can make the planning of the path with restrictions. This type of device has high popularity, not only for this feature, but also for low cost. Example of this is the Kinect sensor.

1.2 Objectives

The manipulation of objects by robotic arm is growing in the industrial sector. However, they are expected to collaborate with humans. In this case, avoiding collision during trajectory is a very important requirement. Based on a robotic manipulator, it is intended to develop a security system capable of avoiding the collision of manipulators with objects and humans. That is, having an anthropomorphic manipulator and giving it the ability not to collide with obstacles in the work environment while performing the planning of movements, as shown in Figure 1.1. Thus, the collaborative robot can then perform different movements to reach the desired point, different from the way it is done in the present day.

This work proposes a system that acquires the workspace of manipulators, based on a Kinect sensor. Performs path planning by avoiding the objects and people who are in this environment. The developed system must first be elaborated by simulation, in order to avoid possible risks. If evaluated as fit, it should be applied in real environments.

Figure 1.1: Tool diagram for manipulators to avoid collisions.

1.3 Structure of the document

Within the scope of the curricular unit of degree of Master in Industrial Engineering, branch of Electro-technical Engineering, presents itself to ESTiG this document developed in 7 chapters. The integration of these reports all work developed in IPB, here are the synopses of each of these elements.

The Chapter 1, emphasizes the characterization of the problem addressed, and in turn are clarified the objectives proposed for solution.

In Chapter 2, the state of the art is surveyed, that is, the bibliographic revision necessary for the development of the work. Where some studies are arranged over the years, which demonstrate the history of industrial manipulators.

For the study of the work, the Chapter 3 is defined, where all the content necessary to understand the development of the work is emphasized. The knowledge referred to begins with the introduction of industrial robots, robotics and concludes in the components of perception.

For the purpose of working methods, Chapter 4 is defined. In this is analyzed in detail all the solutions adopted due to the problem presented, the procedures followed for the development of the system and the reasons of the tools used.

The work developed for the proposal presented has evolved in its course. Chapter 5 exposes all the evolutions that the work has undergone during its development, from the simulation to the application in real situations.

The results obtained in the real manipulator of the developed system are presented in Chapter 6. And therefore the conclusions of this work are defined in Chapter 7.

Chapter 2

State of art

This Chapter 2 has the state-of-the-art research, in other words, the bibliographic review necessary for the development of the work. Where some studies are arranged over the years, which demonstrate, in Section 2.1, the history of industrial manipulators. Following in the Section 2.2, some important studies in the development of the pick-and-place operation. Then, in Section 2.3, the different types of industrial robots and their histories are related. For the history of the studies in manipulator planning and robotic perception, in Sections 2.4 and 2.5 are defined respectively.

2.1 Industry with manipulator robots

The industry and robotic development have been in joint progress since 1938, which in turn is the same year that the first crane capable of performing tasks in small constructions was developed [2]. However the first industrial robot is only developed years later, in 1959 [3]. It is possible to imagine, these robots were too heavy, they used hydraulic actuators and their accuracy was reduced. Another factor that made them obsolete is present in the continuous repetition of the same movements [4].

Until the present time, the technique of manipulating objects with mechanical arms is on the rise in all industrial sectors [5]. The tasks of transporting an object from one place to another are commonly called pick-and-place [6]. And to follow in the constant of

development, the article published by International Federation of Robotics (IFR) mentions that the advances in manipulators must follow in the branches of robotic intelligence and in the simplification in the construction of the robots [7]. This is justified in the use of industrial manipulators in various sectors, this robots can be seen in drug packaging operations [8], in automation systems for food [5] and micrometer scale accurately enough to manipulate dimensions of objects below the ten micrometers [9].

2.2 Pick-and-place operations

In 2000, Through studies of the main problems of pick-and-place operations it was possible to classify the pick-and-place operation into two segments: object detection and Gripping [10], [11]. Already in 2005, algorithms were developed to perform pick-and-place motions in real time, using multi-chamber image processing [12]. In 2015, a survey addresses the visual control problem for a robotic arm with 5 Degree Of Freedom (DOF) that performs pick-and-place tasks. In this research it is proposed a visual control structure inspired by detection and reaction behavior, without the use of computation. To accomplish this, a different approach is taken in the reactive function of the hardware, by mapping vectors with image errors directly. With this, the new approach helps simplify the application of servo visual systems with greater reliability by solving visual problems in real pick-and-place applications [13]. In 2016, researchers incorporated the controller of an industrial manipulator based on the location and orientation of an object in 2D coordinates. Comparing some problems of sensor cycles with human manipulation, the work seeks to find the position x and y, in coordinate format. By performing the conversion of pixel coordinates into real-world coordinates, the search attempts to enhance pick-and-place maps by copying the human view.[14]. In 2017, an efficient algorithm was developed to operate with pick-and-place manipulations in domestic environments. This type of environment is a problem for the autonomous assembly, however the research sought to apply the triangulation of 2D images to solve the task. By calculating the total number of pixels of a piece, an efficient algorithm was created that is capable of recognizing objects and

capturing them. This same algorithm contains routine learning and training functions for autonomic robots, and can be applied in real-time pick-and-place operations. [15].

For the calibration of the gripper position of the manipulator, a research was developed in 2010, proposing a strategy of dividing the calibration in three steps. The first is to align the claw with the part in the plane of the captured image, the second is to align the claw with the part in the normal direction of the plane of the image and the third is to close the claw until the parts is fully secured [16]. And in 2014 the research with the cooperation between two robots conducting the joint operation of pick-and-place [17].

As for the control system of manipulators, in 2002 a work was proposed with the use of neuronal algorithms that allowed tasks in environments with obstacles. The main idea of the research was to elaborate planning techniques of optimal control for assembly of robotic parts, with the use of sensors. The system works by joining pressure and vision sensor data, informing the assembly conditions of a part. As a result of this research, the developed system and technique can be applied to assembly tasks and pick-and-place operations, improving the efficiency of manipulator control [18]. In 2011, the comparison between two algorithms through some criteria and associated planning discussions paths, sought to improve the system of control of industrial manipulators. The study sought to elaborate two situations, the first sought to identify the input and output pattern of a neural network and the second was applied the learning process. In the results of the research it was possible to verify the probability of success in search of planning of paths with the change of parameters in the control system [19].

2.3 Types of Robotic arms

Taking into consideration the construction geometry of an industrial manipulator, in 2006, it was possible to classify in two classes: series and parallel [20]. As a result, in the same year of 2006, a study proposed the application of parallel manipulators in pick-and-place operations, the result of this approach originated the labels of Cartesian, Cylindrical, Spherical, Anthropomorphic and Selective Compliance Assembly Robot Arm

(SCARA) [21]. And in 2010 another study on the geometry of industrial manipulators demonstrated the efficiency of parallel manipulators in pick-and-place operations, the Delta format robots [22]. Adept became the first company to market Delta robots [23].

Another way to perform the pick-and-place operation is by means of cable robots [24], in 2007 the idea of performing this operation was conceived by some authors [25], and in 2012 there was the development of a system for locating patients between hospital beds, which used this genre of robot [26]. Cable robots are controlled by an end effector using multiple actuated cables [27], which are controlled by positioning systems acting on motors for winding or unwinding cables [28]. There are few cable robotic systems on the run these days, however, is possible to find literatures that deal with the theme. [29], [30], [31].

2.4 Path planning and algorithms

An important step to consider when developing a robotic manipulator system is path planning, being a key area of robotics. It includes planning algorithms, configuration space discretization strategies, and related constraints. It is well known that planning the path for robots with many degrees of freedom is a complex task.

In 1991, a new approach to robot path planning took place, which consisted of building and searching a graph that connects the local minimums of the potential function defined in the configuration space of the robot [32]. This new approach was proposed considering robots with multiple degrees of freedom. Later, in 1994, this same algorithm was refined with the purpose of accelerating the system, calculating solutions with less estimated runtime [33]. And in 1996, probabilistic methods were introduced with the goal of reducing the complexity of configuration free space. These methods generate nodes in free space and connect them via viable paths to create a graph [34].

However, this method is not adapted for dynamic environments, since a change in the environment causes the reconstruction of the entire graph. Then some variants of these methods have been proposed: Visibility based Probabilistic Roadmap Method (PRM)

[35], Medial axis PRM [36], Lazy PRM [37] and sampling based roadmap of trees, the Rapidly-exploring Random Trees (RRT) [38].

Over the years other methods have also been studied, such as the method for planning local paths for manipulator robots. This solved the problem of local minimums by doing a search on graphs that describe the local environment using the algorithm A*, until the local minimums are avoided [39].

Path planning becomes more complex when obstacles are introduced in a given environment. In 2006, an algorithm that addresses the problem with disjunctive programming is presented. Ability to use existing constrained optimization methods to generate optimal trajectories for obstacle handler path planning [40].

The real-time path planning method is presented, also in 2006, in dynamic environments. This approach is based on the constraints method, along with the procedure to avoid locations, bypassing obstacles using a threshold that is part of the strategy [41]. In 2016, the double A* algorithm is applied to multiple industrial manipulators, which tries to approximate the target and a more refined A* to reduce the error [42].

2.5 Perception in robotics

It is intended that the new generation of industrial robots work with humans in an efficient and secure way, so new manipulators will be implemented in dynamic environments [43]. For this to occur it is necessary to realize the perception of the environment in which the manipulator is applied [44]. In the year 2003, researchers developed a vision system capable of recognizing a specific object mixed among others [45]. So in 2013, another study proposes that these collaborative robots can perceive static obstacles and avoid them, as well as dynamic obstacles [46]. In order to develop a Cyber-Physical Systems (CPS) system in industrial environments, in the year 2016, a system capable of detecting dynamic obstacles around a manipulator of 7 DOF is applied [47].

Chapter 3

Study of tools

In this Chapter 3 all the theoretical content used for the understanding and development of the work is addressed. Started by the study of industrial robots in Section 3.1 and addressing current manufacturers of industrial manipulators in Section 3.2. In the Section 3.3 are the motion components, and in Section 3.4 the algorithms and planners that aid in these tasks. And consequently the softwares to make the configurations in robotic applications in Section 3.5, soon afterwards the software chosen to carry out this work in Section 3.6. Finally, the functionality of a Point Cloud and the depth sensors, in the Sections 3.7 and 3.8, respectively.

3.1 Industrial Robots

The word robot came from an English translation made from a tale created by the Czech author Karel Capek in 1923, in his native language the word *robota* means: a worker. In this tale, robots were some kind of humanoid machine, created by humans, for the purpose of obeying a service [48].

However, it took more than 40 years of development in technology to achieve a computer-controlled mechanical manipulator. This technology was named robotics [49]. The International Organization for Standardization (ISO) defines as industrial manipulator [50]:

> "Automatically controlled, reprogrammable, multipurpose, manipulator, programmable in three or more axes, which can be either fixed in place or mobile for use in industrial automation applications" [ISO 8373:2012] [51].

Still in the same ISO standard are other definitions for terms used in robotic applications. As:

- **Manipulator**: Machine in which the mechanism generally consists of a series of segments, hinged or slidable relative to one another, for the purpose of grasping and/or moving objects (parts or tools) generally in varying degrees of freedom;
- **Reprogrammable**: Designed so that the programmed movements or auxiliary functions can be changed without physical change;
- **Physical change**: Change of mechanical system (changes in mechanical system do not consider storage media, ROM and etc.);
- **Multifunctional**: Capable of being adapted to a different application with physical change.

In this work, it is considered that a robot is a set of articulated mechanisms, controlled by computerized electronic circuits. The following terms will be used to define devices that fall under this definition: robot, manipulator, industrial manipulator, robotic manipulator, robotic arm, mechanical arm and robotic mechanism.

3.1.1 Anatomy

The working capacities of each manipulator range from simple repetitive movements from one point to another, to versatile movements that can be controlled by computer. But before understanding how the moves are performed, it is important to understand the physical construction of a manipulator [48].

Usually handlers are fixed on some type of floor, this part is called the base. This base of the robot is used to support the body, which serves to make the connection with the arm. At the end of the arm is the handle, which will be where the tool is installed to do the job [49], in Figure 3.1 it is possible to notice the entire mechanical structure through an illustration..

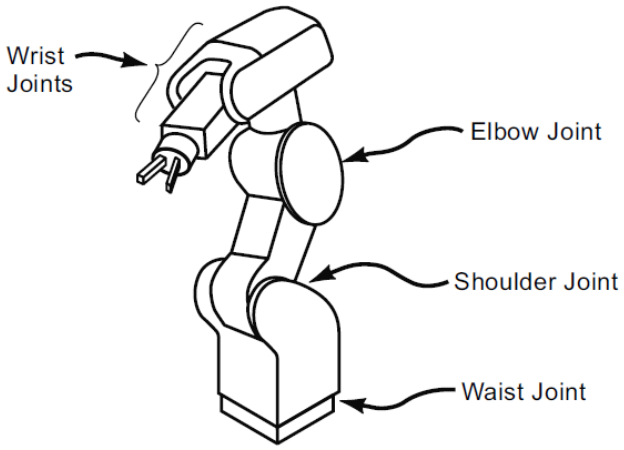
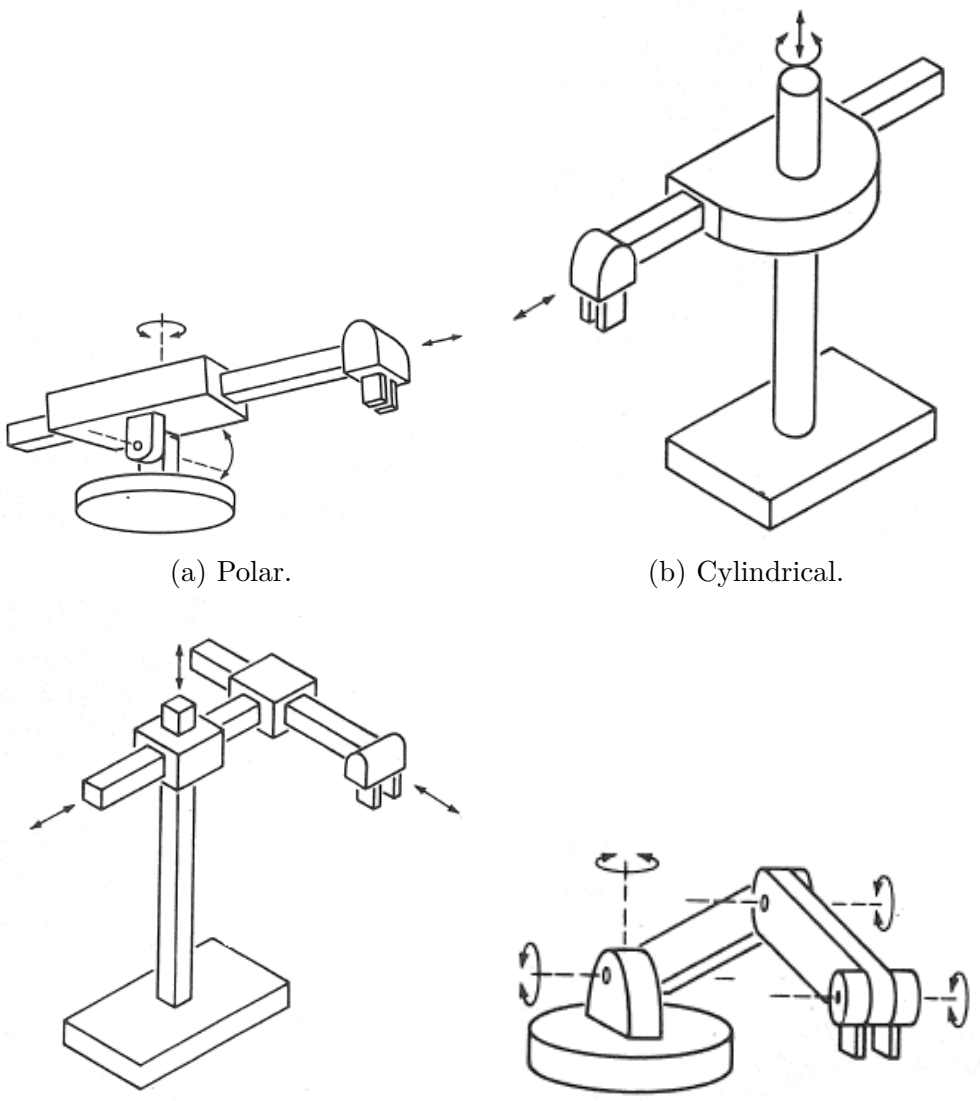


Figure 3.1: Illustration of the mechanical scheme of an industrial manipulator [52].

There are various sizes, shapes and physical configurations of manipulators. But four basic configurations dominate the market, in Figure 3.2 these basic configurations are illustrated. The Polar configuration (Figure 3.2a) uses a telescoping arm that can be lifted or lowered around a horizontal axis, this axis is attached to a base. The Cylindrical configuration (Figure 3.2b) is very similar to the polar configuration, however it uses a vertical column. In the Cartesian configuration (Figure 3.2c) three perpendicular axes are used. And the Articulated configuration (Figure 3.2d) imitates a human arm, where each element (or links) of the manipulator is sequentially connected by joints [49].

As shown in Figure 3.3, the movements of a robot are accomplished by means of driven joints. These joints form the union of the links, which are the rigid members of the robot [49]. The way the joints between joints and links happen, can generate different movements, Figure 3.4 illustrates these movements and their respective names. Each link between the joint and the link needs an axis between them. This articulated axis provides, for the robot, a DOF in movement [52].



(a) Polar.

(b) Cylindrical.

(c) Cartesian.

(d) Articulated.

Figure 3.2: Mechanical scheme of the basic anatomy of an industrial manipulator. Image adapted [49].

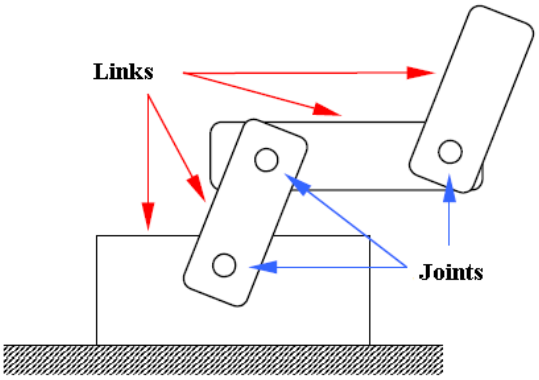
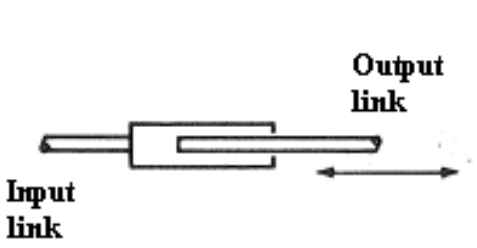
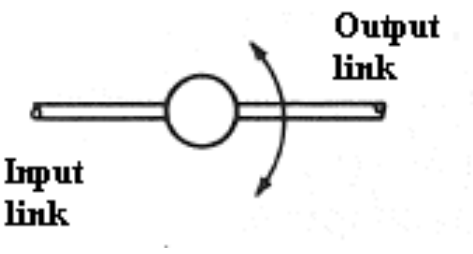


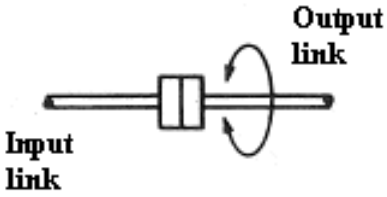
Figure 3.3: Example of a mechanical diagram of the joint between the joint and the joint. Image adapted [53].



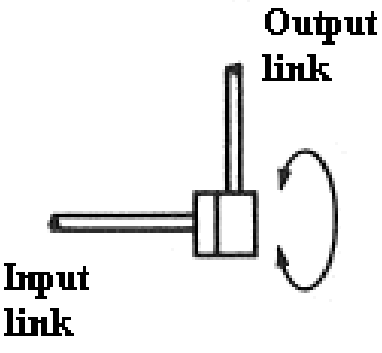
(a) Joint called linear, with sliding or translational movement of the links, type L.



(b) A so-called rotational joint, with perpendicular movement between the links, of type R.



(c) A so-called torsion joint, with parallel movement between the links, of type T.



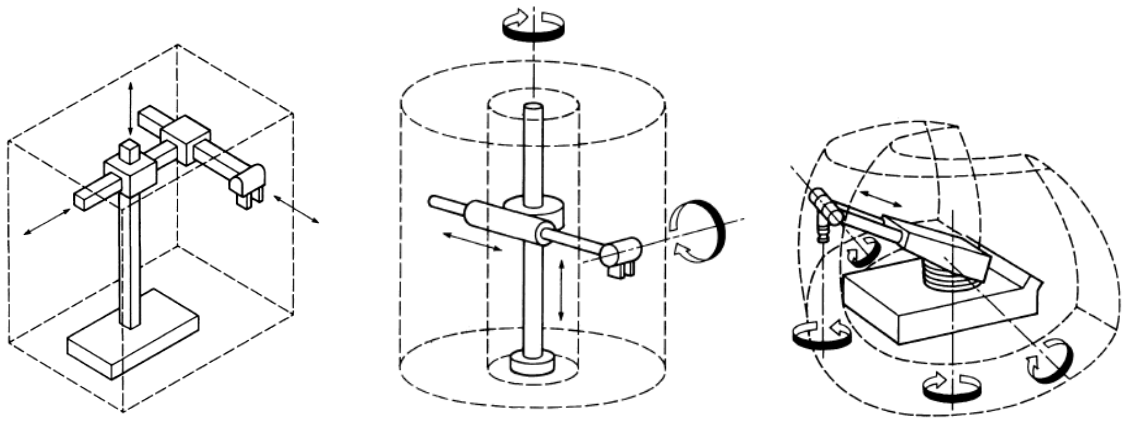
(d) A so-called revolving joint, with movement of one of the links surrounding another link, of type V.

Figure 3.4: Mechanical illustration of joints for an industrial manipulator. Image adapted [49].

3.1.2 Workspace

Every place where the manipulator can move the tool is called the workspace. Some literatures point out that it would not be the tool, but the wrist movement. The important thing is to analyze the scope of the manipulator [54].

The size of the workspace is influenced by the dimensions of the elements of the manipulator, ie it is necessary to ensure that the base, arm, handle and tool will have the necessary measure for the space of work that is desired. The physical configurations of the manipulator also influence the workspace, since certain physical configurations can not perform some movements [55]. These aspects are shown in Figure 3.5.



(a) Workspace of the Cartesian manipulator.

(b) Workspace of the cylindrical manipulator.

(c) Polar manipulator workspace.

Figure 3.5: Illustration of the various workspaces of industrial manipulators. Image adapted [56].

3.1.3 Robotic sensors

Acquiring the description of a working environment has always been a challenging task for robotic applications. However, this task was simplified with the development of sensors [1]. Developing a robotic application requires the use of sensors, there are currently several types of sensors that support these applications. The main objective of a sensor is to indicate to the control system of the manipulator how is the working environment, in order to achieve the purpose of the application according to the environment [56].

Each type of sensor is developed for certain cases, so they are classified according to the physical principles on which they are based. Position sensors determine the positions of the links or external elements, such as the encoder that informs the position by pulse counts. The touch sensors inform the control system, by means of a binary signal, if the manipulator has collided with an object. The pressure sensors have a certain mechanical structure that determines the pressure on a certain mechanical deformation [53]. Other sensors can perform the acquisition of the environment, such as depth sensors, in Section 3.8 there is more information about this type of sensor.

3.1.4 Applications of industrial robots

After the great advances in robotics, there has been an explosive growth in the range of robotic applications in the industrial sector, so characterizing all applications requires a more detailed approach. In this section the focus is on a selection of applications of particular interest, usually because of their widespread use or because it points the way to the future.

Robotic technology should be considered whenever productivity improvements are sought because this technology is reliable, accessible, and suitable for hazardous or repetitive human work environments [52]. Dangerous or uncomfortable workplaces are those where human workers may be exposed to high temperatures, dangerous atmospheric conditions, handling of harmful objects, difficult working positions (such as kneeling or squatting), in short, any cause significant stress to the worker. Work environments that require a high degree of precision or are highly repetitive also indicate the use of robots. Since modern robots are more accurate and reliable than human workers [52].

Another factor that sets them apart from humans is that they do not get bored or distracted and therefore are ideal for repetitive work. Robotic application in this type of environment inevitably produces the additional benefit of better quality in industrial production [52]. The use of industrial robotic applications for handling tools or workpieces is an established practice in the industrial sector. An example of such manipulation by

robots is to perform tasks such as loading and unloading tools on machining machines (as shown in Figure 3.6), or to move components from one conveyor system to another [56]. Robotic applications can be characterized by many areas, however the three main areas are: material handling, processing operation, assembly and inspection.

Material Handling

Material handling is a good area with which to begin a study of industrial robot applications, as it immediately introduces some of the most important principles. Robots used for handling are usually relatively simple devices. In this type of application, the manipulator simply moves the part to a position in which it can be processed. Within this branch, machines like: feeders (as in Figure 3.6), conveyors and automatic cranes are well established in the industry. More recently, industrial robots have become increasingly applied in handling, particularly at low cost [56].

In choosing and in place or transferring material, the robot takes an object from one place and moves or transfers to another, so the tool at the tip of the manipulator used will usually be a forceps. Loading and unloading of some machines are common materials handling applications. Used for various types of processes, such as: casting, machining, injection molding, pressing, forging and heat treatment [52].

Figure 3.6: Task of loading and unloading of parts, achieved through a manipulation robot before and after the machining process [57].

Processing operation

The most common processing operation applications for industrial manipulators include: spot welding, arc welding and spray painting. In spot welding processes, the robot is typically large because it is necessary to perform work with load. As in Figure 3.7 a manipulator performs standard Metal Inert Gas (MIG) and pulsed Cold Metal Transfer (CMT) welding tests. And usually with 6 DOF can easily manipulate a heavy welding tool in tight spaces [52].

For arc welding processes robotic technology is recommended, due to the fact that these processes are dangerous and very repetitive. Therefore, the use of robots is an advantageous solution to avoid accidents. Other areas in which this robotic technology is heavily used are spray painting and machining processes [52].

Figure 3.7: Manipulator with tool for welding, conducting tests of Standard MIG welding and pulsed CMT in aluminum alloy [58].

Assembly and inspection

The assembly and inspection process is yet another area where robots are often applied. Some robot models have advantages as to the purpose of this application, robots of the SCARA type and Cartesian types, offer a good set of components and have high vertical stability, as mentioned in the Section 3.1.1 [52].

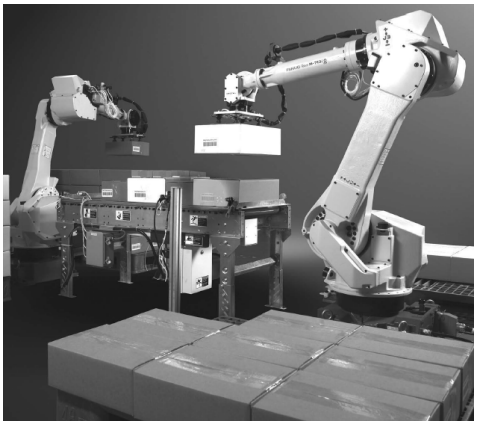


Figure 3.8: Example of assembly of parts aided by robotic manipulators [52].

The use of manipulators in automatic assembly systems will be particularly important in the future, as seen in Figure 3.8. Currently the development of robotics is going through a considerable effort to meet the automatic assembly. However, some successful installations are already being created [56].

Assemblies of small mechanical and electrical components, automotive assemblers, and consumer electronics, are some areas where assembling robots are beginning to be applied [56]. In inspection applications, the robot can either carry the part to the inspection device, or take the inspection device to check the part [59].

3.2 Manufacturers of industrial robots

There is a wide variety of companies that manufacture industrial robots in the world, to meet the needs of industrial production. Industrial robotics have become an indispensable part of manufacturing because of their persistent accuracy. In this section are some of the largest manufacturers of industrial robots and some examples of models manufactured.

According to the IFR executive report, the expectation of sales of industrial robots for the coming years is quite promising. Data show that between the years 2017 and 2020 growth has an average of close to 15% per year [7]. In Figure 3.9 is possible to check the organization's report data. To meet the industrial market, some manufacturers of industrial robots lead the sales market. Those who sold the most in 2016 [60]:

- Asea Brown Boveri (ABB) Ltd.;
- Mitsubishi Electric;
- Fanuc Corp.;
- Yaskawa Electric Corp.;
- Adept Technology Inc.;
- Stäubli.

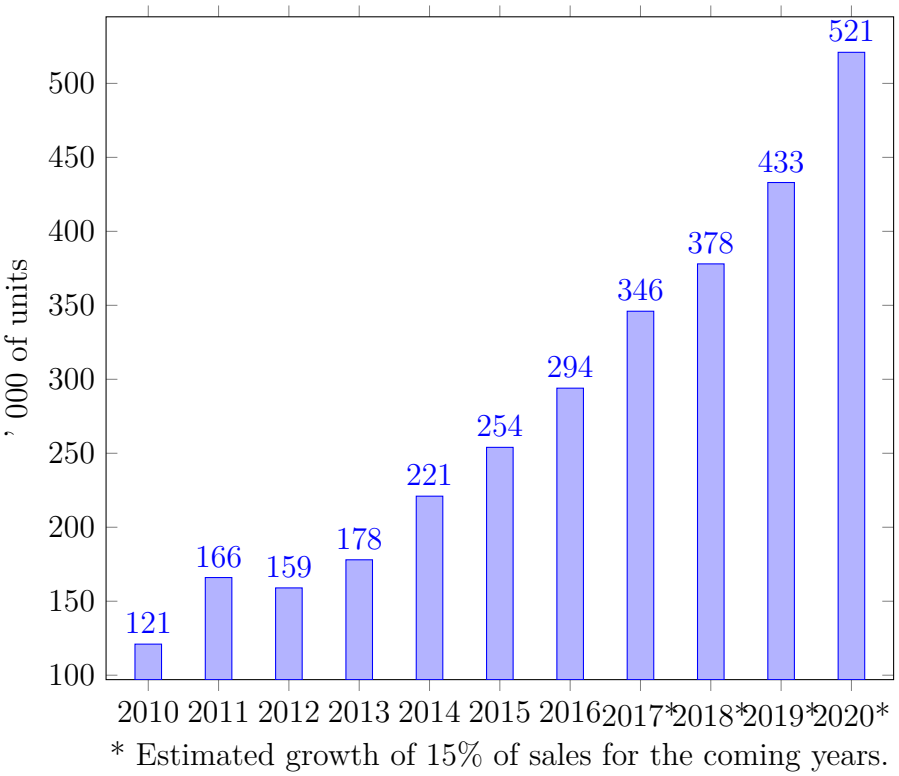


Figure 3.9: Record of previous years and annual worldwide expectation of sales of industrial robots for the coming years, data extracted from [7].

Some manufacturers can still draw attention in the sales market. Even though it does not appear in the list of best-selling products, the following list shows some companies that also perform services for the industrial sector.

- Cloos [61];
- Comau [62];
- DENSO Robotics [63];
- Epson Robots [64];
- Hyundai Robotics [65];
- IGM Group [66];
- IAI [67];
- Janome [68];
- Kawasaki [69];
- KUKA [70];
- Nachi [71];
- Nidec Sankyo [72];
- OTC [73];
- Reis [74].

ABB Ltd.

ABB is one of the leading suppliers of software for robots, equipment and industrial robots. This industrial robotics company has already installed more than 300,000 robots, which holds the most extensive service network in the industry. ABB industrial robots improve energy efficiency, reliability and productivity for industrial customers, utilities and infrastructure [60], Figure 3.10 shows some of the company's models. ABB's suite of products includes: Control room solutions, Drivers, High voltage products, Low voltage products and systems, Measurement and analysis, Mechanical energy transmission products, Medium voltage products, Robotics, Automation.

(a) Articulated robot, model IRB 1660ID.

(b) Parallel robot, model IRB 360 FlexPicker.

(c) Robot with two arms, model IRB 14000 YUMI.

Figure 3.10: Some models of industrial robots sold by ABB [75].

Mitsubishi Electric

Mitsubishi Electric has been creating high quality robots for the automation industry over the years. This company provides compact, powerful and accurate robots for the needs of industrial automation [60]. Mitsubishi Electric has the main applications for: Materials Handling, Pick and Place, Material Removal, Assembly, Distribution, Packaging and Palletizing. Figure 3.11 shows some of the company's models.

Mitsubishi's robotics has the main innovations to handle efficiency, flexibility, safety and ergonomics. The company's technology can increase security and increase productivity for industries [60], in the sectors of: Health Care, Education, General Manufacturing, Packaging, Food & Beverage, Automotive, Semiconductor electronics.

(a) Robot orbital type SCARA, model RH-FHR.

(b) Robot of the horizontal type SCARA, model RH-F.

(c) Robot of the vertical type, model RV-F.

Figure 3.11: Some models of industrial robots sold by Mitsubishi Electric [76].

Fanuc Corp.

FANUC America offers a series of products and services, such as: Computer Numerical Command (CNC) systems and factory automation solutions. This company provides its customers with robotic technology capable of improving efficiency, reliability, quality and profitability. FANUC's automated assembly systems can be assembled continuously through inspection tests [60]. Some of the main products of FANUC include: Collaborative Robots, Genkotsu-Robot, Mini Robot, Arc Welding Robot, Large Size Robot, Palletizing Robot, Robot Painting. Figure 3.12 shows some of the company's models.

(a) Collaborative robot, model CR-7iA-L.

(b) Articulated robot, model M-10iA-7L.

(c) Top mounting robot, model M-20iA-20MT.

Figure 3.12: Some models of industrial robots sold by FANUC America [77].

Yaskawa Electric Corp.

Yaskawa Electric Corp. has created a corporate image based on its management philosophy of contributing to the evolution of society and the well-being of mankind through the performance of its business. This company strengthens the servo motors, controllers, CA Drivers and industrial robots business. Offering the customer innovative solutions through essential technologies [60]. The most common areas of operation are: Robots for painting, Robots for manipulation, Vacuum transfer robots for semiconductor and LCD manufacturing equipment, Arc and dot welding robots. Figure 3.13 shows some of the company's models.

The company also provides vertical articulated robots, such as products that contribute to the automation of welding, painting, assembly, transportation and others. The main focus is to serve the production of markets related to automobiles and several other fields. The Yaskawa product line: CA Drivers, Servo Drivers and Machine Drivers, Robots, Systems Engineering, Energy Saving and Creation Equipment.

(a) Articulated robot, model VS100.

(b) Articulated robot with two arms, model SDA10F.

(c) Articulated robot for painting, model MPX3500.

Figure 3.13: Some models of industrial robots sold by Yaskawa Electric Corp. [78].

Adept Technology Inc.

Adept Technology Inc. has been serving the industry in developing innovative and powerful industrial robots for high precision manufacturing, packaging and automation. The innovations of industrial robots of this company continue in their expansive lines of robots. Providing cost-effective robotics systems and services to high growth markets including electronics and semiconductors, packaged goods, traditional industrial markets, automation of machines and automotive components. Adept is also ISO 9001: 2000 certified [60]. Its products are: Parallel Robots, SCARA Cobra Robots and Viper Six-Axis Robots. Figure 3.14 shows some of the company's models.

(a) Robot SCARA, model eCobra 600.

(b) Articulated robot, model Viper s650.

(c) Parallel robot, model Quattro s650H.

Figure 3.14: Some models of industrial robots sold by Adept Technology Inc. [79].

Stäubli

Stäubli is a company that provides mechatronics services with three dedicated activities: Connectors, Robotics and Textiles. Serves customers who want to increase their productivity in many industries. This company offers innovative solutions for all industrial sectors worldwide. Stäubli provides a series of robotic products covering all industries and applications. Through its extensive line of products, such as industrial robotic arms, the company offers distinct technical advantages: compact size, large work volume, high speed and precision [60]. Stäubli's robotic product line: Robotic arms, Robot controllers and Robot software. Figure 3.15 shows some of the company's models.

(a) Robot SCARA, model FAST picker TP80.

(b) Articulated robot, model RX160 HE.

(c) Robot SCARA, model TS80 ESD.

Figure 3.15: Some models of industrial robots sold by Stäubli [80].

Universal Robots

Universal Robots specialize in projects at universal levels, so it is included in the name of the company. This means that the installation of the company's products can automate almost everything from the assembly phase to the painting, the tightening of screws to the labeling process, the packaging to the polishing, the injection molding to the welding [81].

Another goal is to make collaborative robot technology accessible to companies ranging from small to large. The product line offered by the company has precision, speed, work optimization and ergonomics (relationship between man and machine safely). These factors contribute to the financial return, whose average time is 195 days [81]. In Figure 3.16 is possible view the available models.

Robotic models developed by the company can be implemented virtually in any industry, in any process and by any employee. This is of great importance in the project creation stage, since it is possible to simulate beforehand in order to avoid failures [82].

The UR line robots have a variety of safety features coupled as well as electrical interfaces with safety rating to connect to other machines and additional protective equipment. Each interface and safety function is monitored according to EN ISO13849-1: 2008 [82]. This encourages the robot to work side-by-side with employees without extra security installations.

Since its integrated sensitivity of force causes the robots to stop working automatically, when faced with obstacles in their trajectory. Another feature of these models is that they can be programmed to operate in reduced mode when a worker is at their workload [82].

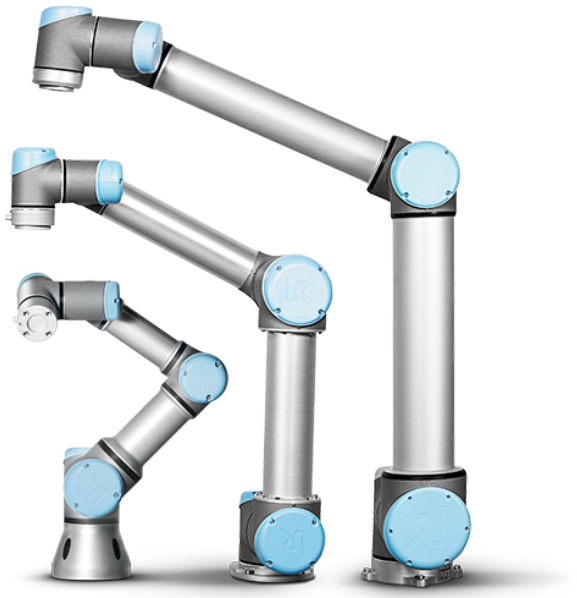


Figure 3.16: Universal Robots collaborative robot models [81].

3.3 Kinematics of manipulators

To perform manipulator movements, techniques must be developed to identify the positions of certain points on the arm. This identification depends on the structuring of the joints and the links (discussed in Section 3.1.1) in reference to some point, which is usually the basis of the manipulator itself. In this section we present a technique to identify the positions of the robotic arm and move them and in the end an example for manipulators with 2 and 3 DOF.

The kinematic and dynamic models of manipulator robots are needed both to perform motions in real applications and in simulations on any platform [83]. Kinematics for manipulators is responsible for studying the movements without considering the causes that give rise to them [49], so the kinematics will deal with distances, angles, accelerations and translational and angular velocity [59]. And the dynamics is the analysis of the movement considering the forces, masses, moments of inertia and torques. These studies are of great importance for articulated arms as they are widely used by industry in determining the position of the tool [53].

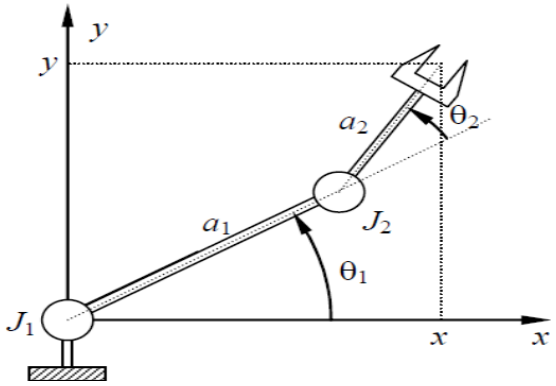
There are two kinematic branches, forward kinematics and inverse kinematics. The first deals with modeling the position and orientation of the tool, given the angles of the joints. The other focuses on finding the joint angles from the position and orientation of the tool. In turn, inverse kinematics is a more complex process than forward kinematics and the resolution methods vary depending on the robot layout [84].

The position of each joint in space is realized through rectangular axes, they are the so-called Cartesian coordinates. The kinematic equation, forward or inverse, requires knowledge of the length of the links with precision. The manipulator manufacturers provide any existing displacements between joints, so it is possible to correctly calculate the Cartesian position [53].

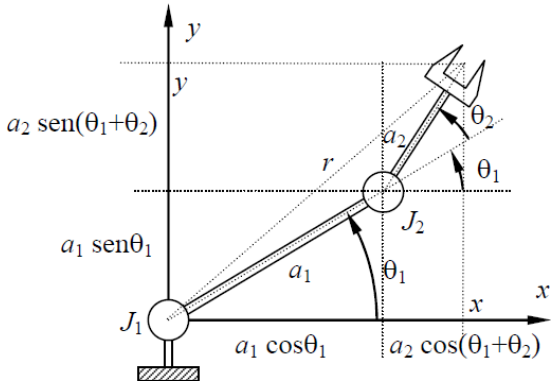
For any mechanical anatomy, the solution of kinematics requires knowledge of geometry, trigonometry and vector calculus. In Appendix, there is a summary of trigonometry to solve basic kinematics problems [53].

3.3.1 Example forward kinematics and inverse kinematics manipulator for 2 DOF

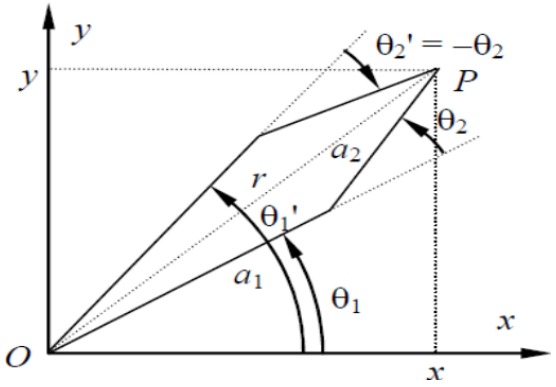
To demonstrate a kinematic solution, in this subsection we have an example of RR manipulators, that is, with 2 DOF. In Figure 3.17a, it is possible to identify the illustration of this model. The first step is to perform the identification of each joint (J_1 and J_2) and check the angles that each joint may form (θ_1 and θ_2), then the tool movement (In the example being analyzed in the vertical plane) [53].



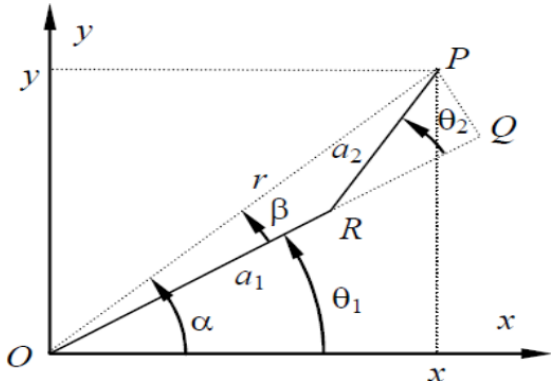
(a) RR manipulator in vertical plane movement.



(b) Trigonometric demarcations of the RR Manipulator.



(c) Possible solutions for inverse kinetics with $\pm\theta_2$ values.



(d) Demarcations of angles α and β to get θ_1.

Figure 3.17: Example analysis for RR manipulator. Image adapted [53].

Then, the equations of forward kinematics can be obtained by applying trigonometry to the triangles formed by joints and links, forming Equation 3.1 [53]:

$$\begin{aligned} x &= a_1 \cos\theta_1 + a_2 \cos(\theta_1 + \theta_2) \\ y &= a_1 \sin\theta_1 + a_2 \sin(\theta_1 + \theta_2) \end{aligned} \tag{3.1}$$

Already for the inverse kinematics equations it is necessary to carry out the inverse process. This means that one must obtain the position of the tool in space and then find the angles of the joints. Figure 3.17b assists in obtaining this position of the tool. Through Equation 3.2 is the inverse kinematics [53]:

$$\begin{aligned} r^2 &= x^2 + y^2 \\ r^2 &= (a_1 \cos\theta_1 + a_2 \cos(\theta_1 + \theta_2))^2 + (a_1 \sin\theta_1 + a_2 \sin(\theta_1 + \theta_2))^2 \\ r^2 &= a_1^2 + a_2^2 + 2a_1 a_2 \left[\cos\theta_1 \cos(\theta_1 + \theta_2) + \sin\theta_1 \sin(\theta_1 + \theta_2)\right] \end{aligned} \tag{3.2}$$

Applying the trigonometric relation $\cos(a + b)$ and isolating $\cos\theta_2$:

$$\begin{aligned} x^2 + y^2 &= a_1^2 + a_2^2 + 2a_1 a_2 \cos\theta_2 \\ \cos\theta_2 &= \frac{x^2 + y^2 - a_1^2 - a_2^2}{2a_1 a_2} \\ \theta_2 &= \pm \arccos\left(\frac{x^2 + y^2 - a_1^2 - a_2^2}{2a_1 a_2}\right) \end{aligned} \tag{3.3}$$

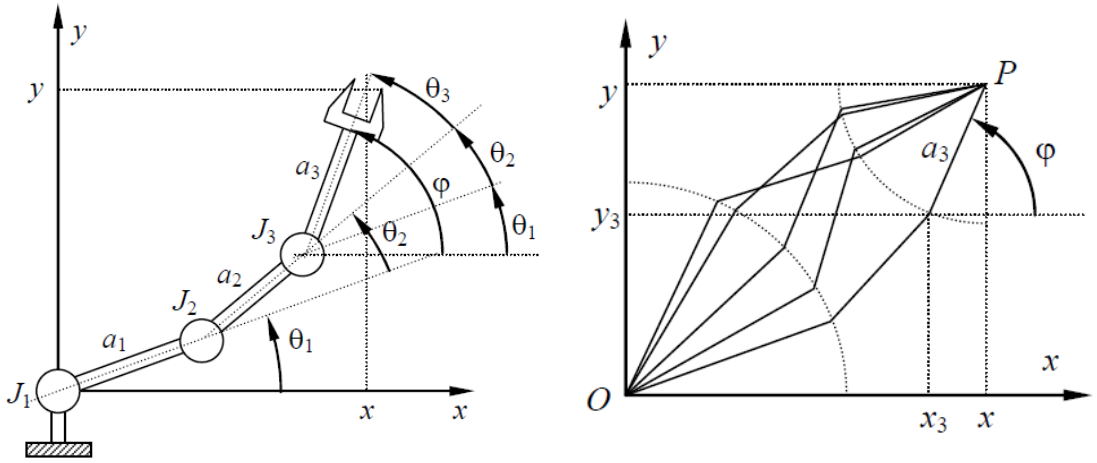
The "±" sign indicates that there are two motion solutions (Figure 3.17c) for the robot to reach the point chosen with the tool. The choice of which motion to perform can be made during system programming [53].

To obtain θ_1, simply perform the procedure of the tangent of the difference between the angles indicated in Figure 3.17d [53]:

$$\begin{aligned} \tan\beta &= \frac{a_2 \sin\theta - 2}{a_1 + a_2 \cos\theta_2} \\ \tan\alpha &= \frac{y}{x} \\ \tan\theta_1 = \tan(\alpha - \beta) &= \frac{\left(\frac{x}{y} - \frac{a_2 \sin\theta_2}{a_1 + a_2 \cos\theta_2}\right)}{\left(1 + \frac{x}{y}\frac{a_2 \sin\theta_2}{a_1 + a_2 \cos\theta_2}\right)} \\ \tan\theta_1 &= \frac{y(a_1 + a_2 \cos\theta_2) - x a_2 \sin\theta_2}{x(a_1 + a_2 \cos\theta_2) - y a_2 \sin\theta_2} \end{aligned} \tag{3.4}$$

3.3.2 Example forward kinematics and inverse kinematics manipulator for 3 DOF

In this subsection we have an example RRR manipulator, that is, with 3 DOF. Performing movement in a vertical plane and the last link with orientation in relation to the horizontal plane, forming the angle φ. In Figure 3.18, it is possible to identify the illustration of this model [53].



(a) RRR manipulator in vertical plane movement.

(b) Lines of possible moves to reach the point P.

Figure 3.18: Example analysis for RRR manipulator. Image adapted [53].

In the same way as in the example of Subsection 3.3.1, the equations for forward kinematics are obtained by adding the projections of the joints in the Cartesian axes, resulting in Equation 3.5:

$$\begin{aligned} x &= a_1 \cos\theta_1 + a_2 \cos(\theta_1 + \theta_2) + a_3 \cos(\theta_1 + \theta_2 + \theta_3) \\ y &= a_1 \sin\theta_1 + a_2 \sin(\theta_1 + \theta_2) + a_3 \sin(\theta_1 + \theta_2 + \theta_3) \end{aligned} \tag{3.5}$$

With the increase of one degree of freedom, the inverse kinematics formulation has infinite solutions. That is, the robotic arm can reach the point P with numerous movements, Figure 3.18b shows some of these possible movements. By fixing the orientation of the last link in relation to the horizontal axis, one can obtain solutions that satisfy the

inverse kinematics equations. For this, it is necessary to know the position of the tool at x, y and the angle φ and thus obtain the position of J_3 [53]:

$$\begin{aligned} x_3 &= x - a_3 \cos\varphi \\ y_3 &= y - a_3 \cos\varphi \end{aligned} \tag{3.6}$$

With Equation 3.6, the inverse kinematics analysis of the RRR manipulator is identical to the inverse kinematics analysis of the RR manipulator (determine θ_1 and θ_2), thus substituting the values of x and y of Equations 3.3 and 3.4 for the values of x_3 e y_3 of Equation 3.6, has [53]:

$$\theta_2 = \pm \arccos\left[\frac{(x - a_3\cos\varphi)^2 + (y - a_3\sin\varphi)^2 - a_1^2 a_2^2}{2a_1 a_2}\right] \tag{3.7}$$

Then θ_1:

$$\theta_1 = \arctan\left[\frac{(y - a_3\sin\varphi)(a_1 + a_2\cos\theta_2) - (x - a_3\cos\varphi)a_2\sin\theta_2}{(x - a_3\cos\varphi)(a_1 + a_2\cos\theta_2) + (y - a_3\sin\varphi)a_2\sin\theta_2}\right] \tag{3.8}$$

With the aid of Figure 3.18a, it is noted that φ is the sum of θ_1, θ_2 and θ_3. Then [53]:

$$\theta_3 = \varphi - \theta_1 - \theta_2 \tag{3.9}$$

3.4 Path planning

In the previous sections the solutions for the problems of direct and inverse kinematics are identified, however, the possibility of collision movements between the robot body itself and the objects in the work space is not considered. This section addresses the theme of planning paths to move the robotic arm, assuming there is a starting and ending point.

The line in the space where the manipulator tool moves from the starting point to the end point is called the path [54]. The trajectory planning calculates a function $q^d(t)$ that completely determines the movement of all joints of the robot as it traverses the path between the two points. This path is provided according to the geometric description of

the points, in other words, the best possible route [83].

The task of finding a path between points may seem simple, but it is a difficult problem to compute. The computational complexity increases exponentially with the number of DOF. So one of the possible solutions is to perform path planning as a search problem [83].

The algorithms presented perform a gradient descent search to find a route between the points. Even so, these algorithms can get "lost" and random searches can prevent these failures [83].

Library and algorithms for planning

Different methods of path planning can be explored in an application of robotic manipulators such as: Exact and close cell decomposition, control-based methods, potential fields, random planning, and sampling-based movements.

An excellent library of algorithms for planning is Open Motion Planning Library (OMPL), because in its composition there are several algorithms for calculating trajectory [85]. The OMPL planner is specializes in the planning of movements based on free space sampling, using the concept of robot state sampling in free space. Resulting in quick and effective responses to planning queries [86].

In other words, the sampling process calculates a uniform set of possible movements in free spaces to perform the motion and connect them in order to fit the path between two points. [86]. In its official page, it is possible to obtain information about the glider documentation, tutorials and interact with the community of collaborators, by the service of doubts and answers [87].

3.5 Software for manipulators

Unlike programming for computers, robotics there is still no universal programming language that meets all models of the area. This is due to the fact that some manufacturers have their own programming languages. However some software seek to support the most

used robot models, the advantage is in the execution or simulation of a system that can be applied to any supported robotic model. This section introduces some of these platforms.

3.5.1 Java library dLife

The dLife is an open source Java library that supports teaching and research in artificial intelligence, artificial life and robotics. The library includes packages for neural networks, algorithms for genetics, reinforcement learning, robotics, and computer vision. The dLife software is suitable for developmental and robotic work. [88].

It can be installed on Windows, Linux and Mac OS operating systems, so it supports educational and research robots. Supported robot models can be directly observed and controlled using `ControlCenter`, which is the central module. Java programs that use the dLife API can control robots both in `ControlCenter` and in stand-alone Java programs. The dLife platform also interacts with robot simulation systems, such as the Gazebo [88].

3.5.2 OpenRDK

OpenRDK is software focused on the rapid development of robotic systems. It is designed according to the requests of users, which has generated a success in applications with robots of different models. It is open source, because its developers believe it is useful for third parties [89]. The software meets the needs of researchers whose goal is to create innovative algorithms for complex, robotic and mobile systems. Helping you not to spend more time than necessary on the research infrastructure [90]. It can be installed on Linux and Mac OS operating systems and its main entity is a so-called `agent` process. The robotic application is made by modules, which can be loaded and started dynamically after the `agent` process is running [91].

3.5.3 OROCOS

The development project of general purpose robot control software has become the main focus of Open Robot Control Software (OROCOS). Always seeking the objectives: to

develop a platform with open source license, flexibility and quality [92].

It has components for kinematics, dynamics, planning, detection, control, hardware interface and others. Therefore it is a software that can be used dynamically, offering services independent of the programming language [92]. Real-time motion control is based on distributable and configurable components. Therefore, its real-time core provides a generic control structure for easy-to-customize installations [93].

The OROCOS project is concerned with the levels of its users, because it is possible to find in the official website some tutorials that have basic, intermediate and advanced modules [94]. It has great portability, as it can be installed on Windows, Linux and Mac OS operating systems [95].

Recently OROCOS has made advances in its design and implementation, with functionalities for real time movement control of mobile robots and manipulators. These advances can be obtained with OROCOS `Toolchain`, the main tool for robotic applications in the platform [94].

3.5.4 RDS

The Robotics Developer Studio (RDS) was launched by Microsoft with the goal of providing a software standard for the control of industrial robots. In order to provide some sets for technological solutions, such as common problems in robotic development [96].

Although its development has a relatively short period of time, RDS appears as a tool with great potential to program robots in the Windows operating system. What makes it one of the softwares responsible for the emergence of a common pattern in the field of robotics [97].

The interaction of RDS with robots happens through various software services, similar to Web services. These services facilitate the real-time application between the modules and hardware components or subsystems of the robot [96]. It is also possible to perform simulation and pre-programming, which can help avoid failures and identify possible problems before applying the control to robots [98].

Programming in the RDS environment can be done through two options: with C# (C Sharp) in Microsoft Visual Studio or with Visual Programming Language (VPL). The first option may be a bit difficult for some developers, because it takes a little more C# language knowledge. The second option provides a graphical interface for the developer to program, which resembles a flowchart [98].

To help developers, the official website features tutorials and documentation that demonstrate how to run applications. The content goes from the simple "Hello Robot" to more complex applications, like using depth sensors [94].

3.5.5 ROS

The development of software for general use in robotic applications is a difficult task, as has been mentioned in previous sections. However Robotic Operating System (ROS) was developed, from the beginning, with the focus of serving the simplest possible applications in robots. With a vast collection of tools and library, this software is already consolidated in the robotics industry [99]. To describe this software in detail, a good way is to designate a distinct section for this. Section 3.6 is in charge of this task.

3.5.6 V-REP

This Virtual Robot Experimentation Platform (V-REP) software is a framework that meets many requirements for extreme quality robotic simulation. In addition to offering the traditional approaches found in other simulators, the V-REP has several additional codes embedded in its repository [100].

The robot simulator V-REP is developed in a distributed control architecture, that is, each model can be controlled individually. Controls can be via: programming codes, remote API client, and custom solutions. Therefore this software is ideal for simulations of different robotic models [100].

The codes for control can be written in languages: C / C ++, Python, Java, Lua, Matlab or Octave. Has the ability to develop simulations: simple, factory automation,

education related to robotics, security checking and other purposes. It is also possible to simulate various sensor applications, kinematics, collisions, distance calculations and motion planning [100]. The V-REP can be installed on Windows, Linux and Mac OS operating systems. On its official website it is possible to obtain tutorials and information regarding the licenses of each available version of the software [101].

3.6 ROS Software

Problems that seem trivial to humans can be difficult for robotic applications, such as interacting with the environment around them. ROS has a wide range of uses in developing programs for robots that make these applications easier. In this Section, the ROS concept is discussed, indicated in the subsections of the software's emergence, its architecture and the essential tools.

3.6.1 History of ROS

The ROS was developed by a group of collaborators who in the mid-2000s analyzed the lack of such software for the robotics research community. Some projects have boosted ROS creation, such as STanford AI Robot (STAIR) and Personal Robots (PR). In 2007 Willow Garage provided extensive support and ROS started using an open source Berkeley Software Distribution (BSD) license, so the software became license-free widely used on Linux operating systems [102].

The ROS is software that contains a wide range of use in developing programs for robots. Problems that seem trivial to humans can be difficult for robotic application, as they interact with the environment around them. According to [103], The philosophy is to make a piece of software that can work on other robots making small changes to the code. What we get with this idea is to create features that can be shared and used in other robots without much effort, so that we do not reinvent the wheel. The Figure 3.19 shows the official logo.

Figure 3.19: Official ROS software logo [99].

Therefore, ROS makes interactions between the functionalities of a robot (sensors, locomotion, vision, navigation and location) with the aid of libraries and services, thus facilitating robotic application, as indicated in Figure 3.20.

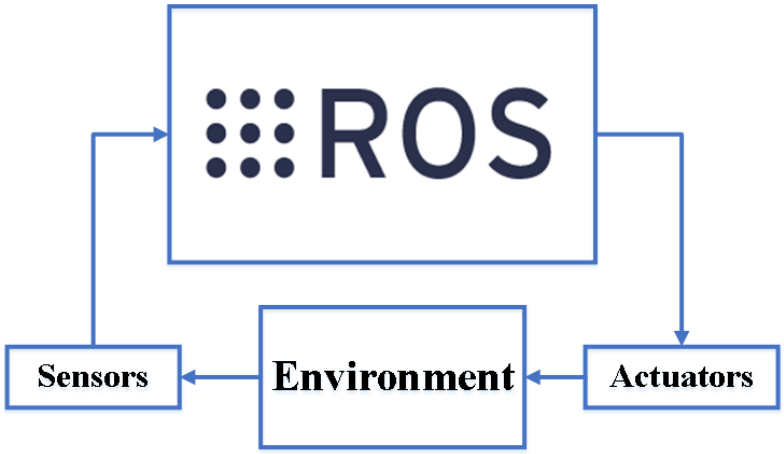


Figure 3.20: Operating Flowchart ROS software. Image adapted [104].

Throughout the world, this development platform has been growing. Since the main idea of the ROS project is the creation and sharing of *code samples*, that is, models or examples of pre-defined code. The result of all this is the day to day increase of the devices supported in this program [105].

With a simple installation and guided tutorials, the user can access the official website and find different versions for different platforms [99]. The user can also find the entire communication and sharing community, which will be better explained in the next subsection 3.6.2.

3.6.2 Structure of ROS

It may seem simple how ROS performs robotic applications, with its installation facilities and code shares. However behind all this, there are some processes working together. For a better understanding, the ROS is divided into three levels. These are: File System, Computer Graphics and the ROS Community.

The entire structure functions as an operating system, simply because there are specific folders for each process within the file system, as in Figure 3.21. Each folder contains files that will have different functions and operations [103].

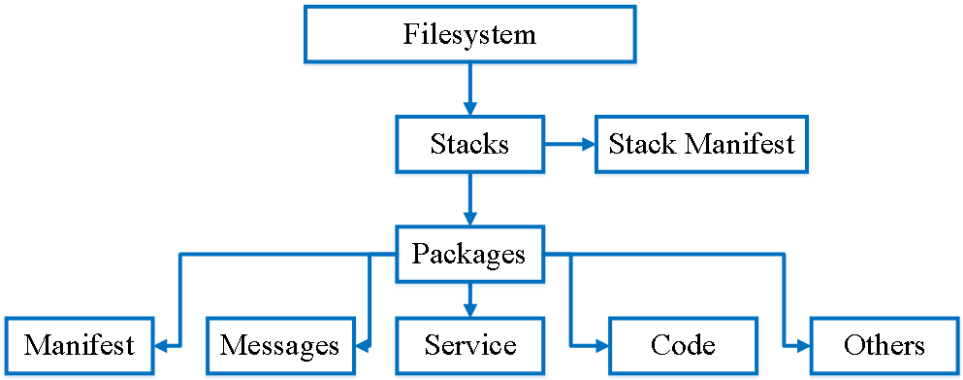


Figure 3.21: Flowchart concept of ROS file system structure. Image adapted [103].

ROS packages form the most basic part of the ROS program. Its main function is to contain folders with basic settings, libraries, run-time processes and other functions for the application's operation. The manifest file exists to store package creation information, since the message files send information from one process to another. To work with request-response type interactions, services are used. Other files can still be found in some ROS packages, as each developer may need more tools. And finally, the code files that serve to execute all the necessary processes of an application [103].

There is the possibility to create several packages and to make an interaction between them. For this, there are batteries. In current versions they are called meta-packages. It is very simple to implement batteries in ROS, it can be created through a command or manually. To make it confusing at the time of sharing a package, there is the stack file that contains information about creating and running the stack, the stack manifest [85].

It may seem a bit unnecessary how many folders to separate functions, however it may be noted that the ROS file structure is very clear and objective, presenting a simplicity allied to the organization. Such organization is evidenced in the computational sector of the program [105].

A network of processes is performed to obtain the computational part of the program, as in Figure 3.22. All processes connected to this network can access data from any other processes connected to the network as well. The main concepts are: Nodes, Master, Parameter Server, Messages, Topics, Services and Bags [105].

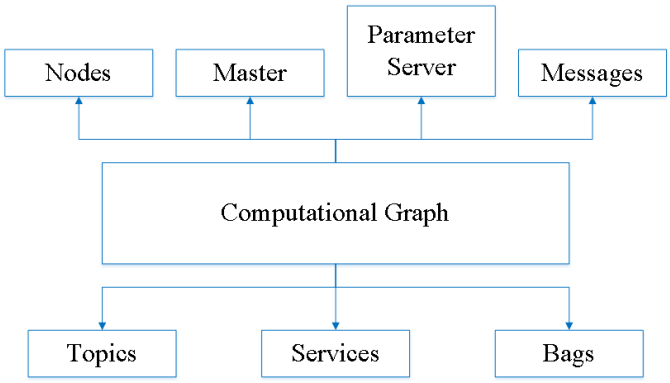


Figure 3.22: Flowchart of the structure concept of the ROS computational graph. Image adapted [85].

Nodes are processes where computation is performed. Typically, a system will have many nodes to control different functions. Sometimes it is better to have many nodes that provide only a single functionality rather than a large node that does everything in the system. Nodes are written with a library of ROS clients, for example, *roscpp* or *rospy* [85].

For these nodes to communicate there is the ROS Master. To do a graphical computation it is necessary to execute the ROS Master to be able to interact one node with another, otherwise the nodes will not be found. This type of communication happens using some message patterns (floating point, Boolean and others). In addition, ROS Master contains the parameter server which helps in the configuration of the nodes [85].

With all this network of information, it is necessary to define some routes of communication between the nodes, otherwise it can confuse the communication network. Usually this works by using the topics, as in Figure 3.23, where a node publishes a message in one topic and another node can subscribe to the same topic. A topic is unable to answer a request or a response from a node. Then to make a request and answer, the services are used.

Figure 3.23: Example of a communication between two nodes using a topic. Nodes in blue and green, the communicating through topic, in red. [106].

The last level is the sharing of resources between the ROS community. This level is extremely important for the distribution of ideas and help among developers. In the ROS community the most outstanding concepts are: ROS Wiki and ROS Answers. The first one is composed of several tutorials, ranging from the installation to the execution part of an application. ROS Answers is a question and answer forum where the user can get answers from a moderator who has advanced knowledge in ROS. Even within the community some sectors can be cultivated, such as: Distribution (form the possible installation versions of ROS), Repositories (code network developed among programmers), Blog (composed of news, photos and videos) [103].

Although it seems a bit confusing, it is notorious that all these divisions and ramifications of each level are justified to better organize all software. It would be a lot more confusing if it was just a folder with all files merged. It takes little time to use ROS to note that this method of organization is essential for a robotic application.

3.7 Point Cloud

One of the great challenges in the industry is to indicate to the control system of a manipulator the environment around it. This process can be called by some perception authors. In this section it will be indicated how the Point Cloud system helps in the process of perception. To perform a robotic application in environments that are not structured, it is necessary to indicate to the control system the situation of the environment. That is, it is necessary to make the perception of the real world [107].

Therefore a Point Cloud is a data structure used to represent a set of multidimensional points, commonly used to design three-dimensional data. In a 3D Point Cloud, one usually seeks to represent geometric coordinates X, Y, and Z. If the cloud has color, this representation becomes 4D [107]. The Point Cloud can be obtained by hardware sensors, such as: stereo cameras, 3D scanners or even can be generated from a program that simulates the points [108]. There are different developments in the Point Cloud business, where each of the projects seeks different objectives. The most common projects are:

- **PCL**: Project was designed to create large-scale Point Cloud processing to meet the demand for robotic applications [108].
- **Euclideon**: This technology is able to provide an unlimited amount of Point Cloud rendering, without depending on the hardware [109];
- **MeshLab**: Open source system for 3D triangular mesh processes [110];
- **CloudCompare**: High Density 3D Point Cloud processing software, originally designed to compare two sharp and dense points or between a point cloud [111].

3.8 Sensors RGB-D

The acquisition of the work environment description for robotic applications is a challenging task of great importance, as already mentioned in Section 3.8. The challenge increases when one seeks to accomplish these tasks in three-dimensional planes, which means that

the control system must have the ability to acquire the working environment of the robot throughout its workload. Some factors contribute to achieving this data acquisition, so in this section we present the basic concepts of a depth sensor and the sensor used to perform the work.

For the data acquisition process of the robot's work environment, depth sensors are applied in order to record the perspective of the environment. With the data provided by these types of sensors it is possible to measure distance between two points of interest [1]. There are many devices on the market with this depth register feature, such as Red, Green and Blue (RGB)-Depth.

The sensor RGB-D is a set of three different sensors: a camera RGB, an infrared camera and an infrared projector [112]. The set of these three sensors design a pattern of points in the environment, this pattern is reflected when faced with some obstacle. The end result of this process is the formation of a cloud of points [113].

The union between depth data and data RGB provides solutions to a number of problems, such as detecting objects or people. The great advantage is in the segmentation of objects based on depth information, since the identification does not depend on the light of the environment or the texture of the object [114]. Another factor is that in recent years the RGB-D sensors have attracted great attention in 3D modeling due to the low cost [115].

If well-configured, the RGB-D sensors can provide information for the robot control system to perform unsupervised tasks with a high degree of complexity. With the data of this sensors one can assemble maps of environments, create models of objects both two-dimensional and three-dimensional, avoid obstacles during the movement of the robot and other resources. [116].

Chapter 4

System methodology

In this chapter 4, will be analyze in detail all the solutions adopted, taking into account the specific case of this work. All the procedures and considerations will be presented, starting with the problem facing the industry and the proposed solution, seen in Section 4.1. Then, in Section 4.2 the paths to reach the solution are defined, that is, the methods for the development of the system. Finally, Section 4.3 indicates the architecture of the system.

4.1 Problem and Solution

In this section are brief summaries of the technological steps that the industry has followed since its First Revolution to the present day. At the end it is demonstrated the prospects that the industry has for the future, one of these tasks being the fuel for the proposed work.

The term Industrial Revolution was used for the first time by French analysts around 1820, where it was tried to evidence the great transformation that England obtained in the economy from the year 1780 [117]. There are some divergences among historians about the expression used to define the process of revolution, some believe that by using the word “revolution” it is indicated that the industry has returned to its starting point. However the expression became conventional over the years.

The great transformation in the method of production is in the substitution of some manual steps for steps made by machines, the maquinofaturas. These machines were fueled by coal, called steam engines (Figure 4.1), and interpreted the movements of workers cite Fabio, thus replacing artisanal production. Therefore man becomes the third element in production, producing more and more and more [118]. The importance of the First Industrial Revolution was not only in the production processes and consumer articles, it also reached: agricultural techniques, transportation means, economy and modified society [119].

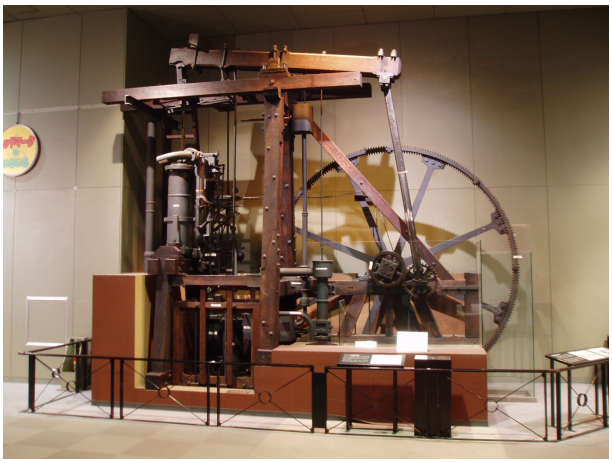


Figure 4.1: Steam engine created by James Watt [120].

Even with the great advance in productive methods, the industry still had to reinvent itself. Aiming the search for energy sources and new materials of production, by the 1870s a new industrial revolution began. Therefore, the fundamental basis for the Second Industrial Revolution was the development in new energy sources, such as: electricity and oil. The new energy sources have expanded the industrial sectors, they are: electrical, chemical and steel industry. In addition to improving existing sectors that were still expanding [119].

With the emergence of electric energy, new products appeared in the market and boosted consumption: the replacement of the lighting from oil and gas, the electric lamp, the creation of the electric motor, alternating current or continuous, allowed the dissemination of domestic utilities, and the development of the combustion engine [121].

In the Figure 4.2, it can be noted how the Second Industrial Revolution changed society, its greatest legacy was to make the machine lighter and more dynamic. Unlike the First Industrial Revolution, there were machines that were fixed and almost impossible to transport.

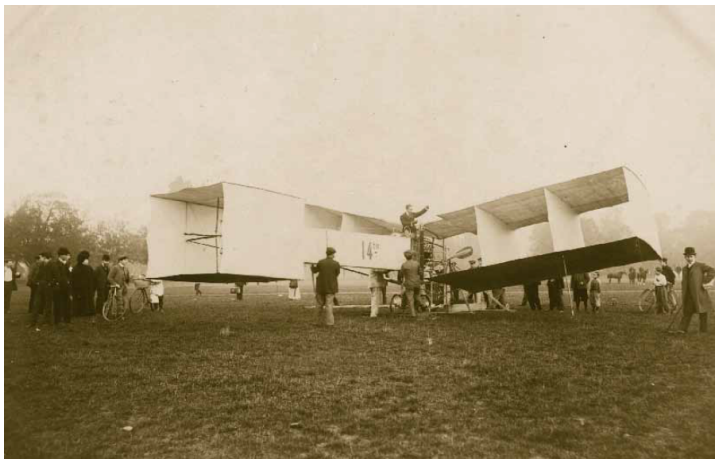


Figure 4.2: Airplane 14 bis created by Alberto Santos Dumont [121].

Based on a productive dynamics with sophisticated technology, the Third Industrial Revolution arises in the middle of the XX century. The new transformation of the industry happens with the insertion of computing, communication worldwide and robotics [122]. The new generation of machines joined the mechanics developed in the past generations with the new technology developed, microelectronics. The use of microelectronics enabled a set of a new industrial sector, automation in the production line [123].

Industrial automation is a technology that performs operation and production control, with the use of mechanical, electronic and computer based systems. This technology can be seen in transfer lines, mechanized assembly machines, machine tools with numerical commands and robots [49]. With new equipment in the industrial sector, the workforce focuses on production, management and engineering [123], so the worker loses his space on the production line, as indicated by Figure 4.3, for new machines and starts to operate them. These machines have mechanical movements controlled by programming techniques, the robots. Because they are programmable, robots can move their arm sequentially to perform tasks [49], details of the robots are covered in Section 3.1.

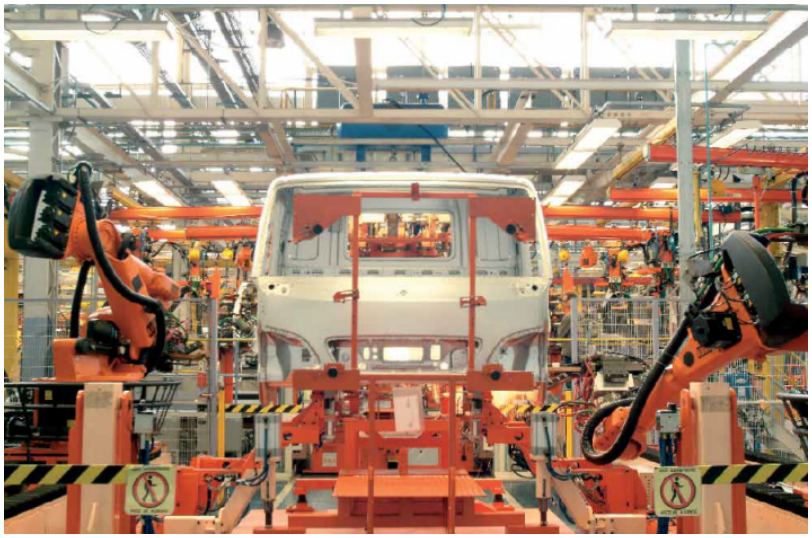


Figure 4.3: Mercedes-Benz production line with the help of industrial automation [121].

The technologies of hardware, software and communications networks are being developed independently, it is notorious the beginning of a rupture with the technology developed in the Third Industrial Revolution. In a sophisticated and integrated way, this technology begins to change society and the global economy [124]. This is just one of the justifications for claiming that the Fourth Industrial Revolution is beginning to thrive in today's technological landscape.

In 2011, in Germany, there was the creation of the term *Industrie* 4.0 and how this new revolution will affect the global market. Another subject questioned during the discussion is what the legacy and research needed for such a revolution would be [125].

The perspective points to emerging technologies that are interconnected and capable of creating a totally intelligent production line, that is, creating an intelligent industry. Meanwhile, Industry 4.0 can also reach areas such as: nanotechnology, gene sequencing, renewable energy and even quantum computing [124]. Many challenges will be put to the test, such as merging something physical with the virtual world. It is possible to make this fusion through CPS. These systems can perform the monitoring and control of physical environments, using collaborative computational elements [126].

The CPS can still be used in situations where humans and robots can share work environments in this way can help each other. Then there is a need for new robotic

models capable of interacting with humans, estimating and avoiding collisions, capable of making decisions and other skills. This fact can be observed in the manipulation of objects by robotic manipulators, which is on the rise in the industrial market..

Based on a robotic manipulator, it is intended to develop a security system capable of avoiding the collision between manipulator and objects around it. That is, have an anthropomorphic manipulator and endow it with the ability not to collide with objects in your work environment.

To this end, the system to be developed will be applied first in simulations, and thus avoid possible failures and risks. If it demonstrates adequate performance, it can be applied in the real robotic arms environment, that is, if the system does not trace trajectories between the obstacles during the simulations. The Figure 4.4 briefly illustrates the system's proposal to avoid obstacles during the trajectory of a real or virtual manipulator.

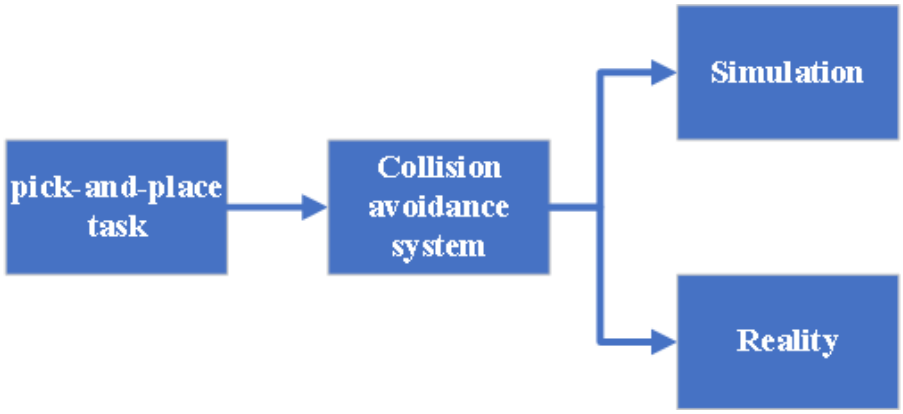


Figure 4.4: Illustration that sums up the proposal for the industry problem in avoiding collisions in the path of an industrial robot.

In order to reach this proposal, some decisions must be taken regarding the tools used. The study carried out in the previous chapter (Chapter 3) contributes to the choice of the procedures followed and the reasons will be presented in Section 4.2.

4.2 Solutions adopted

This section has the solutions adopted to achieve the proposal to develop a system capable of avoiding collisions in obstacles inserted in the work environment of an anthropomorphic manipulator. The solutions are adopted according to the tools presented in Chapter 3. Therefore the solutions for this work range from choosing software to laboratory simulation.

The starting point to develop the proposed system is based on the choice of software, or the operating system of brings all processes. According to the previous study, done in Section 3.5, there are good software capable of serving as a basis for system development. However, two of these have excellent libraries and simulation power, they are: ROS and V-REP. In the previous study, V-REP presented a great capacity in robotic applications and programming versatility; providing all the resources in excellent simulations.

On the other hand, ROS besides offering the same features of the previous software, has the possibility of applying the project developed in real robots. Finally, both are enabled to serve as base software, but in case the developed system has applicability in real robots, ROS is the best choice. In the figure 4.5, the representation of the layers of the operating systems used in this work is illustrated.

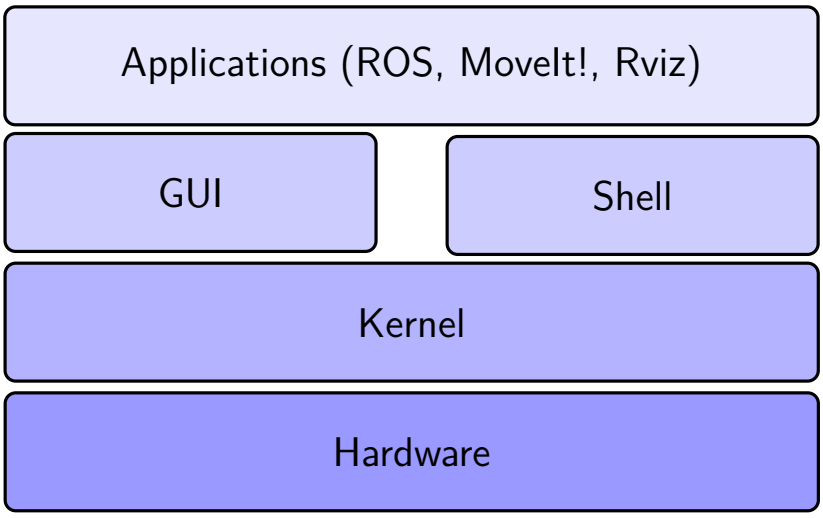


Figure 4.5: Illustration of architecture layers of software installed in this work.

Once the base software has been chosen to develop the system, it is necessary to choose the manipulator model. This will serve as a model for simulations and tests during avoidance system applications. This handler was chosen because it has good compatibility with the packages ROS. In this respect, in the subsection 4.2.1 is indicated the model and some characteristics of the collaborative robot UR5, applied in the simulation phase of the system.

As for the fact of the simulations, in the Subsection 4.2.2 the visualization tool is presented in ROS, the Rviz. With this tool it is possible to check for possible failures in the implementation of the robot model used (UR5), it is also possible to simulate scenarios such as the addition of boxes. And in the Subsection 4.2.3 the robotic manipulation tool is exposed in ROS, the MoveIt!. This other tool allows performing: motion planning, manipulation, 3D perception, kinematics, control and navigation.

According to the study pointed out in Section 3.3, to perform the movements it is necessary to produce techniques to identify the positions of certain points of the arm. The algorithms chosen to proceed with these calculations were: RRT and PRM. Both are widely employed in the planning field, making the results obtained from the system become validated. Another advantage of using these two algorithms is that they are contained in the OMPL library, which in turn is also inside the MoveIt! Database. In the Subsection 4.2.4 there is a demonstration of how these algorithms work.

For the perception of the work environment, the Kinect sensor was chosen. The choice was made by the ease of obtaining the product. In the laboratory premises of ESTiG, this sensor is easily found and made available for student use. As a consequence of this choice, more information about this sensor RGB-D is displayed in the subsection 4.2.5. It was also observed that the fixation position of the sensor is relevant because it works with infrared projection. This means that certain fixtures will generate shadows in the data obtained, that is, some object in the scenario may block the visualization of another object of interest. For this purpose, the fixation of the sensor was chosen in order to minimize these shadows. After some tests, it has been noticed that the best way to fix Kinect is to always return it to the desktop of the manipulator in question. In other words, it is

necessary to have knowledge of the range of the robot and position the sensor in parallel with this area. Thus, it is possible to guarantee that the generated Point Cloud will have few shadows, better capturing the environment.

With the acquisition of environmental data, it is necessary to treat this three-dimensional map. The ease of implementation with ROS and the layout of many tools and their tutorials, have chosen this library as the one chosen for this work, when it comes to dealing with the data provided by Kinect. Because of this, the Subsection 4.2.6 depicts the features that the Point Cloud Library (PCL) library is able to provide. For work, the library PCL was applied to remove some noise in the Point Cloud and also in the transformation of the Point Cloud into Octree maps.

4.2.1 UR5 manipulator

The UR5 is a collaborative manipulator robot that is gaining market space because it is low cost and recommended to automate light processing tasks such as collecting, depositing and testing. This subsection contains some information from the manufacturer itself.

The UR5 robot is easy to program, fast-paced and offers one of the industry's lowest return on investment time, as well as other collaborative members of the UR family [127]. The safety system has a certain stop time, this time is defined between the intervals of a fault occurrence (or any safety related function) until the complete stop of the UR5. This time must be set according to the application [127]. The following Table 4.1 represents an example of a stop time setting:

Tool speed limit	**Maximum stop time**
1.0 m/s	450 ms
1.5 m/s	500 ms
2.0 m/s	550 ms
2.5 m/s	600 ms
3.0 m/s	650 ms

Table 4.1: Table with safety limits according to the speed of movement of the tool [127].

Another safety device is the button for emergency stop, when this button is pressed all movements of the UR5 are interrupted. This feature should be used as a secondary protection device and may have more buttons according to the need of the application. As already mentioned, UR5 can be configured to work in different modes when detecting a person in your workspace. Here is a list of these modes and their specifications [127]:

- **Reduced mode**: This mode can be activated when the robot tool is positioned beyond a certain plane. This plane can be determined according to the desired configuration. By activating this mode the manipulator reduces some control parameters such as speed and acceleration [127];

- **Normal Mode**: in which mode the positioning of the manipulator tool is contained in the plane determined in the configuration [127];

- **Recovery mode**: this mode is activated when safety limits are violated, then the system restarts. In this mode it is possible to move the manipulator by means of free driving [127]. The limits for this mode are set forth in Table 4.2:

Security function with limitation	**Limits**
Joint speed	30 °/s
Tool speed	250 m/s
Tool strength	100 N
Impulse	10 kgm/s
Power	80 W

Table 4.2: Table with recovery mode security limits [127].

The UR5 workspace extends up to 85 cm from the base to the tool attachment point [127]. Therefore your workspace will have a cylindrical shape with respect to the base (as Section 3.1.2 indicates). Figure 4.6 shows an illustration of an example application of UR5. Because it is a model with versatility in applications, the UR5 can be fixed to different supports. And supported payload can reach up to 5 kg [127].

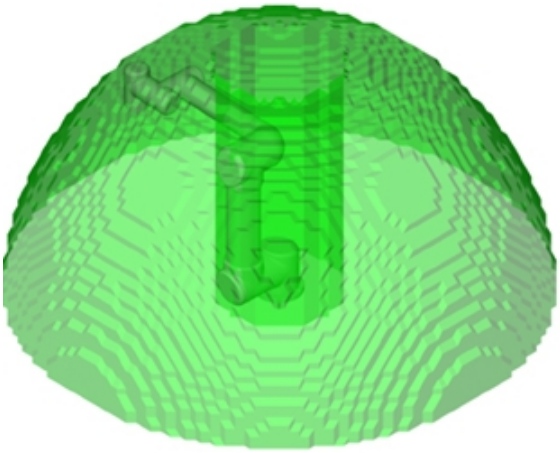


Figure 4.6: Illustration of an UR5 workspace [128].

The calibration, maintenance and repair of the UR5 must be performed by the manufacturer's representatives [127]. In Appendix D, is technical information in reduced format and further information can be obtained from the manufacturer's official website. The cost of an UR5 manipulator for industrial customers can reach the value of 23 900,00 EUR (+ IVA), on the other hand, for universities this value is 18 500,00 EUR. These values may change and have been indicated by a representative of Universal Robots in Portugal [88].

4.2.2 Rviz

The ROS has a large number of powerful tools to help the user and developer in the process of code debugging and to detect problems with hardware and software. This includes debugging facilities such as `log` messages as well as viewing and inspection capabilities, which allows the user to see what is happening in the system easily.

The Rviz is a 3D visualizer developed to make a virtual simulation of robotic models in ROS, as in Figure 4.7. This simulation is important to check for flaws in model designs and simple planning. This visualization tool depends on Unified Robot Description Format (URDF) and XML Macros (Xacro) files for information on robot descriptions, such as: size, joins, and so on [103].

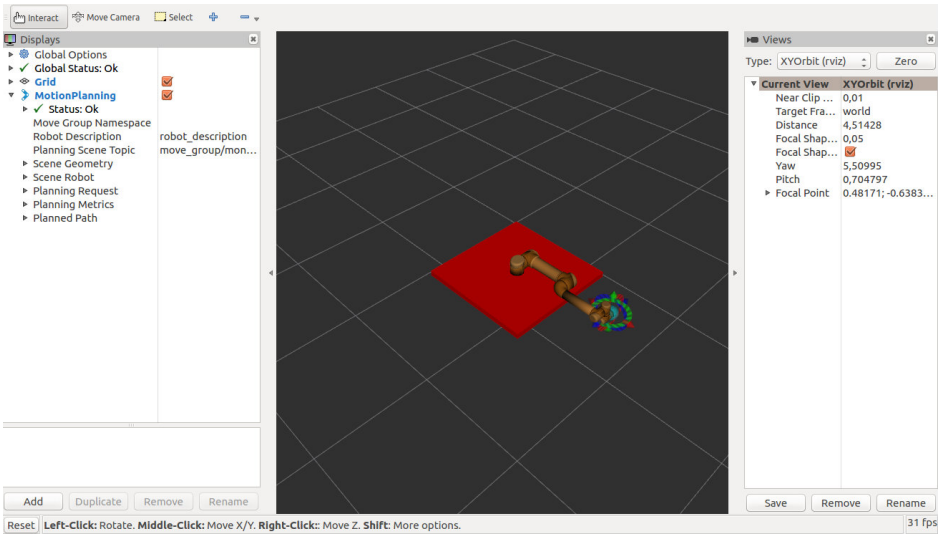


Figure 4.7: Example of a 3D visualization with a simulated robotic application.

During the simulation it is possible to create scenarios in the XYZ format with obstacles, to change positions of pose of a robot or to move them through a virtual "world". You can enter sensor data (such as Microsoft Kinect), change the source points of these sensors, or robot source points. This way it is confirmed that the application is ready to be implemented in an actual application. Avoiding possible problems in real robots [102].

4.2.3 MoveIt!

Another tool that assists developers and users in robotic applications in ROS is MoveIt!. This software contains libraries to perform: motion planning, manipulation, 3D perception, kinematics, control and navigation. In this subsection some of these features are presented, on the official MoveIt! website is possible to find more information and tutorials [129].

A good way to understand how this software works is to understand the basics of its architecture. The `move_group` is the central element in its architecture, as shown in Figure 4.8. The main idea is to define groups that perform actions and work together, with the interconnection of a central element [129]. The following topics indicate the features and characteristics of each of these groups:

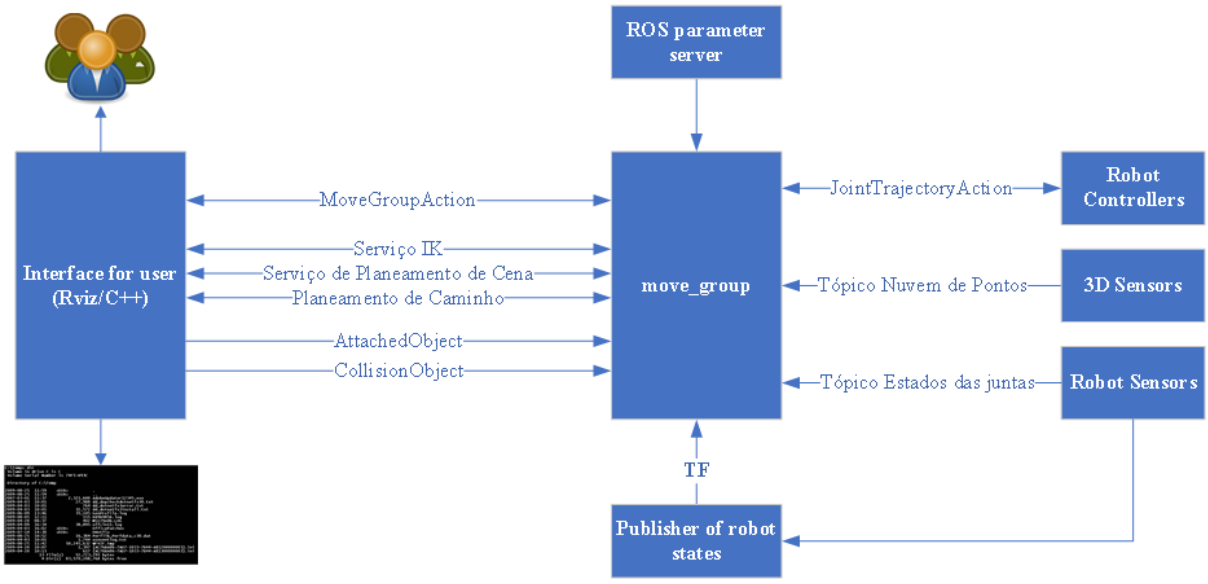


Figure 4.8: MoveIt! architecture diagram. Image adapted [129].

- **ROS parameter server**: The `move_group` search information in files with configuration parameters, these files have ROS standards and description languages, such as: YAML, URDF and Simulation Description Format (SDF). These are files that provide features of robotic applications such as: speed limits, joints, controllers, acceleration and other information [130];

- **Robot Controllers**: The `move_group` node communicates with the robot controllers using the ROS interface: JointTrajectoryAction. The robot must provide a server so that the output of the motion planning can be planned and executed on the robot's hardware (or simulator). Then the JointTrajectoryAction action interface is just a server client created inside the controller [130];

- **3D Sensors**: data from a 3D sensor can be inserted into `move_group` through topics [129];

- **Robot Sensors**: presents to the `move_group` the current state of the robot, in the format of ROS topics. Generally information on the current state of joints of a robot is transmitted by the topic `/joint_states` [85];

- **Publisher of robot states**: node responsible for publishing the TF information of the robot, since the `move_group` only reads the information. ROS uses the TF packets to represent all the coordinates of the state of a world [129];
- ***MoveGroupAction***: is a class that belongs to the ROS messaging package specific to MoveIt!, performing several actions between the `move_group` and the user interface. Such as: planning request, planning options, planning checks among others [129];
- **IK Service**: robotic application kinematics service [130];
- **Scene planning service**: service that serves to represent the world around the robot, including the robot itself [129];
- **Path planning**: Service that tells the `move_group` what movement the user would like to perform with the robotic application [129];
- **AttachedObject**: tool that attaches an object to any part of the robot, in this way the kinematic calculations count on the object to be attached. For manipulator robots, it is common to use this tool to attach objects to the tip of the tool [129];
- **CollisionObject**: tool that checks for possible collisions between the robot and some object in your workspace [129];
- **User interface**: the user can use MoveIt! to perform motions in robots by GUI, through Rviz (mentioned in Section 4.2.2). Or can run applications with C ++ or Python code [130].

The Figure 4.9 shows an example of a simulation through the interface of the MoveIt!:

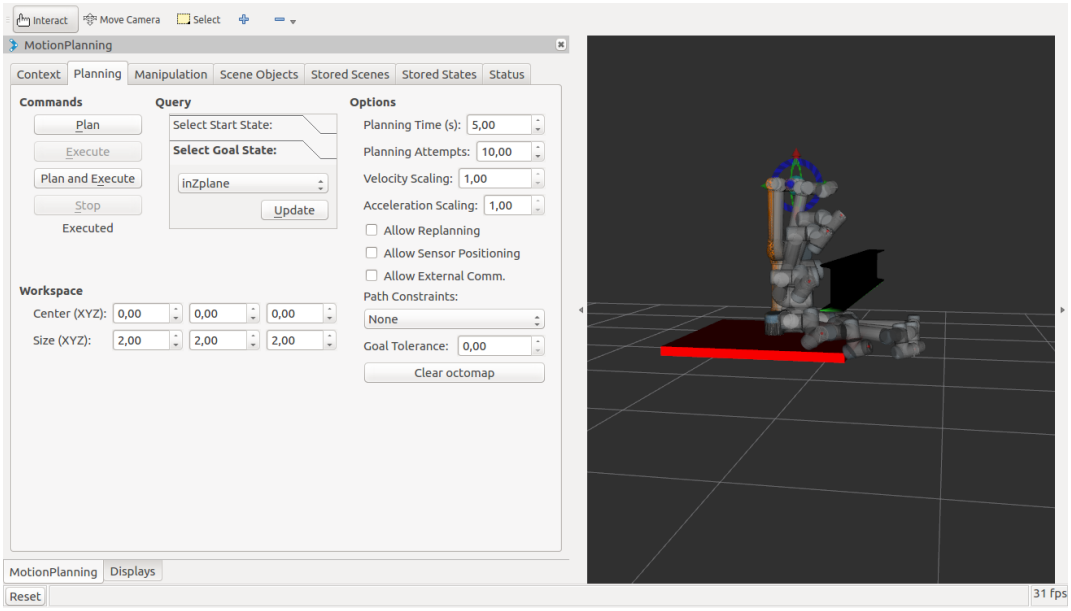


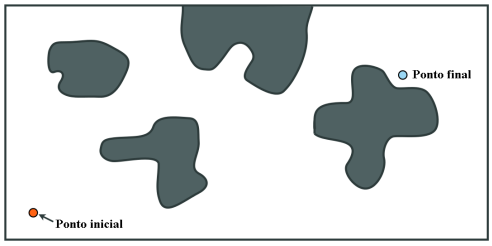
Figure 4.9: Example of a 3D visualization simulating robotic manipulator movement avoiding collisions with obstacle.

4.2.4 PRM and RRT Algorithms

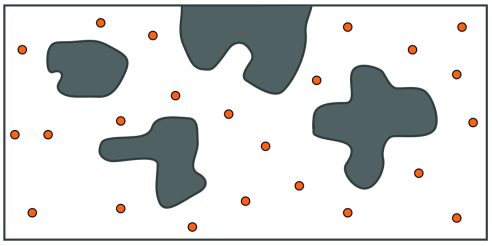
A widely used algorithm for planning is multiple queries of scripts created from the environment, known as PRM, these multiple scripts are random and based on sampling algorithms. These road maps are similar to a street map. The sequence of images shown in Figure 4.10 illustrates the fundamentals of PRM [86].

Another widely used algorithm is RRT, which performs free space sampling with structures that resemble a tree. The tree generated by this algorithm is composed of several branches and nodes, generated by random sampling of free spaces [131]. The sequence of images shown in Figure 4.11 is an example of how RRT acts.

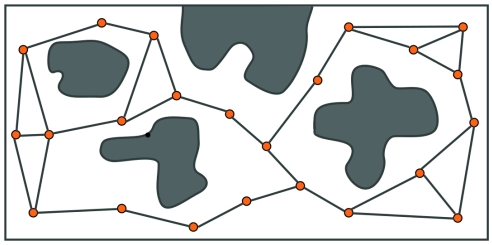
The two algorithms cited in this subsection are part of the OMPL planning library, which in turn is also included in MoveIt! (Subsection 4.2.3).



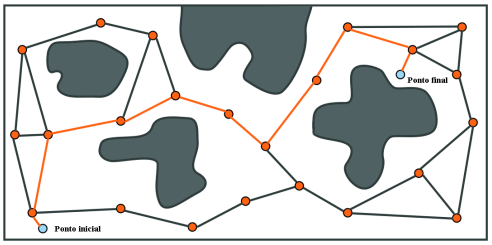
(a) The 2D workspace is the only point for the state of the robot.



(b) A possible set of uniform random samples.



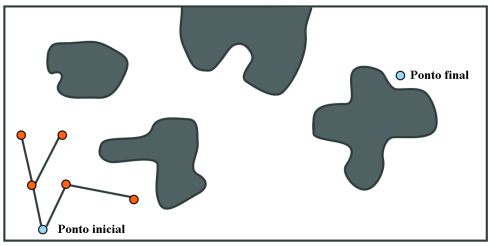
(c) Connect samples close to each other using the direct path in free space.



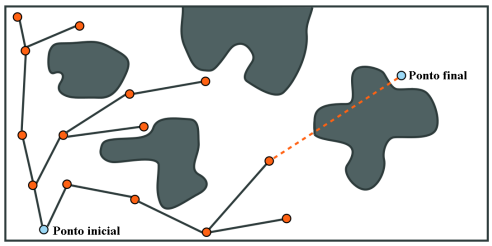
(d) The start and end points are connected through the route, and then the shortest route is found.

Figure 4.10: Example of how the PRM algorithm works in planning [86].

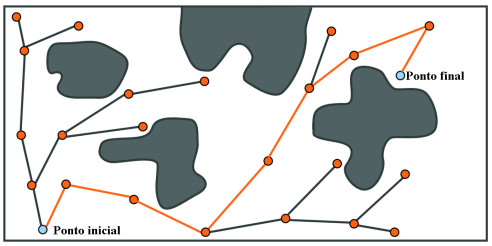
(a) The 2D workspace is the only point for the state of the robot.



(b) Random samples are connected forming the tree.



(c) The tree's lines skirted the first obstacle.



(d) The endpoint is connected to the tree, the path has been found.

Figure 4.11: Example of the operation steps of the RRT algorithm in planning [86].

4.2.5 Microsoft Kinect

With the proposal to replace video game controllers, Microsoft developed the Kinect device (Figure 4.12). Announced in November 2010, the device's main function was motion detection [132]. A revolution in the area of interaction with games has started, it is no longer necessary to use command, because the sensor captures movements and voice commands of the user [133].

Figure 4.12: Kinect sensor.

Shortly thereafter, some researchers noted the potential of the sensor. This technology could be exploited in other applications, not just in games. So the company decided to make the Software Development Kit (SDK) available for developers to use in their own applications [134]. Kinect has the concepts of RGB-D sensors, with its hardware linking: an infrared camera, an RGB camera, four microphones arranged along the sensor, an infrared projector and a motor to adjust the angle of view vertically [135]. Figure 4.13 shows the distribution of these elements.

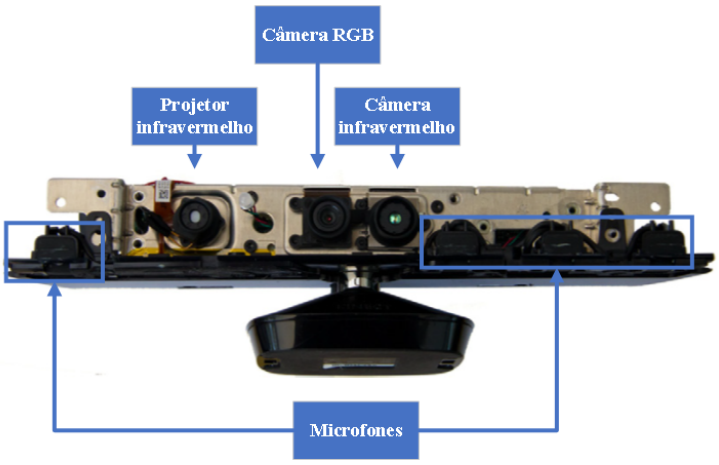


Figure 4.13: Kinect sensor open.

The technique of measuring how far something is consists of an infrared projector and a camera that can see the small dots reflected. These points generate a "depth map" of the environment in front of the sensor. This map is made and sent by USB to some connected host, analogous to a transmission of an image from some camera. However, instead of color information for each pixel in the image, the sensor sends distance data [135]. In Figure 4.14, it is possible to notice to the left side a real environment and to the right side the depth map generated by Kinect.

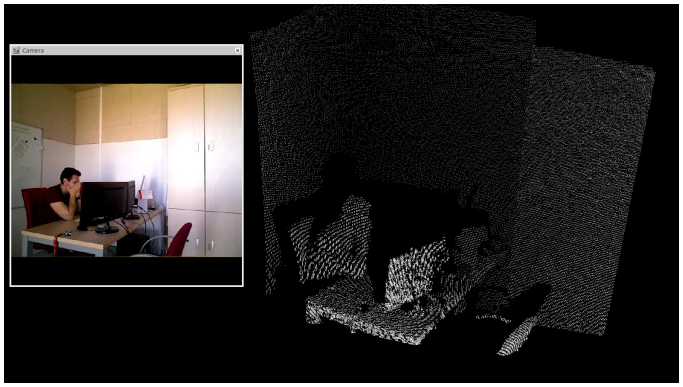


Figure 4.14: Example of a depth map generated by the Kinect sensor.

The sensor is based on optical lenses and has some limitations, but works well in the following parameters of Table 4.3 (all from the center of the Kinect) [132]:

Limits	**Values**
Horizontal viewing angle	57 °
Vertical viewing angle	43 °
Motor movement	-28 °à +28 °
Range of depth	40 cm à 800 cm
Distance for best results	120 cm à 400 cm
Temperature	5 °C à 35 °C

Table 4.3: Table with the operating limits of the Kinect sensor, data obtained [135].

4.2.6 *Point Cloud Library*

PCL is an open-source, large-scale development of image processing, whether in 2D or 3D. This tool contains a wide variety of algorithms to perform numerous jobs. These algorithms can be used to improve the quality of a Point Cloud, an example would be the use of filters to eliminate noise that affect image processing [107].

This library can be installed on several operating systems, that is multi-platform. Another important factor is the contribution of engineers and scientists from different organizations, distributed all over the world [108], this contributes to the development of the library in a way that suits the needs of each application. The algorithms that make up the PCL can be separated by smaller libraries [107]. In the following subsections a brief specification of each of these small libraries is presented.

Filters

During measurement, some data sets may have a shadow over other sets. This affects the estimation of characteristics in the generation of the Point Cloud. In other words, some points may compromise the generation of the Point Cloud. To solve this noise problem in measurements, the `pcl_filters` library was created. By means of statistical methods some of these noises can be filtered, as in Figure 4.15. These methods are based on average distances between the next generated points. When the analyzed points do not meet the requirements of the average distance, they are removed and not added to the data set of the Point Cloud generation [136].

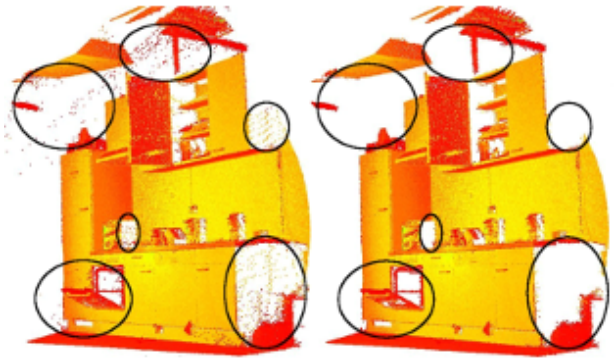


Figure 4.15: Example of an application of filters in Points Cloud. Image adapted [136].

Features

This library contains data structures and mechanisms for estimating 3D features of points in the Point Cloud. 3D features represent 3D points (or spatial positions) that describe the geometric patterns around the point itself. An example to demonstrate the use of 3D features is the estimated curvature of the surface, where the algorithms of the library use information provided by adjacent or near points [137], Figure 4.16 indicates the use of this.

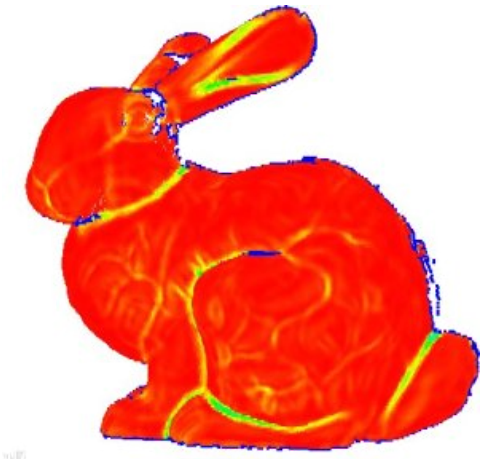


Figure 4.16: Example of a feature application in Points Cloud [137].

Registration

The technique of recording consists of combining data sets into a global model, the main idea is to identify points between data sets and find a transformation that decreases the distance between the corresponding points. The purpose of this process is to improve the alignment between the points [138], as in Figure 4.17.

Whenever the matching search is affected by the relative position and orientation of the data sets, this process will be repeated. The repetition is determined by a threshold value of alignment errors [138].

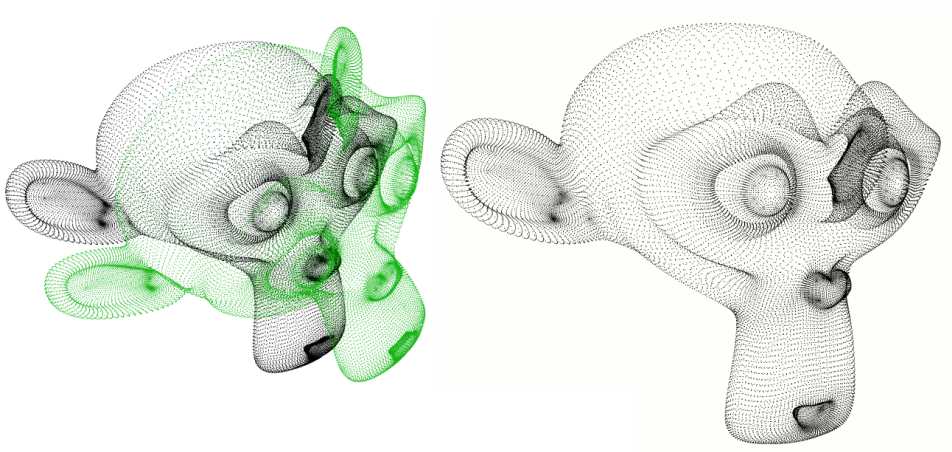


Figure 4.17: Example of a registration application in Points Cloud, on the left the step before the process. Already the right step after the process. Image adapted [139].

Kd-tree

To facilitate the processing of data from a Point Cloud, a good idea is the proposal of this library. When spatially partitioning the dataset, this algorithm creates a kind of tree.[140].

By storing the dataset in a tree format, searching for nearby points becomes more efficient. During the search, the algorithm starts fetching data in groups of larger partitions until it reaches smaller groups [140]. In Figure 4.18, larger blocks contain the smaller blocks.

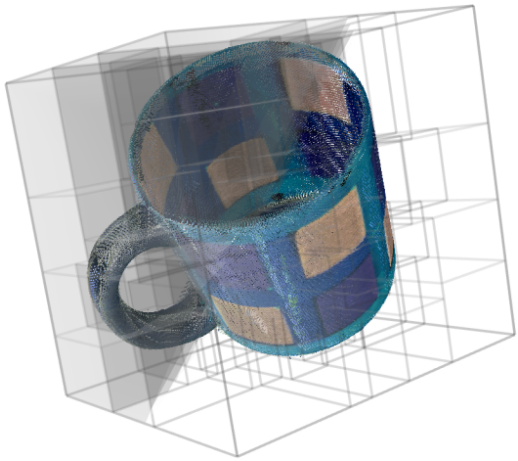


Figure 4.18: Example of an application of Kd-tree in Point Cloud [140].

Octree

Provides algorithms with efficient methods of organizing a data set from a 3D Point Cloud. This allows you to spatially partition the data and aid search operations [141]. The implementation consists of creating a node, it can have eight children or none. The root node delimits a cube around a point, so the point becomes encapsulated, as Figure 4.19 indicates. The larger this process, the higher the resolution of a voxel [141].

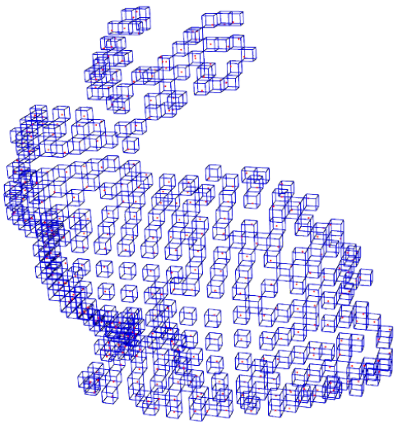
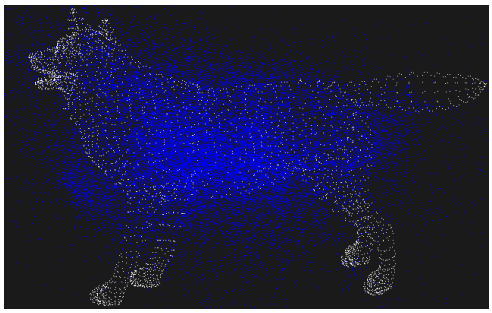


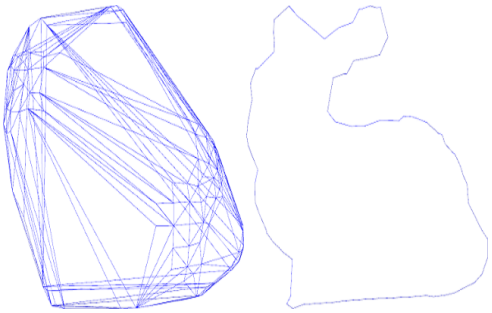
Figure 4.19: Example of an Octree application in Points Cloud [141].

Others

Other small tools can be found within the PCL, these tools contain algorithms that perform the following actions: object recognition (Figure 4.20a), reconstruction of a surface (Figure 4.20b), reading and writing a Point Cloud in files (Figure 4.20c), visualization of the results of algorithms that operate data from the Point Cloud (Figure 4.20d) [108].

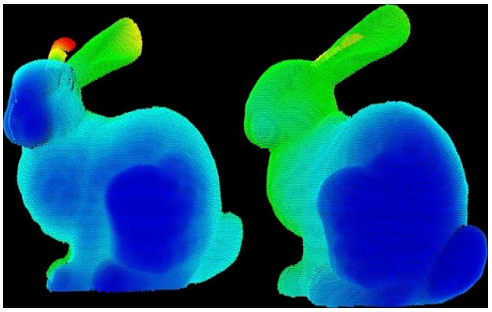


(a) Object recognition [142].



(b) Surface reconstruction [143].

(c) IO [144].



(d) Visualization [145].

Figure 4.20: Set of images with examples of applications in object recognition, surface reconstruction, IO and visualization.

4.3 System architecture

With the union of the related tools in Section 4.2, one can initiate the development of the system that acquires the environment through a sensor RGB-D. And according to the acquisition of this data, plan a path for manipulators (virtual or real) to reach the end point. Therefore, these may have the ability to avoid obstacles. For this purpose, Figure 4.21 presents the simplified block diagram of the system, where each block represents the application of the tools used. The following topics indicate the main characteristics of each of these blocks:

- **User**: At this point the user can check for glitches or view the path found to move the robotic arm;
- **Environment**: Work environment to be acquired by the RGB-D sensor;
- **Command**: Terminal of the operating system, where the user can obtain data in formats of `logs`. These contain information about the entire process. Still in the terminal, the user can perform tasks to control any processes involved;
- **Rviz/MoveIt!**: The union of these two ROS tools graphically demonstrate system performance, that is, the trajectory of the manipulator used and the acquisition of data from the environment. The user can also perform some tasks in the MoveIt! Interface for real or simulation situations;
- **Kinect sensor**: Sensor used for the process of acquiring environmental data. These data are sent to the library PCL for generating the Point Cloud;
- `move_group`: The central element of the system, as in MoveIt !, the `move_group` interconnects all the processes involved, so that they work together in an orderly fashion;
- **Planning scene**: It receives the data, in Point Cloud format, and transforms it into Octree data. Therefore, the system can distinguish the free spaces to trace a path;

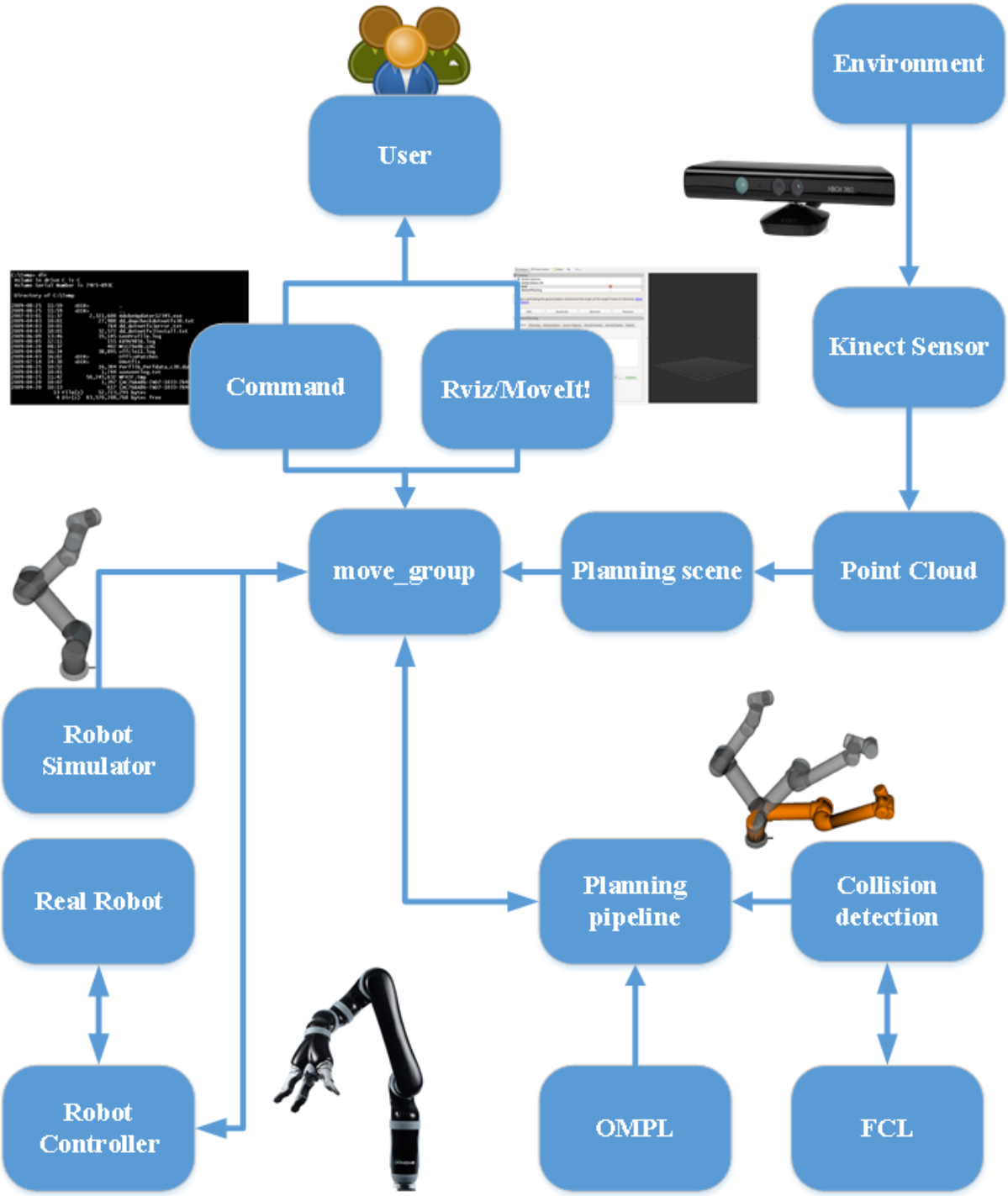


Figure 4.21: Diagram of the system architecture.

- **Point Cloud**: Process that performs the data structure of multidimensional points provided by the sensor;
- **Robot Simulator**: Before applying motion in real situations, it is recommended to run the trajectory in virtual robots to avoid possible failures or accidents. In view of this, this process has great importance in the execution of the system;
- **Real robot**: Execution of the movements in real manipulators;
- **Planning pipeline**: Collect information from the (OMPL planning library and collision library (Flexible Collision Library (FCL)), transmitting to the `move_group` the calculated trajectory and whether or not collisions occur;
- **Collision detection**: Process with the ability to verify that the manipulator is colliding with the parts of its own body and with obstacles fixed at the tip of the tool;
- **Robot Controller**: Controllers of the robot communicate with the central element of the system, in this way they can inform the current position of the robot or receive the execution of movements;
- **OMPL**: Library with algorithms that perform the calculations of trajectory in the free spaces;
- **FCL**: This library contains algorithms for calculating collisions between the manipulators and their parts, in other words, calculate whether the robot will collide or not with its own body.

For the MoveIt! perform the path planning it is necessary to configure the algorithm that will do the routes of the manipulator. For the tests, two algorithms, PRM and RRT, were chosen. Both algorithms were chosen because they are widely used in path planning, so these algorithms can be inserted into the dynamic collision avoidance system.

Only two parameters were changed, planning time and planning attempt while the other parameters remained with the MoveIt! default configuration. The planning time

has been set to 10 seconds. This parameter indicates the time limit at which the system will take to find a path planning. In the parameter planning attempts was set to 15. This parameter indicates to the system how many path planning should be done within the set time. In case the system does not find a path planning within the timeout, the system will not move the manipulator.

Chapter 5

Development

In this chapter 5 describes the steps to implement the system for handlers who avoid collisions, highlighting the most relevant points.

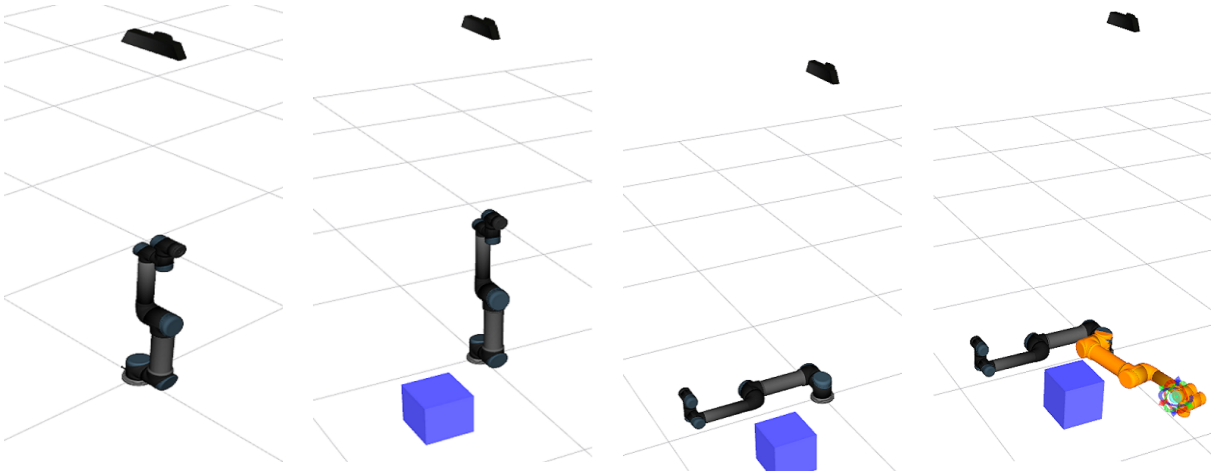
In this Chapter 5 is described the steps to implement the system for manipulators that avoids collisions, highlighting the most relevant points. These stages describe the evolution of the work proposal, which are divided into sections ranging from simulations to the acquisition of the environment as an obstacle. In the Section 5.1 the simulation process of the system with all simulated elements is indicated. In the following section, Section 5.2, the calibration process of the Kinect sensor is described. Finally, in the Sections 5.3 and 5.4, the system simulations with data acquisitions of the sensor used are pointed out. In all cases presented in this chapter, if the system does not find path planning, it will not move the handler. Also, if the system moves the manipulator so that the path passes between the obstacles (tables, boxes and human), it is not able to avoid collisions. All procedures performed herein have been developed in the laboratory of ESTiG, from IPB.

5.1 UR5 Simulation, Kinect and Obstacle

In order to verify if the developed system is able to avoid collisions, a simulation with virtual model of an UR5 manipulator was elaborated (Information about this manipulator

can be found in the subsection 4.2.1). In this way, the objective is to create a scenario with simulated obstacle to guarantee the operation of the system. With the proposal to make MoveIt perform simulated tasks, where the virtual manipulator can not collide with the object, ie the manipulator will have to perform moves to reach the goal position.

The scenario for the analysis of this simulation was elaborated with a box as an obstacle, an UR5 and a Kinect sensor, Figure 5.1 presents this scenario. The UR5 is set at the center point of the simulation, with coordinates [0, 0, 0]. Kinect is centralized with UR5, but at 150 cm in height and its coordinates are [0, 0, 150]. The Figure 5.1a indicates this step. For now the application of Kinect in the scenario is for the purpose of simulation, that is, at this stage the sensor is not yet doing data acquisition of the environment.



(a) Simulated environment with Kinect sensor and UR5.

(b) Inserted a box in the environment (in blue), as an obstacle.

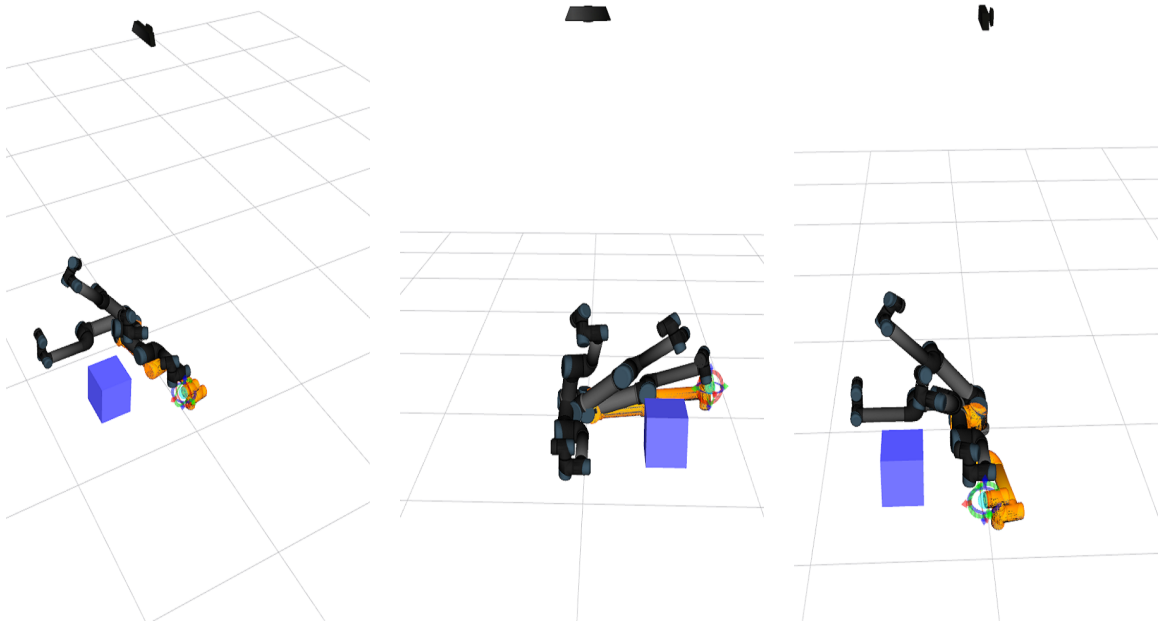
(c) Motion to the starting point.

(d) Determination of the initial and final point of UR5.

Figure 5.1: Sequence of images that demonstrate the steps of the simulation of the environment to move UR5 without collisions.

Next, Figure 5.1b, a dummy box is inserted. With dimensions of 25 cm in length, 25 cm in width and 25 cm in height. And its position [50, 50, 12.5], the frame of reference is relative to its own center of mass. Exactly so, the Z position is half the height of the carton. After setting the initial and final positions, the system is activated to perform the movement. Then, from the path calculations, the system performs the movement of UR5

to reach the end point, the sequence of images in Figure 5.2 demonstrates the trajectory found.



(a) Isometric view. (b) Front view. (c) Side view.

Figure 5.2: Sequence of images that demonstrate from different views the simulation of the system that avoids collisions.

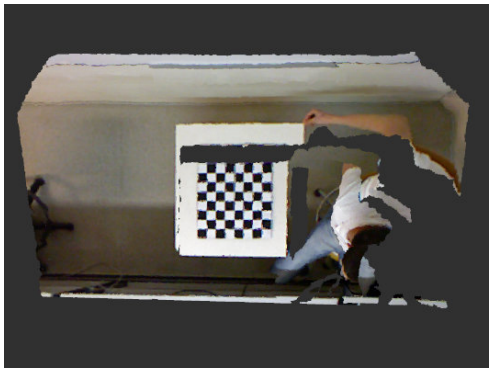
5.2 Kinect Sensor Calibration

Depth sensors provide robots with a complete and instant representation of the appearance and 3D structure of the current surrounding environment. As has been said in Section 3.8, these sensors provide this structure in a Colored Point Cloud format. This kind of information allows the robots to actively perceive and interact with other agents within the work environment.

For reliable and accurate measurement, the intrinsic parameters of each sensor must be accurately calibrated. In other words, calibration must be done because the RGB and depth images are provided by distinct sensors and may be desynchronized, Figure 5.3a indicates exactly this desynchronization. The calibration of consumer depth sensors has

been extensively investigated since the launch of the first generation of Kinect. Several calibration methods, particularly for the depth sensor, have been studied by different research groups [146].

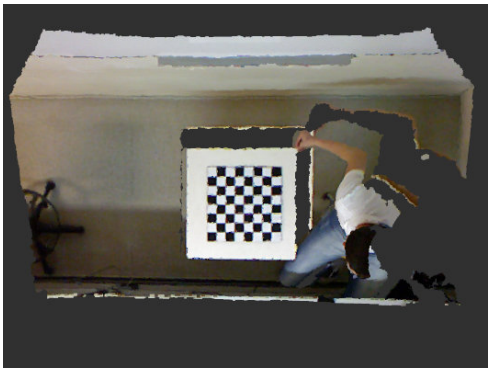
The ROS provides a package, called `camera_calibration`, qualified to perform the calibration procedure for different types of sensors [147]. Its interface can be seen in Figure 5.3b. This package performs the calibration procedure through OpenCV standards, establishing a scale of parameters through a known image [148]. The production of this scale is shown in Figure 5.3c, where there is an 8x8 grid with squares measuring 5 cm. And finally, it is possible to obtain an acquisition of data of the environment with an excellent precision, pointed in Figure 5.3d.



(a) Identification of Kinect desynchronization.

(b) ROS package `camera_calibration`interface.

(c) Recognition of calibration parameters.



(d) Synchronized depth and RGB images.

Figure 5.3: Sequence of images that demonstrate the calibration of the Kinect sensor.

5.3 Simulation of the system avoiding real obstacles

To verify if the developed system is able to avoid collisions with real objects, a simulation was carried out in the laboratory with a Kinect sensor and a virtual model of an UR5 manipulator. In this way, the goal is to create a scenario with real obstacles to ensure the system works. The main idea is to make the Kinect sensor create a Point Cloud of real obstacles and tell MoveIt! where the virtual manipulator can not collide, that is, where the manipulator can perform movements to reach the target position.

(a) Positioning the boxes on the table. (b) Kinect fixation at the top of the scenario. (c) Side view of the elaborate scenario.

Figure 5.4: Sequence of images that demonstrate the scenario elaborated for the simulation of the system that avoids collisions with real obstacles.

The scenario for the analysis of the dynamic system was elaborated with a table as the basis for two boxes that serve as real obstacles, Figure 5.4 presents this scenario. For this analysis, only the surface dimensions of the table matter and are 60 cm long and 60 cm wide. The first box (Box 1) used has dimensions of 26.5 cm in length, 15.6 cm in width and 22.8 cm in height. The other box (Box 2) measures 34.5 cm in length, 6.5 cm in width and 19.2 cm in height. These values are indicated in Table 5.1. The space between the two boxes is 37.7 cm and half of both are referenced according to the center of the table, which makes them centered, as indicated by Figures 5.5 and 5.4a.

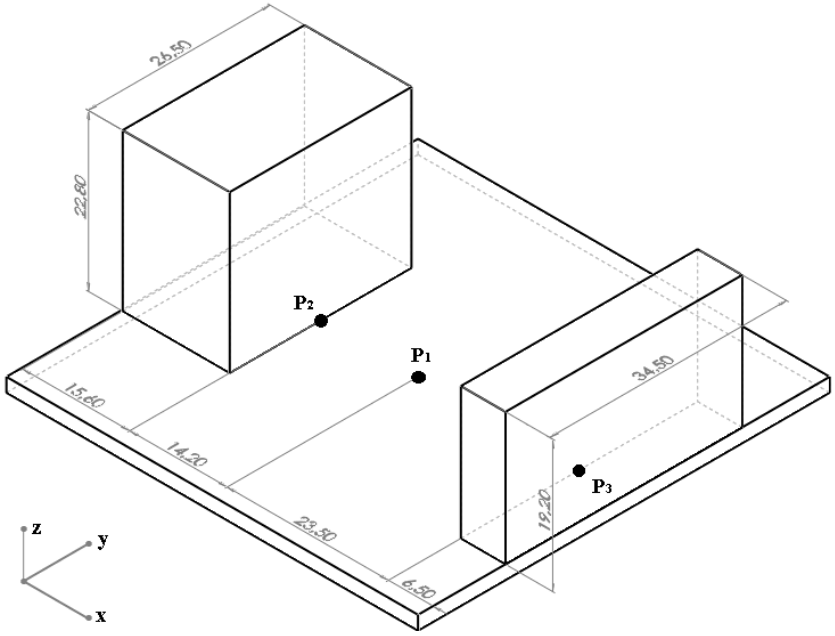


Figure 5.5: Geometric schematic of objects as an example of obstacles.

Object	**Dimensions [cm]**
Box 1	26,5 x 15,6 x 22,8
Box 2	34,5 x 6,5 x 19,2
Table	60 x 60 x 74

Table 5.1: Table with the dimensions of the objects used as obstacles for the simulation with real objects.

For the acquisition of the scenario, the Kinect sensor, which is positioned 190 cm high (relative to the table surface) and half the table length, is applied. The frame in the Kinect body is the base, so these measurements are relative to the sensor base. In Figure 5.4b indicates how Kinect is fixed, the choice to fix it to the top of the scenario is to avoid shadows in the Point Cloud. Another factor is to follow the best performance distance, according to Table 4.3. The whole elaborate scenario can be seen in Figure 5.4c.

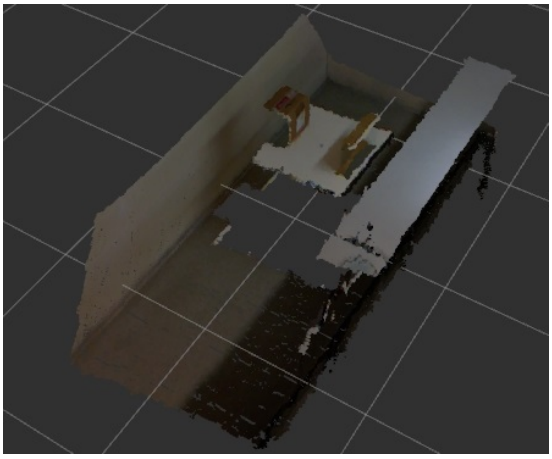
The Table 5.2 shows the Cartesian coordinates of the components used in the simulation. The center of the table is adopted as a reference, so all coordinates are indicated with the distances of each element to the point that represents the center of the table. Therefore, the point in the center of the table is P_1 and has coordinates [0, 0, 0]. The reference point in the components is adopted with the half of the corner closest to the point P_1. Based on the measurements of Figure 5.5 it is possible to extract the coordinate values of points P_2 and P_3, which respectively represent the fixed points of Box 1 and Box 2.

Component	**Distance [xyz] [cm]**
Box 1 (P_2)	[-14,20, 0, 0]
Box 2 (P_3)	[23,5, 0, 0]
Kinect	[30, 0, 190]

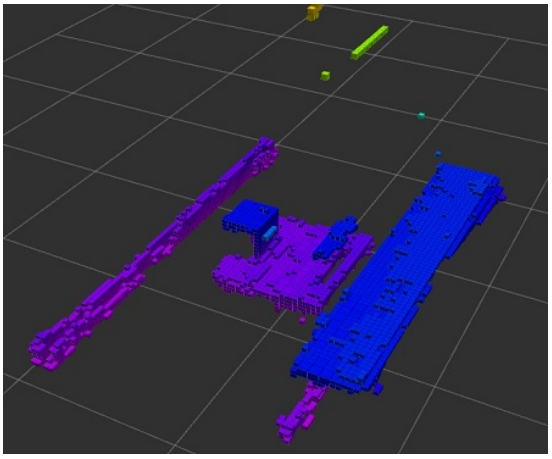
Table 5.2: Table with the coordinates of the objects used and the Kinect sensor for the simulation with real objects.

From the acquisition of scenario data (Figure 5.6a), you can transform the Point Cloud into Octree, method discussed in Section 4.2.6. With the data in Octree format, MoveIt! can determine free spaces to be calculated during planning (Section 3.4 states the importance of knowing free spaces).

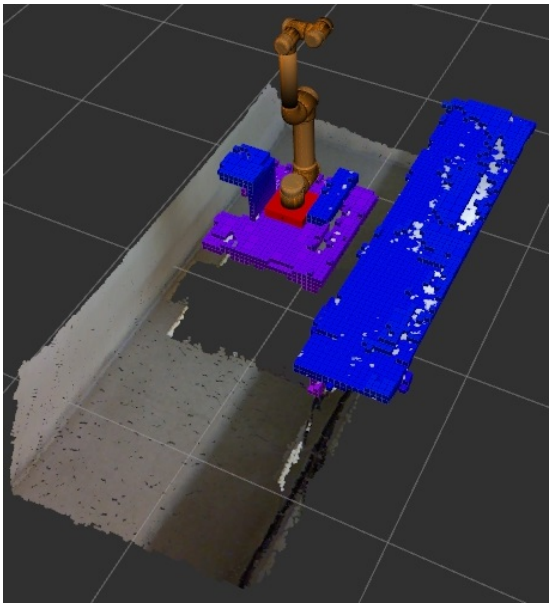
It is analyzed only as obstacle the table and the two boxes, so the method of Octree is configured with values that meet the ranges of height of the obstacles, according to Figure 5.6b. In addition, it is possible to transform the entire Point Cloud into Octree, but it was decided to determine a smaller transformation range to avoid computational costs. So for this case, the UR5 virtual model is fixed on MoveIt! as point P_1, which is



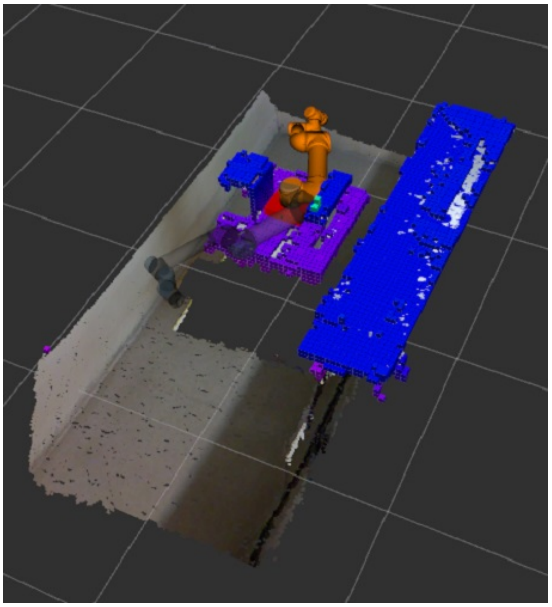
(a) Acquisition of the environment by Point Cloud.



(b) Transformation of the Point Cloud into Octree.



(c) Attach the UR5 handler to the center of the table.



(d) Environment ready for testing with starting and ending points.

Figure 5.6: Sequence of images demonstrating obstacle acquisition and simulation of UR5 to avoid collisions.

the same as the center of the table. So that the manipulator is between the two boxes without maintaining contact with them, as it points to Figure 5.4c. Thus, the initial and final positions can be performed to move UR5.

In the Figure 5.6d, orange indicates the end point and where UR5 is somewhat transparent indicates the starting point. The points were chosen according to the possible movement of the UR5, the highlights can generate different trajectories. For example, the UR5 could perform a trajectory passing perpendicularly on the objects, or could trace a trajectory parallel to the obstacles. There are still other trajectories, which justifies the choice of these points.

With the starting and ending points determined, the system is activated to move the UR5. If the system does not find path planning, the system will not move the handler. And also if the system moves the manipulator so that the path passes between the obstacles (table and the two boxes), the system is not fit for the task of avoiding collisions. The Figure 5.7 shows through the images the movements found by the system to move the robot to the defined end point.

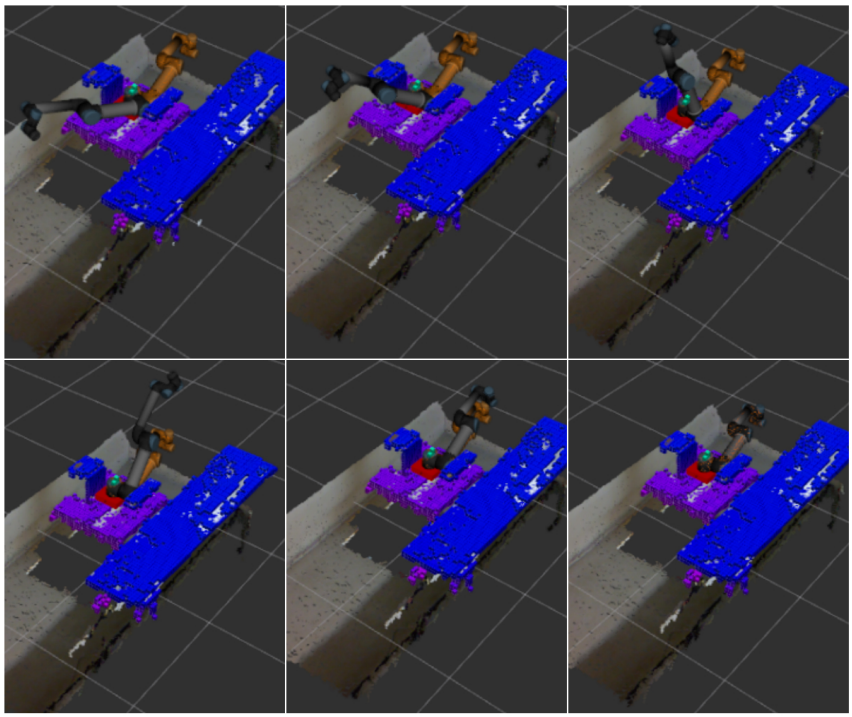


Figure 5.7: Sequence of movements, forming the trajectory of UR5 between obstacles.

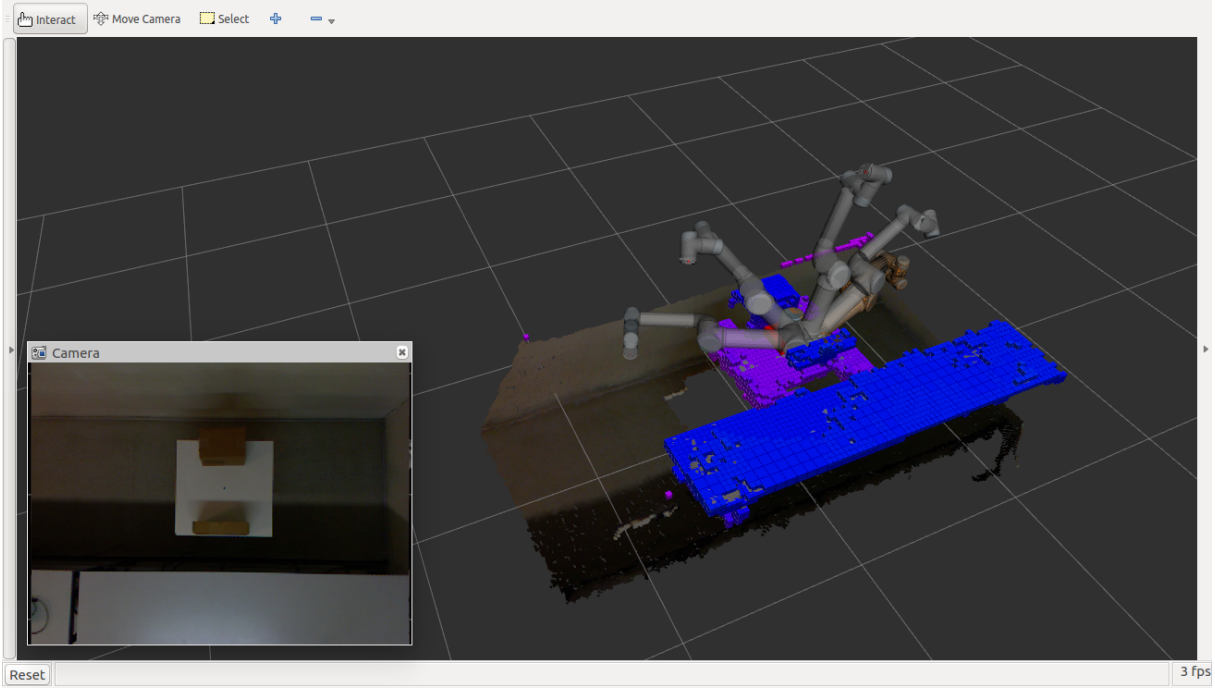


Figure 5.8: Vision of the Kinect RGB camera during the UR5 trajectory between obstacles.

Another representation of how this simulation takes place is in Figure 5.8, we can see in the image the view of the Kinect sensor (left corner) and the simulation in MoveIt! with the trail of UR5 movements between obstacles.

5.4 Simulation of the system avoiding real obstacles and human

To verify that the developed system is capable of avoiding obstacles and human collisions at the same time, a simulation was performed following the same steps as in Section 5.3. The only difference is that for this simulation a box was removed (Box 1), and a human and another table (Table 2) was added. In view of this, it is intended to create a scenario with a real obstacle and a human, to perform the procedure of acquiring the environment and to move the virtual manipulator between the new obstacles. Finally, it is possible to guarantee the operation of the system for this analysis.

(a) Front View. (b) Side View. (c) Isometric View.

Figure 5.9: Sequence of images that demonstrate from different views the creation of the scenario for the simulation of the system that avoids collisions with objects and humans.

The composition of the new scenario can be seen in Figure 5.9, where the human occupies the entire side of Table 1. The new table (Table 2) has the same measurements as the table previously used (Table 1), or 60 cm long, 60 cm wide. The box (Box 2) is centralized between the two tables, this one is 34.5 cm long, 6.5 cm wide and 19.2 cm high, all measurements then indicated in Table 5.3.

Object	**Dimensions [cm]**
Box 1	34,5 x 6,5 x 19,2
Table 1	60 x 60 x 74
Table 2	60 x 60 x 74
Human (occupied space)	60 x 35 x 70

Table 5.3: Table with the dimensions of the objects used as obstacles for the simulation with real and human objects.

The Table 5.4 presents the Cartesian coordinates of the components used in the simulation. As in the previous simulation, the reference is the center of Table 1. Therefore all coordinates are indicated with the distances of each element to the point that represents

the center of Table 1. Then the point (P_1) in the center of the table has coordinates [0, 0, 0]. The referential in the components is adopted with the half of the corner closest to the center, the geometric configuration of the components can be seen in Figure 5.10.

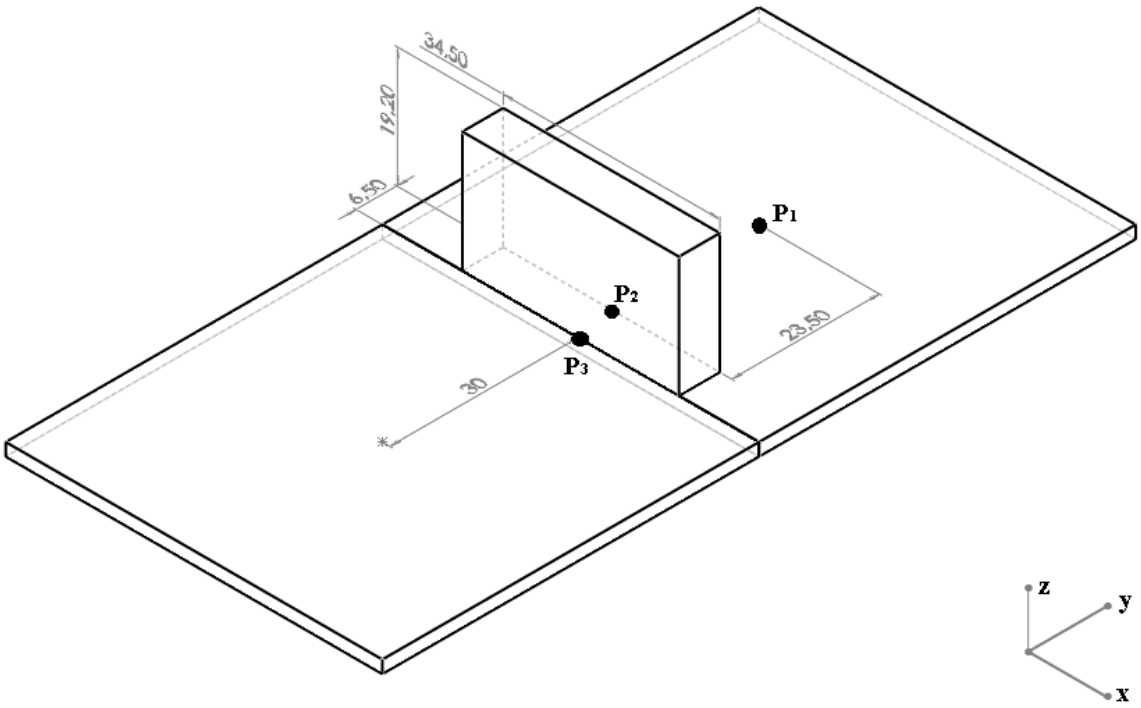


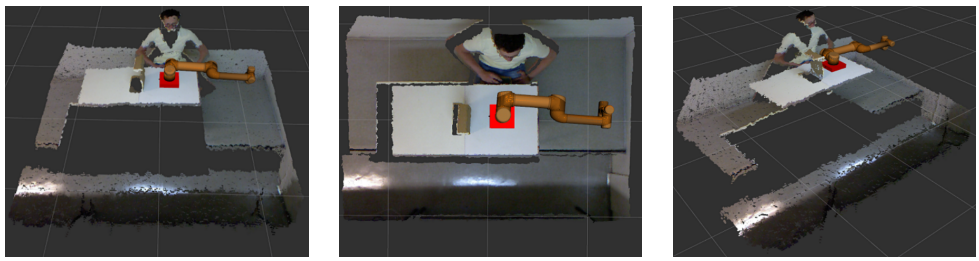
Figure 5.10: Geometric illustration of objects with the mentioned dimensions and points.

Component	**Distance [xyz] [cm]**
Box 1 (P_2)	[0, -23,5, 0]
Table 2 (P_3)	[0, -30, 0
Kinect	[30, 0, 190]

Table 5.4: Table with the distances of the objects used and the Kinect sensor for the simulation with real and human objects.

Based on the acquisition of scenario data, the virtual model of UR5 (in MoveIt!) Can be fixed at point P_1. The Figure 5.11 demonstrates this step from different angles. In this way, the virtual manipulator is positioned without maintaining contact with the human

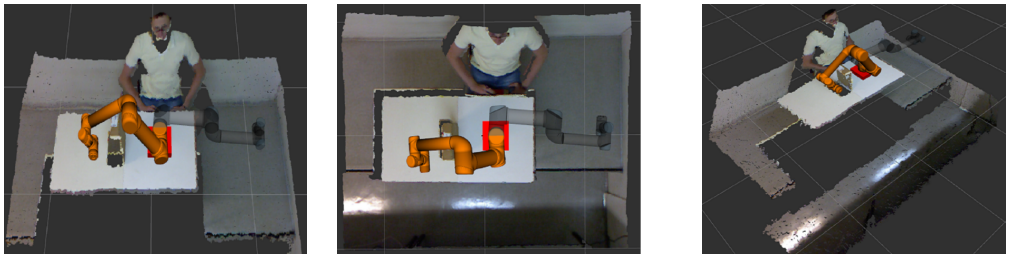
and the box. Therefore, it is possible to carry out initial and final positions to verify if the system is able to move the UR5.



(a) Front view. (b) Top view. (c) Isometric view.

Figure 5.11: Sequence of images that demonstrate from different views the acquisition of the environment and fixation of UR5 for the simulation of the system that avoids collisions with objects and humans.

In Figure 5.12, where UR5 is somewhat transparent indicates the starting point and in orange indicates the end point. Similar to the previous simulation, with the choices of these positions it is possible to expect different trajectories that may or may not pass through the obstacles. Before triggering the system to calculate the trajectory, it is necessary to transform the Point Cloud into Octree data.

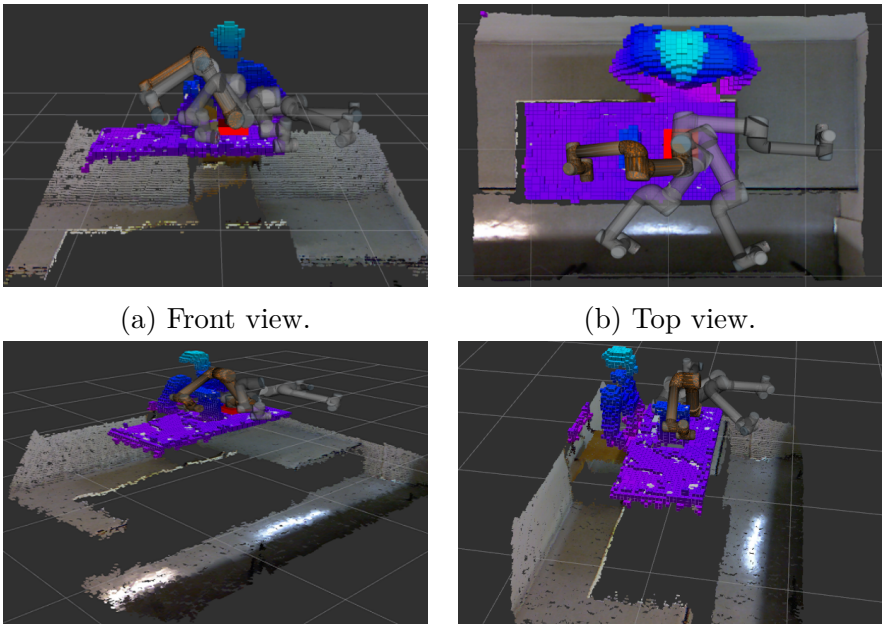


(a) Front view. (b) Top view. (c) Isometric view.

Figure 5.12: Sequence of images that demonstrate from different views the initial and final points of UR5 for the simulation of the system that avoids collisions with objects and humans.

The same method of transforming the Point Cloud into Octree is applied by setting the new values for the height range. In the same way as in Section 5.4, the values for this range are chosen to avoid computational wastage while meeting the space occupied by objects. So with the data in Octree format, the MoveIt! can determine non-colliding

paths to move the UR5 from the start point to the end point. Soon, the system analyzes the tables, the box and the human as an obstacle. And in Figure 5.13 shows through images the Octree data and the movements found by the system to move the robot to the defined end point.



(a) Front view. (b) Top view.

(c) Isometric view. (d) Side view.

Figure 5.13: Sequence of images that demonstrate from different views the movements of UR5 for the simulation of the system that avoids collisions with objects and humans.

Chapter 6

Results

After all the simulations carried out, it is possible to verify if the developed system is able to avoid collisions with obstacles and humans in reality. In which tests were performed on a real robotic manipulator with real obstacles. Similar to the steps in the sections of the previous chapter (Chapter 5), a scenario was created that was captured by an RGB-D sensor and analyzed whether or not the robot collides with objects and humans during its trajectory. The results were obtained in the MIC from the ULE. This Chapter 6 presents the results obtained during the simulations (Section 6.1). All processes and results for the actual handler tests, are in the Section 6.2. And finally, the results obtained by this last step in the Section.

6.1 Result of system simulation

During the simulations, at the end of each path planning performed by the algorithms through MoveIt!, the time and amount of states (or motions) were used to find a route to the final pose. The data obtained during the simulation of the system avoiding real and human obstacles (Section 5.4) were collected and analyzed in graph format. Shown in the Figure 6.1. Although both algorithms find a certain amount of states to perform path planning, it is not necessary to use all found states.

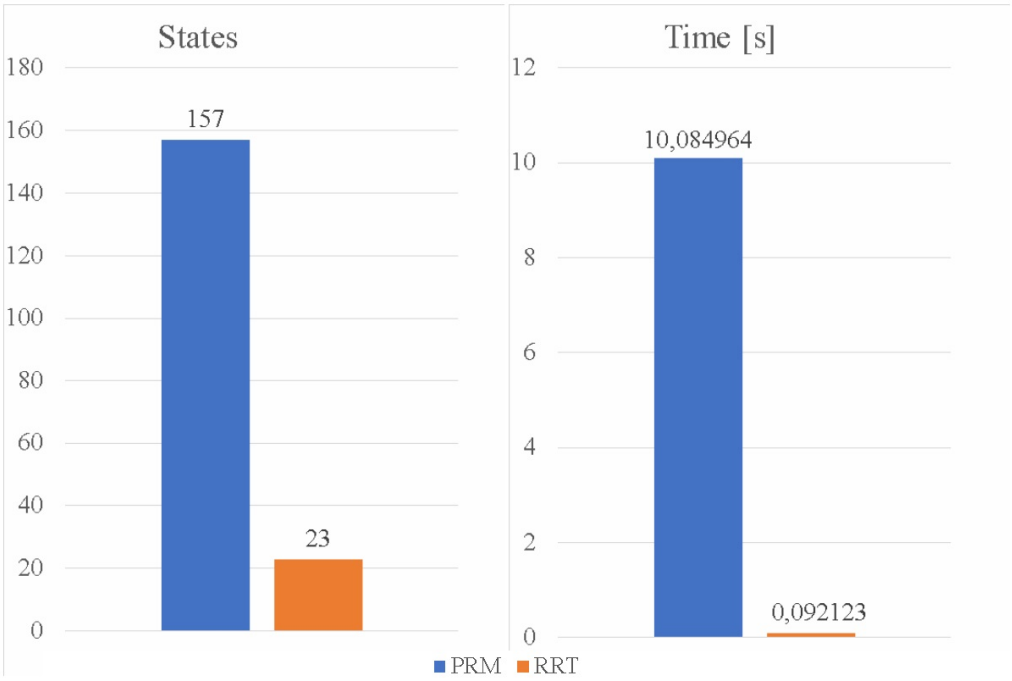


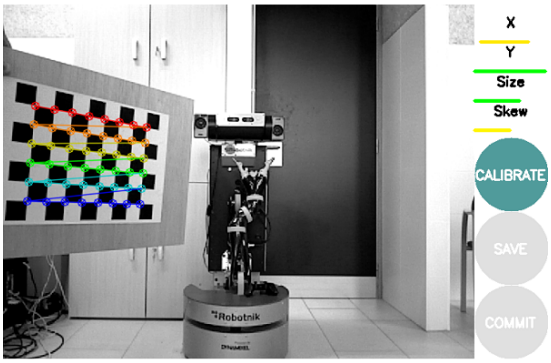
Figure 6.1: Results of the `logs` generated by the system during obstacle and human simulation.

The records presented in graph format show that the PRM algorithm uses all the time it has been configured to find path planning. This resulted in a greater value of pose states. However, the RRT algorithm uses less time to find a path-planning solution, resulting in smaller amounts of pose states.

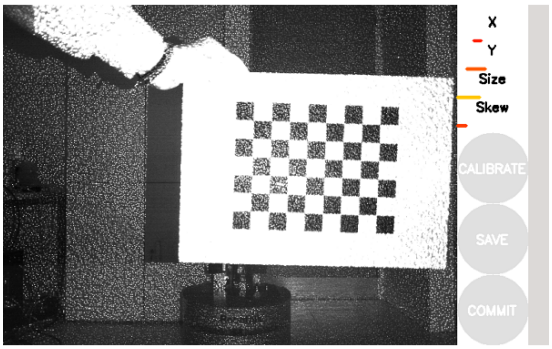
6.2 System avoiding real obstacles with real manipulator

As in the simulations, it is necessary to create a plan to apply the system for manipulators to avoid obstacles in their work environment. The proposed plan starts with the calibration of the Kinect sensor. It was necessary to perform the calibration procedure again, since it is not the same equipment. Thus, to ensure that the new equipment has the same results as the one used in the IPB lab, the calibration procedure is performed, as Figure 6.2 points.

(a) Identification of sensor calibration failure.



(b) Recognition of the frame with the calibration parameters of the RGB image.



(c) Calibration parameters for the depth sensor.

(d) Result after the calibration process.

Figure 6.2: Sequence of images that demonstrate the calibration process for the new Kinect sensor used.

The framework of the scenario for this test can be seen in Figure 6.3, where the Jaco2 6 DOF manipulator is attached to another robot. Further details about the Jaco2 6 DOF can be seen in the Appendix D. While the other robot will serve only as a base for the manipulator, therefore no further details will be addressed. The manipulator model used is different from the simulations, but the idea is to know if the developed system is also capable of being applied to different robotic arms.

(a) Obstacle inserted in the scenario.

(b) Kinect attachment facing the manipulator.

(c) Front view of the elaborate scenario.

Figure 6.3: Sequence of images that demonstrate the elaborate scenario for the test system that avoids collisions with manipulator and real obstacles.

The arm is fixed 39.5 cm from the ground, that is, 13 cm in relation to the base robot, which in turn, is 26.5 cm high. The Kinect is facing the Jaco2 6 DOF, with height of 74 cm (relative to the ground) and 195 cm of the manipulator. Figure 6.4 illustrates the schematic of the scenario with the placement and dimensions of all elements inserted. A foam and rectangular shaped obstacle is inserted into the Jaco2 6 DOF workspace, this obstacle is 27.5 cm long, 5.5 cm wide and 72 cm high. And it is positioned 55 cm away from the arm, or 140 cm from the sensor.

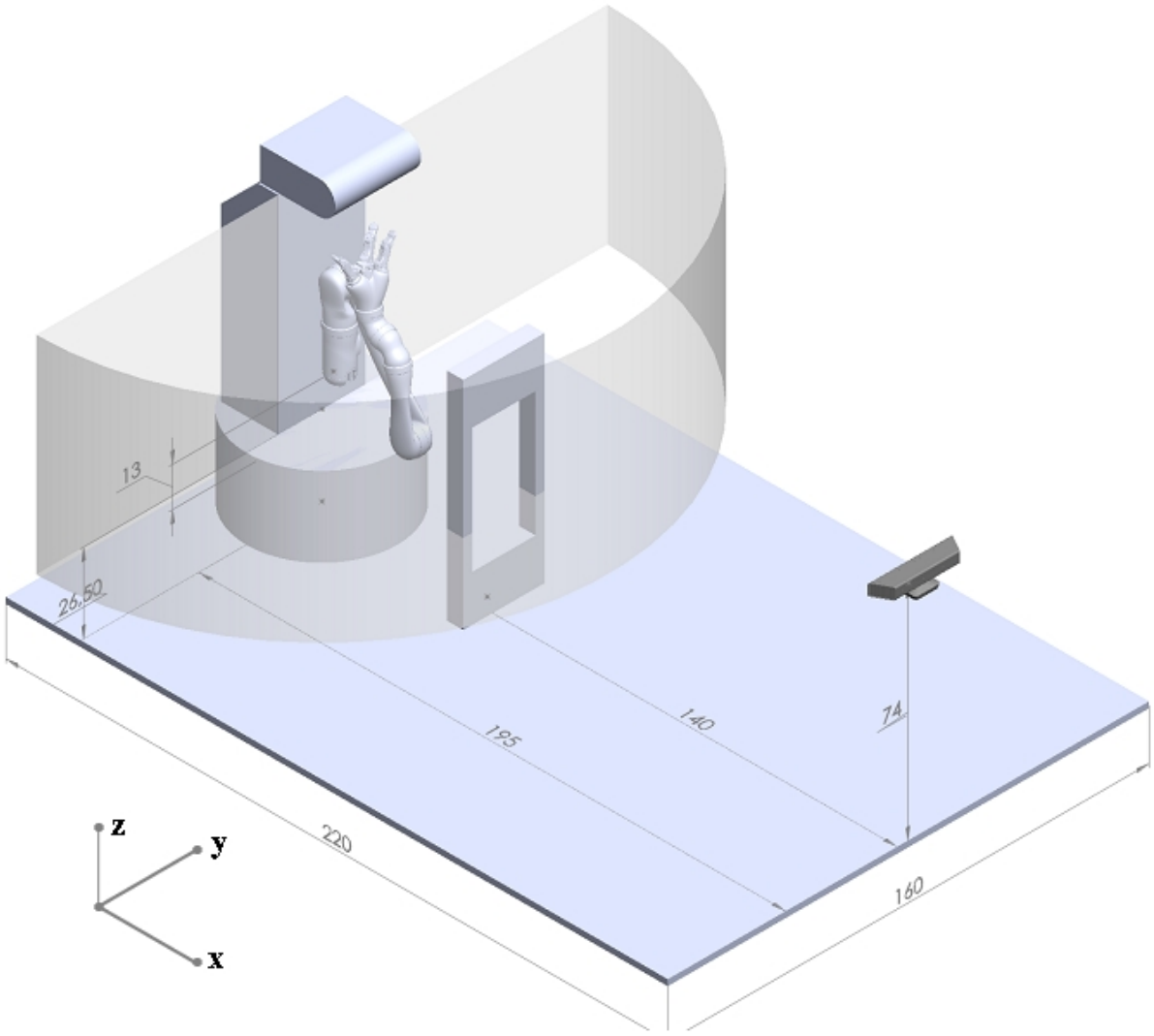
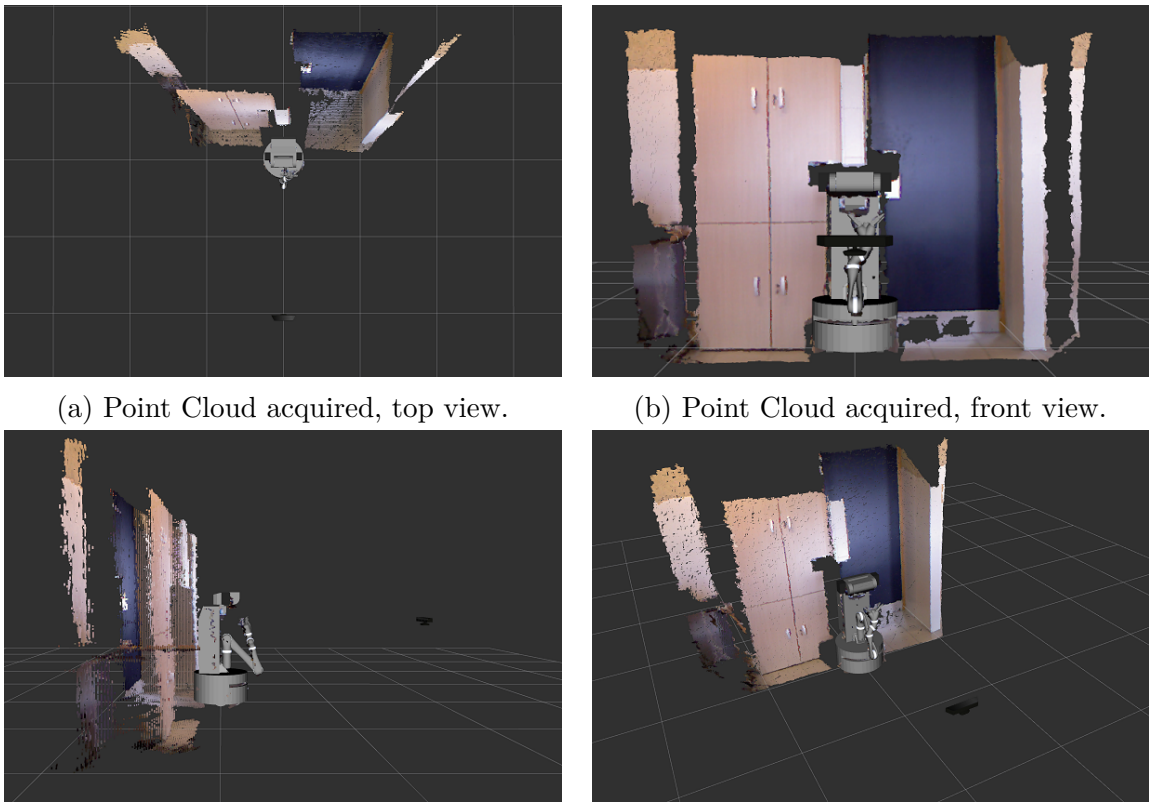


Figure 6.4: Geometric illustration of the components with the dimensions of the positions and the workspace of the robotic arm.

Based on the acquisition of scenario data, one can verify that the virtual model of the Jaco2 6 DOF (in the MoveIt!) Is in the same position as the actual Jaco2 6 DOF. The Figure 6.5 demonstrates this step from different angles. In this way, it is concluded that the virtual manipulator is actually aligned with the actual manipulator positioning. Therefore, the process of transforming the Point Cloud into Octree can be done.



(a) Point Cloud acquired, top view.

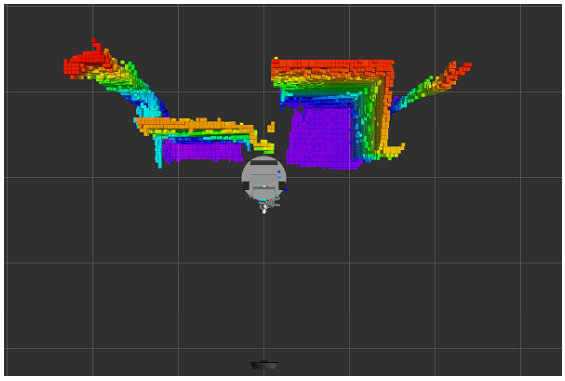
(b) Point Cloud acquired, front view.

(c) Point Cloud acquired, side view.

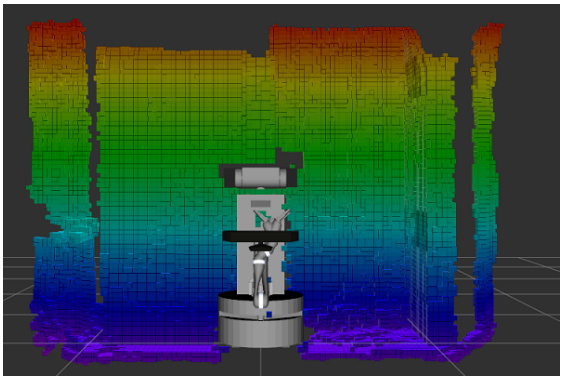
(d) Point Cloud acquired, isometric view.

Figure 6.5: Point Cloud acquired for synchronization with real robot in the MoveIt! environment.

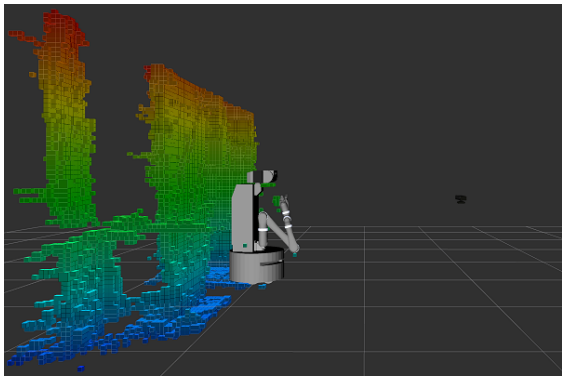
The method of transforming the Point Cloud into Octree is applied without any settings that determine the height range. As the goal is to know if the system is capable of being applied in real situations, it is not necessary to avoid computational wastes. So with the data in Octree format, the MoveIt! can determine the free paths, that is, paths that do not have collisions to move Jaco2 6 DOF. Figure 6.6 shows, from different views, through images the Octree data.



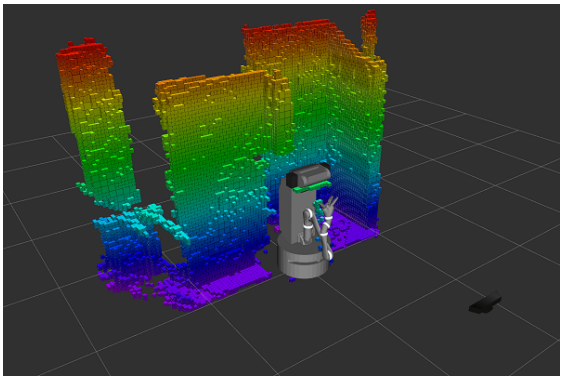
(a) Point Cloud transformed into Octree data, top view.



(b) Point Cloud transformed into Octree data, front view.



(c) Point Cloud transformed into Octree data, side view.



(d) Point Cloud transformed into Octree data, isometric view.

Figure 6.6: Sequence of images that show the Point Cloud transformed into Octree data, from different angles.

With the depth map data in Octree format, you can set the start and end positions to move Jaco2 6 DOF. The Figure 6.7a, indicates the starting point, and in the Figure 6.7b, the end point is indicated. The points were chosen according to the possible movement of the Jaco2 6 DOF, the chosen points can generate different trajectories. For example, the Jaco2 6 DOF could perform a path passing perpendicularly to the obstacle, or could trace a path parallel to the obstacle. There are other possible trajectories, which justifies the choice of these points.

(a) Start point, arm to the right. (b) End point, arm to the left.

Figure 6.7: Definition of start and end points for the test with the system avoiding obstacle with real manipulator.

Before running the proposed obstacle avoidance system, we chose to move Jaco2 6 DOF without the system. The intent is to verify that the manipulator control system has the ability to perform the movement from the starting point to the end point without collisions. Thus, there would be no need to apply and develop the system for this model of manipulator. The Figure 6.8 demonstrates in sequences of images the trajectory obtained in this step. It is concluded that the control system does not have such a capability, so it is applicable to test the system developed to avoid obstacles in this model of manipulator, the Jaco2 6 DOF.

(a) First state of motion, with collision.

(b) Second state of motion, with collision.

(c) Third state of motion, with collision.

Figure 6.8: Sequence of images that show the movement from the start point to the end point, with collision in the obstacle.

With the same starting and ending points, the system is activated to move the Jaco2 6 DOF from the end point to the initial point, ie, to return from the previous movement. If the system does not find path planning, the system will not move the handler. And also if the system moves the manipulator so that the path passes between the obstacle, the system is not fit for the task of avoiding collisions in real situations. The Figure 6.9 and the Figure 6.10 show through images the movements found by the system to move the robot to the starting point.

(a) First state of motion, no collision.

(b) Second state of motion, no collision.

(c) Third state of motion, no collision.

Figure 6.9: First sequence of images demonstrating the movement of the end point to the initial, without collision with the obstacle.

(a) Fourth state of motion, no collision.

(b) Fifth state of motion, no collision.

(c) Sixth state of motion, no collision.

Figure 6.10: Second sequence of images demonstrating the movement of the end point to the initial, without collision with the obstacle.

Due to the results obtained in the sequence of images presented, it is possible to apply the system developed in real situations with humans. For this step, the next session is defined, where there is the same movement with a human in the manipulator workspace. The video that demonstrates the trajectory of this step can be visualized in the following link:

`https://www.youtube.com/watch?v=ZXH10iNyP0Q&t=16s`

6.3 System avoiding human with real manipulator

To verify if the developed system is able to avoid human collisions in the real manipulator, tests were performed following the same steps as in Section 6.2. The only difference is that for this test the foam obstacle has been replaced by a human. In view of this, it is intended to create a real scenario with a human in the working environment of the manipulator, to perform the procedure of acquiring the environment and to move the real manipulator in the free spaces. The system is expected to find a trajectory between the start and end points (the same points defined in the previous section). Figure 6.11 and Figure 6.12 demonstrate through images the sequence of movements found for this trajectory. Finally, it is possible to guarantee or not the operation of the system for this analysis.

(a) First state of motion.

(b) Second state of motion.

(c) Third state of motion.

(d) Fourth state of motion.

Figure 6.11: First sequence of images demonstrating the movement of the end point to the initial, without collision with the human.

(a) Fifth state of motion.

(b) Sixth state of motion.

(c) Seventh state of motion.

(d) Eighth state of motion.

Figure 6.12: Second sequence of images demonstrating the movement of the end point to the initial, without collision with the human.

The results obtained in the sequence of images presented in this section supplied the initial expectation, that is, in the initial phase of this test it was expected that the manipulator had a trajectory that did not collide with the human. The video that demonstrates the trajectory of this step can be visualized in the following link:

`https://www.youtube.com/watch?v=v2Jq4Dhwpe0&t=9s`

Chapter 7

Conclusions

In the course of this dissertation, a system was developed capable of planning in robotic manipulators, without this colliding with objects and humans present in the work environment. Then, the system supplies the proposal of robotic applications that collaborate with humans, according to the new paradigms of Industry 4.0. In the presented document, collaborative manipulators (real and virtual) acted as models for tests and two path planning algorithms were compared.

The system uses the ROS and an RGB-D sensor (Kinect) to acquire the environment, such as human objects and positions. The implemented system allows to re-plan the robotic movement avoiding collisions, guaranteeing the execution of the operations. The `logs` generated by the system show a difference between the analyzed algorithms. While RRT uses agility when finding a solution to plan the path to the final pose, the PRM uses all the time it has been allocated to find a path. Therefore, the use of these algorithms must be adequate to the objective of the implemented project. That is, for projects where the trajectory can be agile and with a low cost structure, the RRT algorithm is the best choice. However, for projects that do not need agility and do not have to worry about the costs of the structure, the PRM algorithm is recommended.

The simulation with the UR5 robot fixed in a Point Cloud acquired by sensors, validates the approach of both algorithms. This process was important to apply the system developed in real environments and robots. Where it was verified that the system had

a good performance, that is, during the movements of the robot (real and virtual) there were no trajectories that crossed obstacles and humans. And when the system did not find a trajectory within the free spaces, it did not move the robotic arm; making it a secure system to be implemented in real robots.

The development of this work has already provided the production of an article of scientific interest and an internship in MIC, in the city of Leon, Spain. Currently there are other projects from this work, such as at least two more articles and a technical report. In conclusion, the work exposed performs a good solution for pick-and-place operations with obstacles. This proposed solution can be applied in several fields of industrial robotics, due to its flexibility for applications in different models of manipulators.

7.1 Future works

Based on the work presented, it is possible to identify several points that can support future studies. The studies to be carried out in the future can be searched through the following topics:

- Optimize the Point Cloud acquisition so that the system has a faster response to the introduction of new objects on the robot's desktop. And test the system developed in other models of robotic manipulators, even in simulation format. In this way, the system could be elaborated with a more generic approach, in order to satisfy all the models available in the market of industrial robots;

- Run the system developed using higher computational resources, that is, with processing capacity and available memory larger than those already used. With this, the resolutions for transformation of the Cloud of Points in Octree can be bigger, causing in greater precision of movement;

- Apply the system with different planning algorithms, such as other derivations of RRT and PRM. And introduce more RGB-D sensors, creating a Point Cloud and analyze if it causes greater efficiency in the generation of trajectories.

Bibliography

[1] C. D. Mutto, P. Zanuttigh, and G. M. Cortelazzo, *Time-of-flight cameras and microsoft kinect*, 1st ed., ser. SpringerBriefs in Electrical and Computer Engineering. Springer-Verlag New York, 2012, ISBN: 978-1-4614-3806-9,978-1-4614-3807-6.

[2] M. Magazine, "An automatic block-setting crane", Tech. Rep., 1938.

[3] Y. K. Hwang and N. Ahuja, "Gross motion planning - a survey", *ACM Comput. Surv.*, vol. 24, pp. 219–291, 1992.

[4] J. Wallén, "The history of the industrial robot", Tech. Rep., May 2008.

[5] P. Chua, T. Ilschner, and D. Caldwell, "Robotic manipulation of food products – a review", *Industrial Robot: The international journal of robotics research and application*, vol. 30, no. 4, pp. 345–354, 2003. DOI: 10.1108/01439910310479612. [Online]. Available: https://doi.org/10.1108/01439910310479612.

[6] X. Jiang, K. m. Koo, K. Kikuchi, A. Konno, and M. Uchiyama, "Robotized assembly of a wire harness in car production line", in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Oct. 2010, pp. 490–495. DOI: 10.1109/IROS.2010.5653133.

[7] I. F. of Robotics, "Executive summary world robotics 2017 industrial robots", Tech. Rep., Sep. 2016.

[8] A. Farnsworth, "Featuring packing medicine rhinocort with irb 140 and palletizing with irb 4400", Tech. Rep., May 2008.

[9] Y. Zhang, B. K. Chen, X. Liu, and Y. Sun, "Autonomous robotic pick-and-place of microobjects", *IEEE Transactions on Robotics*, vol. 26, no. 1, pp. 200–207, Feb. 2010, ISSN: 1552-3098. DOI: 10.1109/TRO.2009.2034831.

[10] R. Mattone, G. Campagiorni, and F. Galati, "Sorting of items on a moving conveyor belt. part 1: A technique for detecting and classifying objects", *Robotics and Computer-Integrated Manufacturing*, vol. 16, no. 2, pp. 73–80, 2000, ISSN: 0736-5845. DOI: https://doi.org/10.1016/S0736-5845(99)00040-X. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S073658459900040X.

[11] R. Mattone, M. Divona, and A. Wolf, "Sorting of items on a moving conveyor belt. part 2: Performance evaluation and optimization of pick-and-place operations", vol. 16, pp. 81–90, Apr. 2000.

[12] T. Gecks and D. Henrich, "Human-robot cooperation: Safe pick-and-place operations", in *ROMAN 2005. IEEE International Workshop on Robot and Human Interactive Communication, 2005.*, Aug. 2005, pp. 549–554. DOI: 10.1109/ROMAN.2005.1513837.

[13] C. Y. Tsai, C. C. Wong, C. J. Yu, C. C. Liu, and T. Y. Liu, "A hybrid switched reactive-based visual servo control of 5-dof robot manipulators for pick-and-place tasks", *IEEE Systems Journal*, vol. 9, no. 1, pp. 119–130, Mar. 2015, ISSN: 1932-8184. DOI: 10.1109/JSYST.2014.2358876.

[14] P. Andhare and S. Rawat, "Pick and place industrial robot controller with computer vision", in *2016 International Conference on Computing Communication Control and automation (ICCUBEA)*, Aug. 2016, pp. 1–4. DOI: 10.1109/ICCUBEA.2016.7860048.

[15] K. M. Nalini and R. R. Gondkar, "Roboic recognition for unstructered 2-d parts to pick and place objects", in *2017 2nd IEEE International Conference on Recent Trends in Electronics, Information Communication Technology (RTEICT)*, May 2017, pp. 1478–1482. DOI: 10.1109/RTEICT.2017.8256843.

[16] L. Wang, L. Ren, J. K. Mills, and W. L. Cleghorn, "Automated 3-d micrograsping tasks performed by vision-based control", *IEEE Transactions on Automation Science and Engineering*, vol. 7, no. 3, pp. 417–426, Jul. 2010, ISSN: 1545-5955. DOI: 10.1109/TASE.2009.2036246.

[17] S. Daoud, H. Chehade, F. Yalaoui, and L. Amodeo, "Efficient metaheuristics for pick and place robotic systems optimization", *Journal of Intelligent Manufacturing*, vol. 25, no. 1, pp. 27–41, Feb. 2014, ISSN: 1572-8145. DOI: 10.1007/s10845-012-0668-z. [Online]. Available: https://doi.org/10.1007/s10845-012-0668-z.

[18] C. Son, "Optimal control planning strategies with fuzzy entropy and sensor fusion for robotic part assembly tasks", *International Journal of Machine Tools and Manufacture*, vol. 42, no. 12, pp. 1335–1344, 2002, ISSN: 0890-6955. DOI: https://doi.org/10.1016/S0890-6955(02)00063-9. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0890695502000639.

[19] ——, "Intelligent robotic path finding methodologies with fuzzy/crisp entropies and learning.", vol. 26, Jan. 2011.

[20] M. W. Spong, S. Hutchinson, M. Vidyasagar, *et al.*, *Robot modeling and control*. Wiley New York, 2006, vol. 3.

[21] J.-P. Merlet, *Parallel robots. solid mechanics and its application*, 2006.

[22] H. Ghorabi, Y. Maddahi, S. Mohammad Hosseini Monsef, and A. Maddahi, "Design and experimental tests of a pick and place robot: Theoretical and experimental approaches", pp. 144–150, Mar. 2010.

[23] Z. Li, Y. Lou, Z. Li, G. Yang, and J. Gao, "T2: A novel two degree-of-freedom translational parallel robot for pick-and-place operation", in *IEEE ICCA 2010*, Jun. 2010, pp. 725–730. DOI: 10.1109/ICCA.2010.5524340.

[24] A. M. Pinto, E. Moreira, J. Lima, J. P. Sousa, and P. Costa, "A cable-driven robot for architectural constructions: A visual-guided approach for motion control and path-planning", *Autonomous Robots*, vol. 41, no. 7, pp. 1487–1499, Oct. 2017,

ISSN: 1573-7527. DOI: 10.1007/s10514-016-9609-6. [Online]. Available: https://doi.org/10.1007/s10514-016-9609-6.

[25] E. Ottaviano, M. Ceccarelli, and M. D. Ciantis, "A 4-4 cable-based parallel manipulator for an application in hospital environment", in *2007 Mediterranean Conference on Control Automation*, Jun. 2007, pp. 1–6. DOI: 10.1109/MED.2007.4433839.

[26] T. Dallej, M. Gouttefarde, N. Andreff, R. Dahmouche, and P. Martinet, "Vision-based modeling and control of large-dimension cable-driven parallel robots", in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Oct. 2012, pp. 1581–1586. DOI: 10.1109/IROS.2012.6385504.

[27] K. Usher, G. Winstanley, and R. Carnie, "Air vehicle simulator: An application for a cable array robot", in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, Apr. 2005, pp. 2241–2246. DOI: 10.1109/ROBOT.2005.1570446.

[28] P. H. Borgstrom, B. L. Jordan, B. J. Borgstrom, M. J. Stealey, G. S. Sukhatme, M. A. Batalin, and W. J. Kaiser, "Nims-pl: A cable-driven robot with self-calibration capabilities", *IEEE Transactions on Robotics*, vol. 25, no. 5, pp. 1005–1015, Oct. 2009, ISSN: 1552-3098. DOI: 10.1109/TRO.2009.2024792.

[29] P. Bosscher, L. W. Robert, L. S. Bryson, and D. Castro-Lacouture, "Cable-suspended robotic contour crafting system", *Automation in Construction*, vol. 17, no. 1, pp. 45–55, 2007, ISSN: 0926-5805. DOI: https://doi.org/10.1016/j.autcon.2007.02.011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0926580507000362.

[30] S.-R. Oh and S. K. Agrawal, "Generation of feasible set points and control of a cable robot", *IEEE Transactions on Robotics*, vol. 22, no. 3, pp. 551–558, Jun. 2006, ISSN: 1552-3098. DOI: 10.1109/TRO.2006.870646.

[31] S. Lahouar, E. Ottaviano, S. Zeghoul, L. Romdhane, and M. Ceccarelli, "Collision free path planning for cable driven parallel robots", *Robotics and Autonomous Systems*, vol. 57, no. 11, pp. 1083–1093, 2009, ISSN: 0921-8890. DOI: https://doi.org/10.1016/j.robot.2009.07.006. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0921889009000931.

[32] J. Barraquand and J.-C. Latombe, "Robot motion planning: A distributed representation approach", vol. 10, pp. 628–649, Dec. 1991.

[33] E. Ralli and G. Hirzinger, "Fast path planning for robot manipulators using numerical potential fields in the configuration space", in *Intelligent Robots and Systems '94. 'Advanced Robotic Systems and the Real World', IROS '94. Proceedings of the IEEE/RSJ/GI International Conference on*, vol. 3, Sep. 1994, 1922–1929 vol.3. DOI: 10.1109/IROS.1994.407663.

[34] L. E. Kavraki, P. Svestka, J. C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces", *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, Aug. 1996, ISSN: 1042-296X. DOI: 10.1109/70.508439.

[35] T. Siméon, J.-P. Laumond, and C. Nissoux, "Visibility-based probabilistic roadmaps for motion planning. journal of advanced robotics, 14(6), 477-494", vol. 14, pp. 477–493, Jan. 2000.

[36] S. A. Wilmarth, N. M. Amato, and P. F. Stiller, "Maprm: A probabilistic roadmap planner with sampling on the medial axis of the free space", in *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, vol. 2, 1999, 1024–1031 vol.2. DOI: 10.1109/ROBOT.1999.772448.

[37] R. Bohlin and L. E. Kavraki, "Path planning using lazy prm", in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, vol. 1, Apr. 2000, 521–528 vol.1. DOI: 10.1109/ROBOT.2000.844107.

[38] E. Plaku, K. E. Bekris, B. Y. Chen, A. M. Ladd, and L. E. Kavraki, "Sampling-based roadmap of trees for parallel motion planning", *IEEE Transactions on Robotics*, vol. 21, no. 4, pp. 597–608, Aug. 2005, ISSN: 1552-3098. DOI: 10.1109/TRO.2005.847599.

[39] C. Helguera and S. Zeghloul, "A local-based method for manipulators path planning in heavy cluttered environments", in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, vol. 4, 2000, 3467–3472 vol.4. DOI: 10.1109/ROBOT.2000.845266.

[40] L. Blackmore and B. Williams, "Optimal manipulator path planning with obstacles using disjunctive programming", in *2006 American Control Conference*, Jun. 2006, p. 3. DOI: 10.1109-ACC.2006.1657210.

[41] S. Lahouar, S. Zeghloul, and L. Romdhane, "Real time path planning for multi dof manipulators in dynamic environment", *International Journal of Advanced Robotic Systems*, vol. 3, no. 2, p. 20, 2006.

[42] P. Tavares, J. Lima, P. Costa, and A. Moreira, "Multiple manipulators path planning using double a*", vol. 43, pp. 657–664, Oct. 2016.

[43] D. Han, H. Nie, J. Chen, and M. Chen, "Dynamic obstacle avoidance for manipulators using distance calculation and discrete detection", *Robotics and Computer-Integrated Manufacturing*, vol. 49, pp. 98–104, 2018, ISSN: 0736-5845. DOI: https://doi.org/10.1016/j.rcim.2017.05.013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0736584516303957.

[44] F. Cheng, W. Ji, D. Zhao, and J. Lv, "Apple picking robot obstacle avoidance based on the improved artificial potential field method", in *2012 IEEE Fifth International Conference on Advanced Computational Intelligence (ICACI)*, Oct. 2012, pp. 909–913. DOI: 10.1109/ICACI.2012.6463303.

[45] O. Ghita and P. Whelan, "A bin picking system based on depth from defocus", vol. 13, Feb. 2003.

[46] C. Park, J. Pan, and D. Manocha, "Real-time optimization-based planning in dynamic environments using gpus", in *2013 IEEE International Conference on Robotics and Automation*, May 2013, pp. 4090–4097. DOI: 10.1109/ICRA.2013.6631154.

[47] R. C. Luo and C. W. Kuo, "Intelligent seven-dof robot with dynamic obstacle avoidance and 3-d object recognition for industrial cyber physical systems in manufacturing automation", *Proceedings of the IEEE*, vol. 104, no. 5, pp. 1102–1113, May 2016, ISSN: 0018-9219. DOI: 10.1109/JPROC.2015.2508598.

[48] J. F. Engelberger, *Robotics in practice: Management and applications of industrial robots*. Kogan Page, 1980, ISBN: 978-0-85038-669-1.

[49] M. P. Groover, M. Weiss, R. N. Nagel, and N. G. Odrey, *Robótica: Tecnologia e programação*. São Paulo, Brasil: McGraw-Hill, 1988.

[50] G. Litzenberger, "World robotics 2016 industrial robots", *Executive Summary*, 2016.

[51] I. O. for Standardization, *Iso 83732012*, https://www.iso.org/standard/55890.html, Acessado Julho de 2017.

[52] D. Kandray, *Programmable automation technologies : An introduction to cnc, robotics and plcs*, 1st ed. Industrial Press, 2010, ISBN: 978-0-8311-3346-7,0831133465,293-293-294-2,489-489-490-4.

[53] V. CARRARA, "Apostila de robótica", *Universidade Braz Cubas, Área de Ciências Exatas Engenharia Mecânica, Engenharia de Controle e Automamação*, 2009.

[54] K. S. Fu, R. C. Gonzalez, and C. S. G. Lee, *Robotics: Control, sensing, vision and intelligence*, 1st ed. New York: McGraw-Hill, 1987, ISBN: 0-07-022625-3.

[55] S. Y. Nof, *Handbook of industrial robotics*, 2nd ed. John Wiley Sons Inc, 1999, ISBN: 0-471-17783-0.

[56] E. Appleton and D. J. Williams, *Industrial robot applications*, 1st ed., ser. Open University Press Robotics Series. Springer Netherlands, 1987, ISBN: 978-94-010-7905-1,978-94-009-3125-1.

[57] E. T. Group, *Integrated chiron machining cells give a new dimension to castings specialist*, http://www.engtechgroup.com/blog/integrated-chiron-machining-cells-give-a-new-dimension-to-castings-specialist/, Acessado Julho de 2017.

[58] V. A. M. CECILIO, *Testes de soldadura mig standard e cmt pulsado robotizada em liga de alumínio 6082-t6 para otimização de penetração, largura de cordão e reforço.* http://hdl.handle.net/10198/14592, Sep. 2017.

[59] D. J. Todd, *Fundamentals of robot technology: An introduction to industrial robots, teleoperators and robot vehicles*, 1st ed. Pentonville Road: Kogan Page, 1986, ISBN: 978-94-011-6770-3.

[60] P. Technology, *Top industrial robotics companies in the world*, https://www.plantautomation-technology.com/, Acessado Agosto de 2017.

[61] CLOOS, *Your brand for innovative welding technology*, http://www.cloos.de/de-de/startseite/, Acessado Agosto de 2017.

[62] Comau, *A global spirit, a local presence*, http://www.comau.com/EN, Acessado Agosto de 2017.

[63] DENSO, *We are only satisfied when you are*, http://densorobotics.com/world/, Acessado Agosto de 2017.

[64] E. Robots, *Exceed your vision*, http://robots.epson.com/, Acessado Agosto de 2017.

[65] H. Robotics, *Hyundai robotics , as a pioneer in robot automation industry, will offer products and services with better performance, quality and reliability*, http://www.hyundai-robotics.com/www/english/, Acessado Agosto de 2017.

[66] I. Robotersysteme, *50 years igm robotersysteme ag*, http://www.igm-group.com/en, Acessado Agosto de 2017.

[67] I. Actuator, *The electric alternative for sustainable and energy-saving automation*, https://www.intelligentactuator.com/, Acessado Agosto de 2017.

[68] Janome, *Janome industrial equipment*, http://www.janomeie.com/, Acessado Agosto de 2017.

[69] Kawasaki, *Industrial robots and automation solutions*, https://robotics.kawasaki.com/, Acessado Agosto de 2017.

[70] KUKA, *The power of automation*, https://www.kuka.com/, Acessado Agosto de 2017.

[71] N.-F. CORP., *Challenging growth in business to fulfill aspirations as a manufacturing company*, http://www.nachi-fujikoshi.co.jp/eng/index.html, Acessado Agosto de 2017.

[72] N. Sankyo, *All for dreams*, http://www.nidec-sankyo.co.jp/english/, Acessado Agosto de 2017.

[73] O. Daihen, *Advanced welding and robotic systems*, http://www.daihen-usa.com/, Acessado Agosto de 2017.

[74] R. Robotics, *Worldwide leader in systems integration*, http://www.reisroboticsusa.com/, Acessado Agosto de 2017.

[75] ABB, *Robotica abb*, http://new.abb.com/, Acessado Agosto de 2017.

[76] M. Electric, *Robôs industriais melfa*, http://br.mitsubishielectric.com/pt/, Acessado Agosto de 2017.

[77] F. America, *Industrial robots for manufacturing*, http://www.fanucamerica.com/, Acessado Agosto de 2017.

[78] Yaskawa, *Motoman robots*, https://www.yaskawa.eu.com/en/, Acessado Agosto de 2017.

[79] A. Technologies, *An intelligent robot for every application*, http://www.adept.com/, Acessado Agosto de 2017.

[80] Stäubli, *A passion for innovation*, https://www.staubli.com/, Acessado Agosto de 2017.

[81] U. Robotics, *Making robot automation accessible to all levels of industry*, https://www.universal-robots.com, Acessado Agosto de 2017.

[82] ——, "Brochure: The future is collaborative", Tech. Rep., Aug. 2017.

[83] I.-M. Chen, G. Yang, and S. H. Yeo, "Automatic modeling for modular reconfigurable robotic systems: Theory and practice", in *Industrial Robotics: Theory, Modelling and Control*, InTech, 2006.

[84] L. Josuet, B. Carlos, L. Hsien-I, H. Te-Sheng, and W. Chun-Sheng, "An improved inverse kinematics solution of 6r-dof robot manipulators with euclidean wrist using dual quaternions", in *2016 International Automatic Control Conference (CACS)*, Nov. 2016, pp. 77–82. DOI: 10.1109/CACS.2016.7973887.

[85] L. Joseph, *Mastering ros for robotics programming: Design, build and simulate complex robots using robot operating system and master its out-of-the-box functionalities*, 1st ed. Packt Publishing, 2015, ISBN: 978-1-78355-179-8.

[86] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library", *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, Dec. 2012, http://ompl.kavrakilab.org. DOI: 10.1109/MRA.2012.2205651.

[87] ——, *The open motion planning library*, http://ompl.kavrakilab.org/, Acessado Dezembro de 2017.

[88] E. M. e Robótica, "Descubra a simplicidade", http://epl-si.com/, Acessado Agosto de 2017.

[89] S. Laboratory and R. Laboratory, *Openrdk website*, http://openrdk.sourceforge.net/index.php, Acessado Agosto de 2017.

[90] D. Calisi, A. Censi, L. Iocchi, and D. Nardi, "OpenRDK: A modular framework for robotic software development", in *Proceedings of International Conference on Intelligent Robots and Systems (IROS)*, Nice, France, Sep. 2008, pp. 1872–1877, ISBN: 978-1-4244-2057-5. DOI: 10.1109/IROS.2008.4651213.

[91] ——, "OpenRDK: A framework for rapid and concurrent software prototyping", in *Proceedings of Int. Workshop on System and Concurrent Engineering for Space Applications (SECESA)*, Nov. 2008.

[92] H. Bruyninckx, "Open robot control software: The orocos project", in *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, IEEE, vol. 3, 2001, pp. 2523–2528.

[93] H. Bruyninckx, P. Soetens, and B. Koninckx, "The real-time motion control core of the orocos project", in *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*, IEEE, vol. 2, 2003, pp. 2766–2771.

[94] OROCOS, *Smarter control in robotics and automation*, hhttp://www.orocos.org/, Acessado Agosto de 2017.

[95] H. Bruyninckx, "Orocos: Design and implementation of a robot control software framework", in *Proceedings of IEEE International Conference on Robotics and Automation*, 2002.

[96] J. Jackson, "Microsoft robotics studio: A technical introduction", *IEEE Robotics & Automation Magazine*, vol. 14, no. 4, 2007.

[97] K. Johns and T. Taylor, *Professional microsoft robotics developer studio*. Indianapolis, IN, EUA: John Wiley & Sons, 2009, ISBN: 978-0-470-14107-6.

[98] S.-C. Kang, W.-T. Chang, K.-Y. Gu, and H.-L. Chi, *Robot development using microsoft robotics developer studio*, 1st ed. Chapman and Hall/CRC, 2011, ISBN: 1439821658,9781439821657.

[99] ROS, *Powering the world's robots*, http://www.ros.org, Acessado Junho de 2017.

[100] R. E, S. P. N. Singh, and M. Freese, "V-rep: A versatile and scalable robot simulation framework", in *Proc. of The International Conference on Intelligent Robots and Systems (IROS)*, 2013.

[101] M. Robotics, *Microsoft robotics products and user guide*, `https://msdn.microsoft.com/en-us/library/dd936006.aspx/`, Acessado Agosto de 2017.

[102] M. Quigley, B. Gerkey, and W. D. Smart, *Programming robots with ros: A practical introduction to the robot operating system*, 1st ed. O'Reilly Media, 2015, ISBN: 978-1-44932-389-9.

[103] A. Martinez and E. Fernandez, *Learning ros for robotics programming: A practical, instructive, and comprehensive guide to introduce yourself to ros, the top-notch, leading robotics framework*. Birmingham, Reino Unido: Packt Publishing, 2013, ISBN: 978-1-78216-144-8.

[104] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "Ros: An open-source robot operating system",

[105] S. W. Newman, *A systematic approach to learning robot programming with ros*. Boca Raton: CRC Press Taylor & Francis Group, 2017, ISBN: 978-1-4987-7782-7.

[106] R. Wiki, *Tutorials*, `http://wiki.ros.org/ROS/Tutorials/UnderstandingTopics`, Acessado Julho de 2017.

[107] R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)", in *IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION (ICRA)*, Shanghai, China, May 2011.

[108] PCL, *Point cloud library*, `http://pointclouds.org/`, Acessado Julho de 2017.

[109] E. P. Ltd, *Create a digital copy of the real world.* `http://www.euclideon.com/`, Acessado Julho de 2017.

[110] MeshLab, `http://www.meshlab.net/`, Acessado Julho de 2017.

[111] CloudCompare, *3d point cloud and mesh processing software open source project*, `http://www.danielgm.net/cc/`, Acessado Julho de 2017.

[112] L. Spinello and K. O. Arras, "People detection in rgb-d data", in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sep. 2011, pp. 3838–3843. DOI: 10.1109/IROS.2011.6095074.

[113] F. Endres, J. Hess, J. Sturm, D. Cremers, and W. Burgard, "3-d mapping with an rgb-d camera", *IEEE Transactions on Robotics*, vol. 30, no. 1, pp. 177–187, Feb. 2014, ISSN: 1552-3098. DOI: 10.1109/TRO.2013.2279412.

[114] L. Shao, J. Han, D. Xu, and J. Shotton, "Computer vision for rgb-d sensors: Kinect and its applications [special issue intro.]", *IEEE Transactions on Cybernetics*, vol. 43, no. 5, pp. 1314–1317, Oct. 2013, ISSN: 2168-2267. DOI: 10.1109/TCYB.2013.2276144.

[115] T. F. Cordeiro, *Sistema de deteção de obstáculos para robótica móvel baseada em sensor kinect*, http://hdl.handle.net/10198/10497, 2014.

[116] A. H. R. Costa and J. O. Jr, "Interação de robô no ambiente",

[117] P. Mathias and J. A. Davis, *As primeiras revoluções industriais*. Lisboa, Portugal: Publicações Dom Quixote, Lda, 1993, ISBN: 972-20-1069-7.

[118] F. Duarte, *Arquitetura e tecnologias de informação: Da revolução industrial à revolução digital*, ser. Selo universidade. Annablume, 1999, ISBN: 9788574190624. [Online]. Available: https://books.google.com.br/books?id=AgLyViiB65UC.

[119] C. Vicentino and G. Dorigo, *História e do brasil volume 2*, 2 edição. São Paulo, Brasil: Scipione, 2013, ISBN: 978-85262-9121-8.

[120] M. up in Britain, *Steam engine*, http://madeupinbritain.uk/Steam_Engine, Acessado Julho de 2017.

[121] Abimaq, *A história das máquinas abimaq 70 anos*. São Paulo, Brasil: Magma Cultural e Editora, 2006.

[122] C. Vicentino and G. Dorigo, *História e do brasil volume 3*, 2 edição. São Paulo, Brasil: Scipione, 2013, ISBN: 978-85262-9123-2.

[123] L. Coutinho, "A terceira revolução industrial e tecnológica. as grandes tendências das mudanças", *Economia e Sociedade*, vol. 1, no. 1, pp. 69–87, 2016, ISSN: 1982-3533. [Online]. Available: https://periodicos.sbu.unicamp.br/ojs/index.php/ecos/article/view/8643306.

[124] K. Schwab, *The fourth industrial revolution*. The Fourth Industrial Revolution, 2016, ISBN: 978-1-944835-01-9.

[125] M. Hermann, T. Pentek, and B. Otto, "Design principles for industrie 4.0 scenarios", in *System Sciences (HICSS), 2016 49th Hawaii International Conference on*, IEEE, 2016, pp. 3928–3937.

[126] S. K. Khaitan and J. D. McCalley, "Design techniques and applications of cyberphysical systems: A survey", *IEEE Systems Journal*, vol. 9, no. 2, pp. 350–365, Jun. 2015, ISSN: 1932-8184. DOI: 10.1109/JSYST.2014.2322503.

[127] U. Robotics, "User manual ur5 original instructions", Tech. Rep., Aug. 2017.

[128] K. Mathiassen, J. E. Fjellin, K. Glette, P. K. Hol, and O. J. Elle, "An ultrasound robotic system using the commercial robot ur5", *Frontiers in Robotics and AI*, vol. 3, p. 1, 2016, ISSN: 2296-9144. DOI: 10.3389/frobt.2016.00001. [Online]. Available: https://www.frontiersin.org/article/10.3389/frobt.2016.00001.

[129] I. A. Sucan and S. Chitta, *Moveit*, http://moveit.ros.org/, Acessado Agosto de 2017.

[130] A. Mahtani, L. Sanchez, E. Fernandez, and A. Martinez, *Effective robotics programming with ros*, 3rd. Packt Publishing, 2016, ISBN: 978-1786463654.

[131] C. Ma, Y. Zhang, Q. Zhao, and K. Bai, "6r serial manipulator space path planning based on rrt", in *2016 8th International Conference on Intelligent Human-Machine Systems and Cybernetics (IHMSC)*, vol. 02, Aug. 2016, pp. 99–102. DOI: 10.1109/IHMSC.2016.90.

[132] D. Catuhe, *Programming with the kinect for windows software development kit*. Redmond, Washington: Microsoft Press, 2012, ISBN: 978-0-7356-6681-8.

[133] G. S. Cardoso, *Microsoft kinect: Crie aplicações interativas*. Casa do Codigo, 2014, ISBN: 978-85-66250-21-3.

[134] T. F. Carrera, *Movimentos reativos no robô turtlebot utilizando o kinect*, `http://hdl.handle.net/10198/11982`, 2013.

[135] R. Miles, *Start here! learn the kinect api*. Sebastopol, California: Microsoft Press, 2012, ISBN: 978-0-735-66396-1.

[136] PCL, *Point cloud library documentation*, `http://docs.pointclouds.org/trunk/group__filters.html`, Acessado Julho de 2017.

[137] ——, *Point cloud library documentation*, `http://docs.pointclouds.org/trunk/group__features.html`, Acessado Julho de 2017.

[138] ——, *Point cloud library documentation*, `http://docs.pointclouds.org/trunk/group__registration.html`, Acessado Julho de 2017.

[139] V. Lamoine, *Interactive iterative closest point*, `http://pointclouds.org/documentation/tutorials/interactive_icp.php`, Acessado Julho de 2017.

[140] PCL, *Point cloud library documentation*, `http://docs.pointclouds.org/trunk/group__kdtree.html`, Acessado Julho de 2017.

[141] ——, *Point cloud library documentation*, `http://docs.pointclouds.org/trunk/group__octree.html`, Acessado Julho de 2017.

[142] S. Ushakov, *Implicit shape model*, `http://pointclouds.org/documentation/tutorials/implicit_shape_model.php`, Acessado Julho de 2017.

[143] PCL, *Point cloud library documentation*, `http://docs.pointclouds.org/trunk/group__surface.html`, Acessado Julho de 2017.

[144] N. Blodow, *The openni grabber framework in pcl*, `http://pointclouds.org/documentation/tutorials/openni_grabber.php`, Acessado Julho de 2017.

[145] PCL, *Point cloud library documentation*, http://docs.pointclouds.org/trunk/group__visualization.html, Acessado Julho de 2017.

[146] W. Darwish, S. Tang, W. Li, and W. Chen, "A new calibration method for commercial rgbd sensors", *Sensors*, no. 6, 2017, ISSN: 1424-8220. DOI: 10.3390s17061204. [Online]. Available: http://www.mdpi.com/1424-8220/17/6/1204.

[147] J. Bowman and P. Mihelich, *Camera calibration package*, http://wiki.ros.org/camera_calibration, Acessado Dezembro de 2017.

[148] *The opencv reference manual*, 2.4.9.0, Itseez, Apr. 2014.

Appendix A

Trigonometric Functions

This Appendix A presents some summaries of trigonometric functions, an important requirement for understanding the study of kinematics of manipulators. [53].

Trigonometria

A.1 - Semelhança de triângulos

Dois triângulos são semelhantes quando possuem dois ângulos iguais. Como a soma dos ângulos internos de um triângulo é sempre igual a 180 ^{o}, então todos os ângulos de triângulos semelhantes são iguais

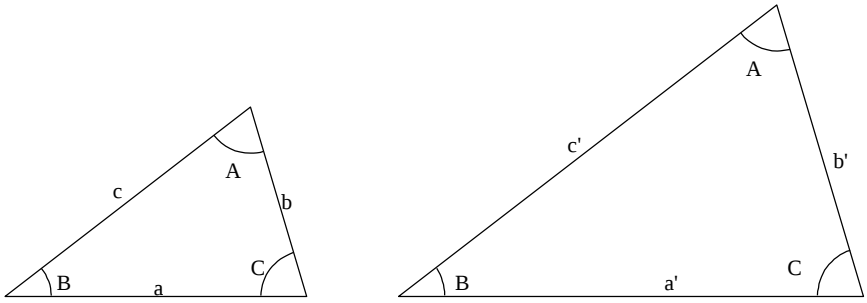


Fig. A.1 – Semelhança de triângulos

Nos triângulos semelhantes valem as regras de proporcionalidade:

$$\frac{a}{a'} = \frac{b}{b'} = \frac{c}{c'}$$

A.2 - Teorema de Pitágoras

Num triângulo retângulo OPQ, no qual o ângulo do vértice Q é reto (igual a 90 ^{o} ou $\pi/2$) e o ângulo do vértice O é α, o cateto oposto é definido como o comprimento b da aresta PQ, o cateto adjacente é definido como o comprimento a da aresta OQ, e a hipotenusa é o comprimento c da maior aresta, OP. O teorema de Pitágoras fornece que o quadrado da hipotenusa é igual à soma dos quadrados dos catetos, ou seja

$$c^2 = a^2 + b^2$$

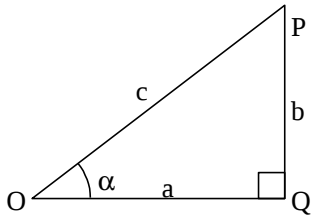


Fig. A.2 – Triângulo retângulo

A.3 - Seno, co-seno e tangente

Numa circunferência de raio unitário desenhamos um sistema de eixos passando pelo centro O da circunferência e um arco de círculo de ângulo α definido pelo ponto P. A

projeção deste ponto nos eixos das abscissas e das ordenadas define os pontos Q e R, respectivamente.

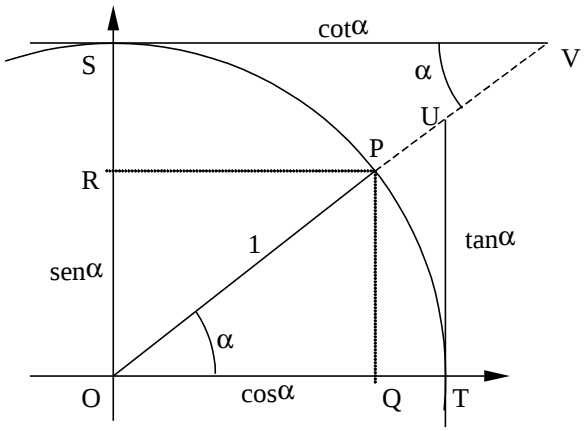


Fig. A.3 – Círculo de raio unitário: seno, co-seno, tangente, co-tangente, secante e co-secante.

O seno deste ângulo é definido como o comprimento do cateto oposto ao triângulo OPQ, ou seja, ao comprimento RO ou PQ. Da mesma forma, o co-seno é o comprimento do cateto adjacente ao ângulo α, cujo comprimento é RP ou OQ. A tangente é medida ao longo da reta paralela ao eixo das ordenadas que tangencia a circunferência no ponto T em que esta encontra o eixo das abscissas, até o ponto U em que esta encontra o prolongamento de OP. Analogamente, a co-tangente do ângulo α é o comprimento medido ao longo da reta paralela ao eixo das abscissas que passa pelo ponto S, encontro da circunferência com o eixo das ordenadas, até o ponto V em que esta reta encontra o prolongamento de OP. Tem-se, finalmente, a secante sendo dada pelo comprimento OU e a co-secante por OV. Resumidamente,

$$\operatorname{sen}\alpha = OQ$$
$$\cos\alpha = OR$$
$$\tan\alpha = TU$$
$$\cot\alpha = SV$$
$$\sec\alpha = OU$$
$$\operatorname{csc}\alpha = OV$$

Do teorema de Pitágoras segue imediatamente que

$$\cos^2\alpha + \operatorname{sen}^2\alpha = 1$$

Da relação de semelhança entre triângulos, tem-se igualmente que

$$\tan\alpha = \frac{\operatorname{sen}\alpha}{\cos\alpha},$$

$$\cot\alpha = \frac{1}{\tan\alpha} = \frac{\cos\alpha}{\operatorname{sen}\alpha}$$

$$\sec\alpha = \frac{1}{\cos\alpha}$$

$$\csc\alpha = \frac{1}{\operatorname{sen}\alpha}$$

A.4 - Complementos de ângulos

Supondo conhecidos o seno e o co-seno do ângulo α, deseja-se saber o valor do seno e co-seno do ângulo complementar $180^{o} - \alpha$ ou $\pi - \alpha$. Conforme indica a figura abaixo, estes podem ser obtidos da semelhança entre os triângulos OAC e OBD:

$$\cos(\pi-\alpha) = -\cos\alpha$$
$$\operatorname{sen}(\pi-\alpha) = \operatorname{sen}\alpha$$

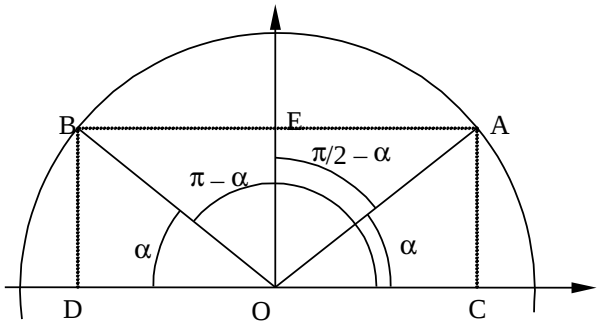


Fig. A.4 – Ângulos complementares.

Da mesma figura pode-se ainda verificar que, da semelhança entre os triângulos OAC e OEA,

$$\cos(\pi/2 - \alpha) = \operatorname{sen}\alpha$$
$$\operatorname{sen}(\pi/2 - \alpha) = \cos\alpha$$

Igualmente, da figura abaixo tem-se que

$$\cos(-\alpha) = \cos\alpha$$
$$\operatorname{sen}(-\alpha) = -\operatorname{sen}\alpha$$

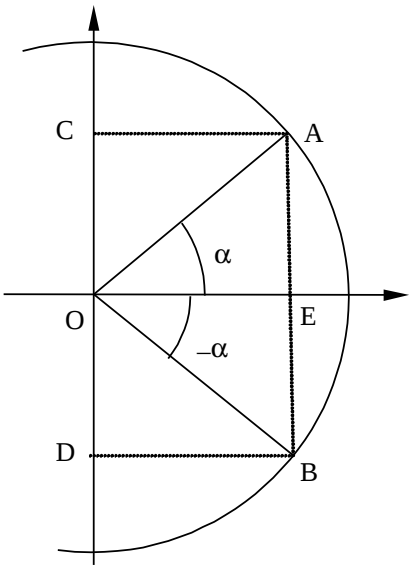


Fig. A.5 – Ângulos recíprocos.

Figuras semelhantes a essas podem ser desenhadas para se obter as igualdades:

$$\cos(\pi/2 + \alpha) = \operatorname{sen}\alpha$$
$$\operatorname{sen}(\pi/2 + \alpha) = \cos\alpha$$

e

$$\cos(\pi + \alpha) = \cos\alpha$$
$$\operatorname{sen}(\pi + \alpha) = \operatorname{sen}\alpha$$

As seguintes relações são válidas para a tangente:

$$\tan(-\alpha) = \tan\alpha$$
$$\tan(\pi/2 - \alpha) = \cot\alpha$$
$$\tan(\pi/2 + \alpha) = \cot\alpha$$
$$\tan(\pi - \alpha) = \tan\alpha$$
$$\tan(\pi + \alpha) = \tan\alpha$$

Pode-se exprimir as funções trigonométricas do seno, co-seno e tangente em termos delas próprias. Do teorema de Pitágoras, por exemplo, tira-se que

$$\cos\alpha = \pm\sqrt{1 - \operatorname{sen}^2\alpha}$$
$$\operatorname{sen}\alpha = \pm\sqrt{1 - \cos^2\alpha}$$

A tangente pode ser obtida de

$$\tan\alpha = \pm\frac{\operatorname{sen}\alpha}{\sqrt{1-\operatorname{sen}^2\alpha}} = \pm\frac{\sqrt{1-\cos^2\alpha}}{\cos\alpha}.$$

Esta última pode ser invertida fornecendo as expressões

$$\operatorname{sen}\alpha = \pm\frac{\tan\alpha}{\sqrt{1+\tan^2\alpha}}$$

$$\cos\alpha = \pm\frac{1}{\sqrt{1+\tan^2\alpha}}$$

Em todas as expressões acima o quadrante do ângulo deve ser determinado para se verificar se a solução correta é a positiva ou negativa.

A.5 - Soma e diferença de ângulos

Em geometria é comum a necessidade de se conhecer o seno ou o co-seno da soma de dois ângulos. Pode-se relacionar o seno (ou o co-seno) da soma com os senos e co-senos dos ângulos individuais. Na figura abaixo, nota-se que $BC = \operatorname{sen}\alpha$ e portanto $FG = BC\operatorname{sen}\beta = \operatorname{sen}\alpha\operatorname{sen}\beta$. Da mesma forma, $OC = \cos\alpha$ e $OG = \cos\alpha\cos\beta$. Percebe-se também que $OF = OG - FG$, de onde

$$\cos(\alpha+\beta) = \cos\alpha\cos\beta - \operatorname{sen}\alpha\operatorname{sen}\beta$$

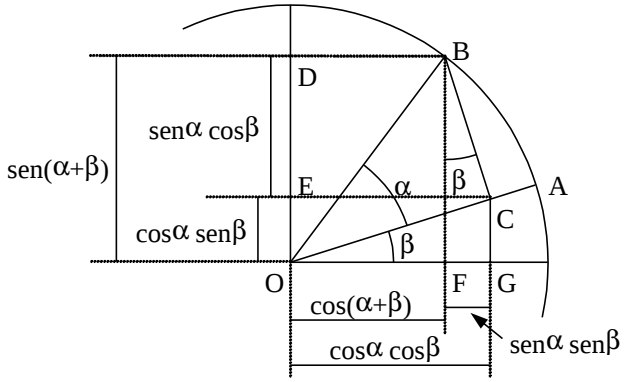


Fig. A.6 – Seno e co-seno da soma de dois ângulos

De forma análoga, tem-se que $DE = BC\cos\beta = \operatorname{sen}\alpha\cos\beta$, e $OE = OC\operatorname{sen}\beta = \cos\alpha\operatorname{sen}\beta$. Como $OD = DE + OE$, então seno da soma fica dado por

$$\operatorname{sen}(\alpha+\beta) = \operatorname{sen}\alpha\cos\beta + \cos\alpha\operatorname{sen}\beta$$

Sabendo-se que $\cos(-\beta) = \cos\beta$ e que $\operatorname{sen}(-\beta) = -\operatorname{sen}\beta$, tem-se que a o seno e o co-seno da diferença entre os ângulos valem

$$\cos(\alpha-\beta) = \cos\alpha\cos\beta + \operatorname{sen}\alpha\operatorname{sen}\beta$$

$$\operatorname{sen}(\alpha-\beta) = \operatorname{sen}\alpha\cos\beta - \cos\alpha\operatorname{sen}\beta$$

Embora seja também possível obter uma solução geométrica para a tangente da soma, é mais fácil neste ponto calcular pela relação entre o seno e o co-seno, ou seja:

$$\tan(\alpha+\beta) = \frac{\operatorname{sen}(\alpha+\beta)}{\cos(\alpha+\beta)} = \frac{\tan\alpha + \tan\beta}{1 - \tan\alpha\,\tan\beta},$$

e

$$\tan(\alpha-\beta) = \frac{\operatorname{sen}(\alpha-\beta)}{\cos(\alpha-\beta)} = \frac{\tan\alpha - \tan\beta}{1 + \tan\alpha\,\tan\beta},$$

A.6 - Lei dos senos

Num triângulo qualquer ABC, de ângulos α, β, e γ, e de lados a, b e c, traça-se uma reta a partir do vértice A perpendicular ao lado BC, como mostra a figura abaixo. No triângulo retângulo formado por ABH, o comprimento AH é igual $c \operatorname{sen}\beta$, e no triângulo AHC, este mesmo comprimento é dado por $b \operatorname{sen}\gamma$. Traçando-se agora uma reta a partir de C perpendicular ao lado AB, tem-se igualmente que $CG = a \operatorname{sen}\beta = b \operatorname{sen}\alpha$. Repetindo-se o processo com o vértice B e o lado AC, ter-se-á igualmente que $a \operatorname{sen}\gamma = c \operatorname{sen}\alpha$. Estas igualdades permitem escrever a lei dos senos:

$$\frac{\operatorname{sen}\alpha}{a} = \frac{\operatorname{sen}\beta}{b} = \frac{\operatorname{sen}\gamma}{c}, \text{ ou } \frac{a}{\operatorname{sen}\alpha} = \frac{b}{\operatorname{sen}\beta} = \frac{c}{\operatorname{sen}\gamma}$$

e que pode ser estabelecida como: “num triângulo qualquer, a relação entre o comprimento de qualquer de seus lados com relação ao seno do ângulo oposto a ele é uma constante”.

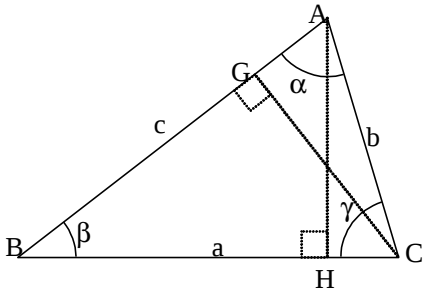


Fig. A.7 – Lei dos senos.

A.7 - Lei dos co-senos

Num triângulo qualquer ABC, traça-se uma a reta que, a partir do vértice A, encontra o lado BC em ângulo reto (perpendicular a BC), como mostra a figura abaixo. No triângulo retângulo ABH, aplica-se o teorema de Pitágoras, obtendo-se

$$c^2 = BH^2 + AH^2$$

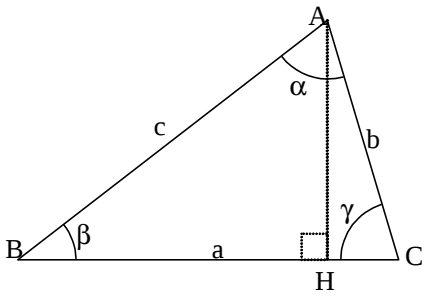


Fig. A.8 – Lei dos co-senos.

Da mesma forma, o triângulo retângulo AHC fornece

$$b^2 = HC^2 + AH^2$$

Isolando-se o lado AH das expressões acima e igualando-as tem-se que

$$c^2 = b^2 + BH^2 - HC^2$$

Porém, lembrando que $HC = b\cos\gamma$ e que $AH = a - HC$, substituindo-se estes valores na última resulta que

$$c^2 = b^2 + a^2 - 2ab\cos\gamma .$$

A lei dos co-senos pode então ser definida como "num triângulo qualquer, o quadrado de um dos lados é igual à soma dos quadrados dos demais, subtraído do duplo produto destes lados pelo co-seno do ângulo entre eles". Uma vez que não foi estabelecida nenhuma condição sobre um dos lados, tem-se igualmente que

$$a^2 = b^2 + c^2 - 2bc\cos\alpha .$$

e

$$b^2 = a^2 + c^2 - 2ac\cos\beta .$$

Appendix B

Essential ROS Commands

This Appendix B has some ROS commands, an important requirement in using this software [99].

ROS Indigo Cheatsheet

Filesystem Management Tools

rospack	A tool for inspecting packages.
rospack profile	Fixes path and pluginlib problems.
roscd	Change directory to a package.
rospd/rosd	Pushd equivalent for ROS.
rosls	Lists package or stack information.
rosed	Open requested ROS file in a text editor.
roscp	Copy a file from one place to another.
rosdep	Installs package system dependencies.
roswtf	Displays a errors and warnings about a running ROS system or launch file.
catkin_create_pkg	Creates a new ROS stack.
wstool	Manage many repos in workspace.
catkin_make	Builds a ROS catkin workspace.
rqt_dep	Displays package structure and dependencies.

Usage:

```
$ rospack find [package]
$ roscd [package[/subdir]]
$ rospd [package[/subdir] | +N | -N]
$ rosd
$ rosls [package[/subdir]]
$ rosed [package] [file]
$ roscp [package] [file] [destination]
$ rosdep install [package]
$ roswtf or roswtf [file]
$ catkin_create_pkg [package_name] [depend1]..[dependN]
$ wstool [init | set | update]
$ catkin_make
$ rqt_dep [options]
```

Start-up and Process Launch Tools

roscore

The basis nodes and programs for ROS-based systems. A roscore must be running for ROS nodes to communicate.

Usage:

```
$ roscore
```

rosrun

Runs a ROS package's executable with minimal typing.

Usage:

```
$ rosrun package_name executable_name
```

Example (runs turtlesim):

```
$ rosrun turtlesim turtlesim_node
```

roslaunch

Starts a roscore (if needed), local nodes, remote nodes via SSH, and sets parameter server parameters.

Examples:

Launch a file in a package:

```
$ roslaunch package_name file_name.launch
```

Launch on a different port:

```
$ roslaunch -p 1234 package_name file_name.launch
```

Launch on the local nodes:

```
$ roslaunch --local package_name file_name.launch
```

Logging Tools

rosbag

A set of tools for recording and playing back of ROS topics.

Commands:

rosbag record	Record a bag file with specified topics.
rosbag play	Play content of one or more bag files.
rosbag compress	Compress one or more bag files.
rosbag decompress	Decompress one or more bag files.
rosbag filter	Filter the contents of the bag.

Examples:

Record select topics:

```
$ rosbag record topic1 topic2
```

Replay all messages without waiting:

```
$ rosbag play -a demo_log.bag
```

Replay several bag files at once:

```
$ rosbag play demo1.bag demo2.bag
```

Introspection and Command Tools

rosmsg/rossrv

Displays Message/Service (msg/srv) data structure definitions.

Commands:

rosmsg show	Display the fields in the msg/srv.
rosmsg list	Display names of all msg/srv.
rosmsg md5	Display the msg/srv md5 sum.
rosmsg package	List all the msg/srv in a package.
rosmsg packages	List all packages containing the msg/srv.

Examples:

Display the Pose msg:

```
$ rosmsg show Pose
```

List the messages in the nav_msgs package:

```
$ rosmsg package nav_msgs
```

List the packages using sensor_msgs/CameraInfo:

```
$ rosmsg packages sensor_msgs/CameraInfo
```

rosnode

Displays debugging information about ROS nodes, including publications, subscriptions and connections.

Commands:

rosnode ping	Test connectivity to node.
rosnode list	List active nodes.
rosnode info	Print information about a node.
rosnode machine	List nodes running on a machine.
rosnode kill	Kill a running node.

Examples:

Kill all nodes:

```
$ rosnode kill -a
```

List nodes on a machine:

```
$ rosnode machine aqy.local
```

Ping all nodes:

```
$ rosnode ping --all
```

rostopic

A tool for displaying information about ROS topics, including publishers, subscribers, publishing rate, and messages.

Commands:

rostopic bw	Display bandwidth used by topic.
rostopic echo	Print messages to screen.
rostopic find	Find topics by type.
rostopic hz	Display publishing rate of topic.
rostopic info	Print information about an active topic.
rostopic list	List all published topics.
rostopic pub	Publish data to topic.
rostopic type	Print topic type.

Examples:

Publish hello at 10 Hz:

```
$ rostopic pub -r 10 /topic_name std_msgs/String hello
```

Clear the screen after each message is published:

```
$ rostopic echo -c /topic_name
```

Display messages that match a given Python expression:

```
$ rostopic echo --filter "m.data=='foo'" /topic_name
```

Pipe the output of rostopic to rosmsg to view the msg type:

```
$ rostopic type /topic_name | rosmsg show
```

rosparam

A tool for getting and setting ROS parameters on the parameter server using YAML-encoded files.

Commands:

rosparam set	Set a parameter.
rosparam get	Get a parameter.
rosparam load	Load parameters from a file.
rosparam dump	Dump parameters to a file.
rosparam delete	Delete a parameter.
rosparam list	List parameter names.

Examples:

List all the parameters in a namespace:

```
$ rosparam list /namespace
```

Setting a list with one as a string, integer, and float:

```
$ rosparam set /foo "['1', 1, 1.0]"
```

Dump only the parameters in a specific namespace to file:

```
$ rosparam dump dump.yaml /namespace
```

rosservice

A tool for listing and querying ROS services.

Commands:

rosservice list	Print information about active services.
rosservice node	Print name of node providing a service.
rosservice call	Call the service with the given args.
rosservice args	List the arguments of a service.
rosservice type	Print the service type.
rosservice uri	Print the service ROSRPC uri.
rosservice find	Find services by service type.

Examples:

Call a service from the command-line:

```
$ rosservice call /add_two_ints 1 2
```

Pipe the output of rosservice to rossrv to view the srv type:

```
$ rosservice type add_two_ints | rossrv show
```

Display all services of a particular type:

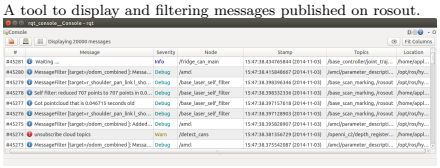
```
$ rosservice find rospy_tutorials/AddTwoInts
```

ROS Indigo Cheatsheet

Logging Tools

rqt_console

A tool to display and filtering messages published on rosout.

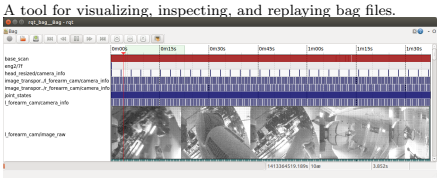


Usage:

```
$ rqt_console
```

rqt_bag

A tool for visualizing, inspecting, and replaying bag files.



Usage, viewing:

```
$ rqt_bag bag_file.bag
```

Usage, bagging:

```
$ rqt_bag *press the big red record button.*
```

rqt_logger_level

Change the logger level of ROS nodes. This will increase or decrease the information they log to the screen and rqt_console.

Usage:

```
viewing $ rqt_logger_level
```

Introspection & Command Tools

rqt_topic

A tool for viewing published topics in real time.

Usage:

```
$ rqt
Plugin Menu->Topic->Topic Monitor
```

rqt_msg, rqt_srv, and rqt_action

A tool for viewing available msgs, srvs, and actions.

Usage:

```
$ rqt
Plugin Menu->Topic->Message Type Browser
Plugin Menu->Service->Service Type Browser
Plugin Menu->Action->Action Type Browser
```

rqt_publisher, and rqt_service_caller

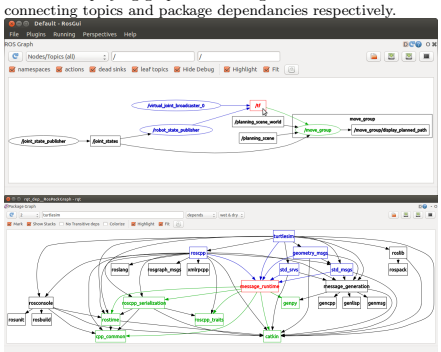
Tools for publishing messages and calling services.

Usage:

```
$ rqt
Plugin Menu->Topic->Message Publisher
Plugin Menu->Service->Service Caller
```

rqt_graph, and rqt_dep

Tools for displaying graphs of running ROS nodes with connecting topics and package dependancies respectively.



Usage:

```
$ rqt_graph
$ rqt_dep
```

rqt_top

A tool for ROS specific process monitoring.

Usage:

```
$ rqt
Plugin Menu->Introspection->Process Monitor
```

rqt_reconfigure

A tool for dynamically reconfiguring ROS parameters.

Usage:

```
$ rqt
Plugin Menu->Configuration->Dynamic Reconfigure
```

Development Environments

rqt_shell, and rqt_py_console

Two tools for accessing an xterm shell and python console respectively.

Usage:

```
$ rqt
Plugin Menu->Miscellaneous Tools->Shell
Plugin Menu->Miscellaneous Tools->Python Console
```

Data Visualization Tools

tf_echo

A tool that prints the information about a particular transformation between a source_frame and a target_frame.

Usage:

```
$ rosrun tf tf_echo <source_frame> <target_frame>
```

Examples:

To echo the transform between /map and /odom:

```
$ rosrun tf tf_echo /map /odom
```

view_frames

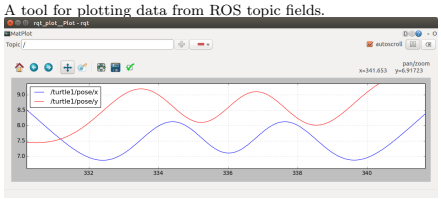
A tool for visualizing the full tree of coordinate transforms.

Usage:

```
$ rosrun tf2_tools view_frames.py
$ evince frames.pdf
```

rqt_plot

A tool for plotting data from ROS topic fields.



Examples:

To graph the data in different plots:

```
$ rqt_plot /topic1/field1 /topic2/field2
```

To graph the data all on the same plot:

```
$ rqt_plot /topic1/field1,/topic2/field2
```

To graph multiple fields of a message:

```
$ rqt_plot /topic1/field1:field2:field3
```

rqt_image_view

A tool to display image topics.

Usage:

```
$ rqt_image_view
```

ROS Indigo Catkin Workspaces

Create a catkin workspace

Setup and use a new catkin workspace from scratch.

Example:

```
$ source /opt/ros/hydro/setup.bash
$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws/src
$ catkin_init_workspace
```

Checkout an existing ROS package

Get a local copy of the code for an existing package and keep it up to date using wstool.

Examples:

```
$ cd ~/catkin_ws/src
$ wstool init
$ wstool set tutorials --git git://github.com/ros/ros_tutorials.git
$ wstool update
```

Create a new catkin ROS package

Create a new ROS catkin package in an existing workspace with catkin create package. After using this you will need to edit the CMakeLists.txt to detail how you want your package built and add information to your package.xml.

Usage:

```
$ catkin_create_pkg <package_name> [depend1] [depend2]
```

Example:

```
$ cd ~/catkin_ws/src
$ catkin_create_pkg tutorials std_msgs rospy roscpp
```

Build all packages in a workspace

Use catkin_make to build all the packages in the workspace and then source the setup.bash to add the workspace to the ROS_PACKAGE_PATH.

Examples:

```
$ cd ~/catkin_ws
$ ~/catkin_make
$ source devel/setup.bash
```

Appendix C

Manual of UR5 and Jaco2 arms

This Appendix C exposes the manuals provided by the manufacturers of the robotic arms used, in simulations (UR5) and reality (Jaco2 6 DOF).

C.1 UR5 arm Manual

UNIVERSAL ROBOTS

Detalhes técnicos

UR5

Desempenho

Repetibilidade	± 0,1 mm / ± 0,0039 pol (4 mils)
Temperatura de trabalho	0-50°
Consumo de energia	Mín. 90W, Normal 150W, Máx. 325W
Possibilidades de colaboração	15 funções de segurança ajustáveis avançadas Função de segurança aprovada pelo TüV NORD Testado de acordo com: EN ISO 13849:2008 PL d

Especificação

Carga	5 kg / 11 lbs
Alcance	850 mm / 33,5 pol
Graus de mobilidade	6 juntas rotativas
Programação	Interfaz gráfica del usuario PolyScope con pantalla táctil de 12" in con soporte

Movimento

Movimento dos eixos do robô	Gama de trabalho	Velocidade máxima
Base	± 360°	± 180°/seg.
Ombro	± 360°	± 180°/seg.
Cotovelo	± 360°	± 180°/seg.
Punho 1	± 360°	± 180°/seg.
Punho 2	± 360°	± 180°/seg.
Punho 3	± 360°	± 180°/seg.
Ferramenta tradicional		1 m/seg. / 39,4 pol/seg.

Recursos

Classificação IP	IP54	
Sala Limpa ISO Classe	5	
Ruído	72dB	
Montagem do robô	Qualquer	
Portas de E/S	Entradas digitais	2
	Saídas digitais	2
	Entradas analógicas	2
	Saídas analógicas	0
Fonte de alimentação de E/S na ferramenta	12 V/24 V 600 mA em ferramenta	

Física

Área ocupada	Ø149 mm
Materiais	Alumínio, plásticos PP
Tipo de conector da ferramenta	M8
Comprimento do cabo do robô	6 m / 236 pol
Peso com o cabo	18,4 kg / 40,6 lbs

PAINEL DE CONTROLE

Recursos

Classificação IP	IP20	
Sala Limpa ISO Classe	6	
Ruído	< 65 dB(A)	
Portas de E/S	Entradas digitais	16
	Saídas digitais	16
	Entradas analógicas	2
	Saídas analógicas	2
Fonte de alimentação de E/S	24V 2A	
Comunicação	TCP/IP 100Mbit, Modbus TCP, Profinet, EthernetIP	
Fonte de energia	100-240 VAC, 50-60 Hz	
Temperatura de trabalho	0-50°	

Física

Tamanho da caixa de controle (LxAxD)	475 mm x 423 mm x 268 mm / 18,7 x 16,7 x 10,6 pol.
Peso	15 kg / 33,1 lbs
Materiais	Aço

PAINEL DE CONTROLE

Recursos

Classificação IP	IP20

Física

Materiais	Alumínio, PP
Peso	1,5 kg
Comprimento do cabo	4,5 mm / 177 pol

C.2 Jaco2 6DOF arm Manual

JACO²

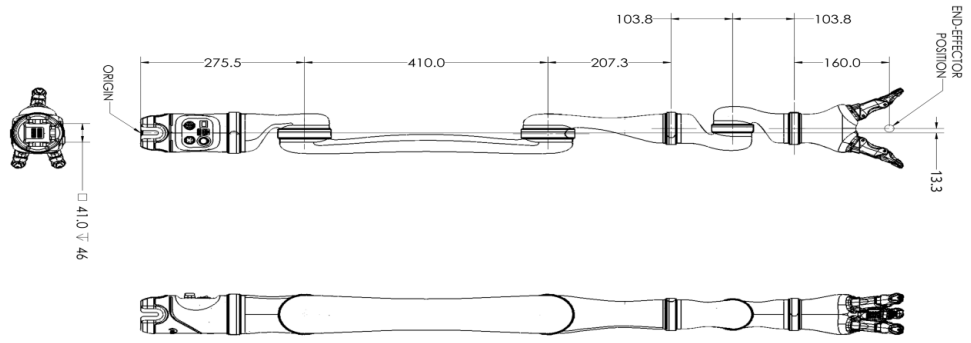
Spherical 6 DOF

Technical specifications

KINOVA ROBOTICS

kinovarobotics.com

JACO² Spherical 6 DOF



GENERAL

TOTAL WEIGHT	4,4 Kg
MATERIALS	Carbon fiber (links), Aluminum (actuators)
PAYLOAD	2.6 Kg (mid-range continuous payload capabilities) 2.2 Kg (full-reach peak/temporary payload capabilities)
REACH	98.5 cm
JOINT RANGE AFTER START-UP (SOFTWARE LIMITATION)	±27.7 turns
MAXIMUM LINEAR ARM SPEED	20 cm/s
POWER SUPPLY VOLTAGE	18 to 29 VDC
AVERAGE POWER	25 W (5W in STANDBY)
PEAK POWER	100 W
COMMUNICATION PROTOCOL	RS485
COMMUNICATION CABLES	20 pins flat flex cable
2 EXPANSION PINS ON COMMUNICATION BUS	
WATER RESISTANCE	IPX2
OPERATING TEMPERATURE	-10 °C to 40 °C expected

4333 Boul. Grande- Allée, Boisbriand QC Canada J7H 1M7

CONTROLLER

PORTS	
Joystick	1 Mbps CANBUS
Power supply	18 to 29 VDC
USB 2.0 (API)	12 Mbps
Ethernet	100 Mbps
CONTROL SYSTEM FREQUENCY	100 Hz (High Level API)
	500 Hz (Low Level API)
CPU	360 MHz
SDK	
APIs	High and low level
Compatibility	Windows, Linux Ubuntu & ROS
Port	USB 2.0
Programming languages	C++
FORCE, CARTESIAN & ANGULAR CONTROL	

DETAILED SPECIFICATIONS

ACTUATORS #1, #2 & #3	**K-75+**
ACTUATORS #4, #5 & #6	**K-58**

kinovarobotics.com

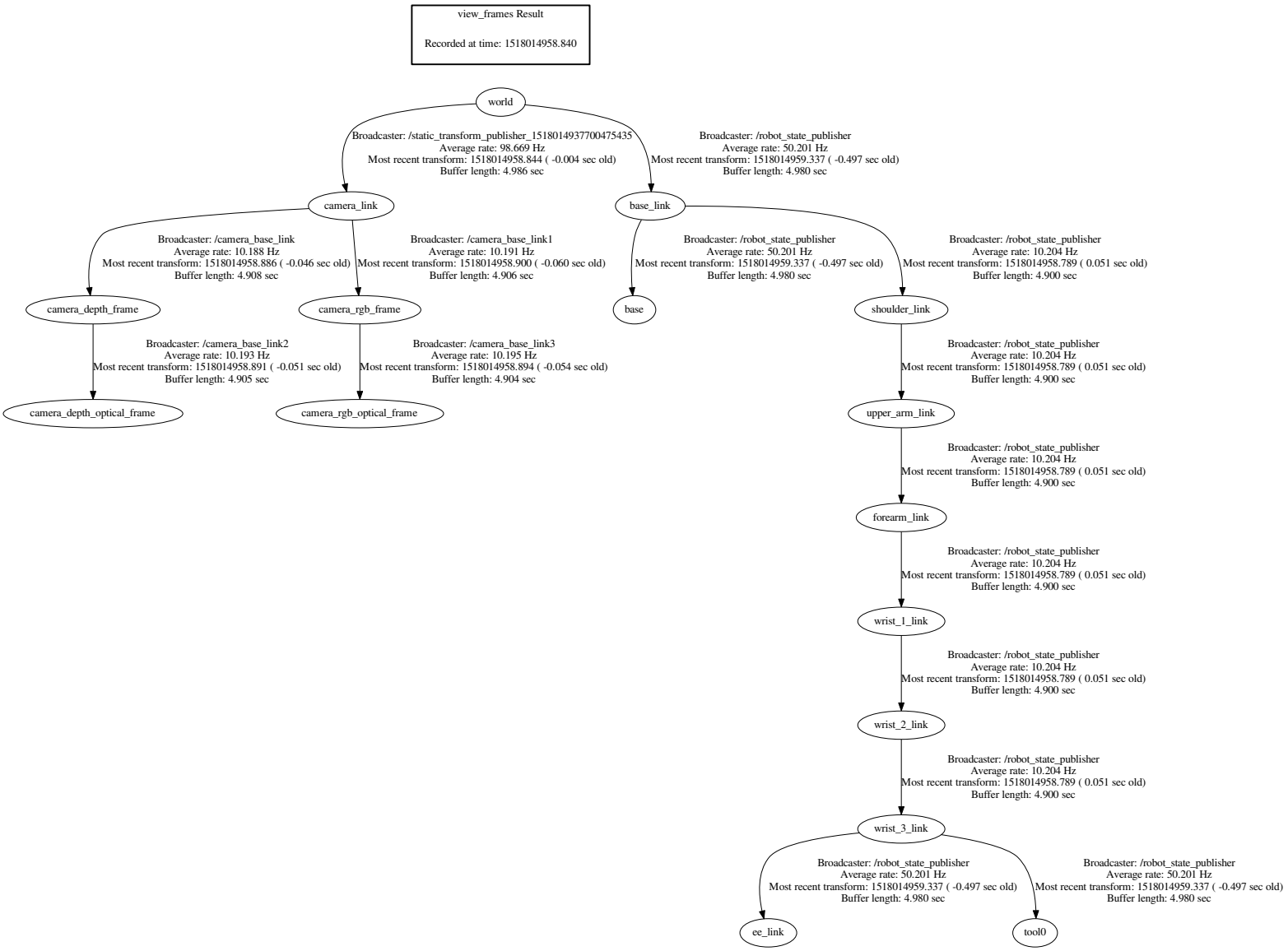
4333 Boul. Grande- Allée, Boisbriand QC Canada J7H 1M7

Appendix D

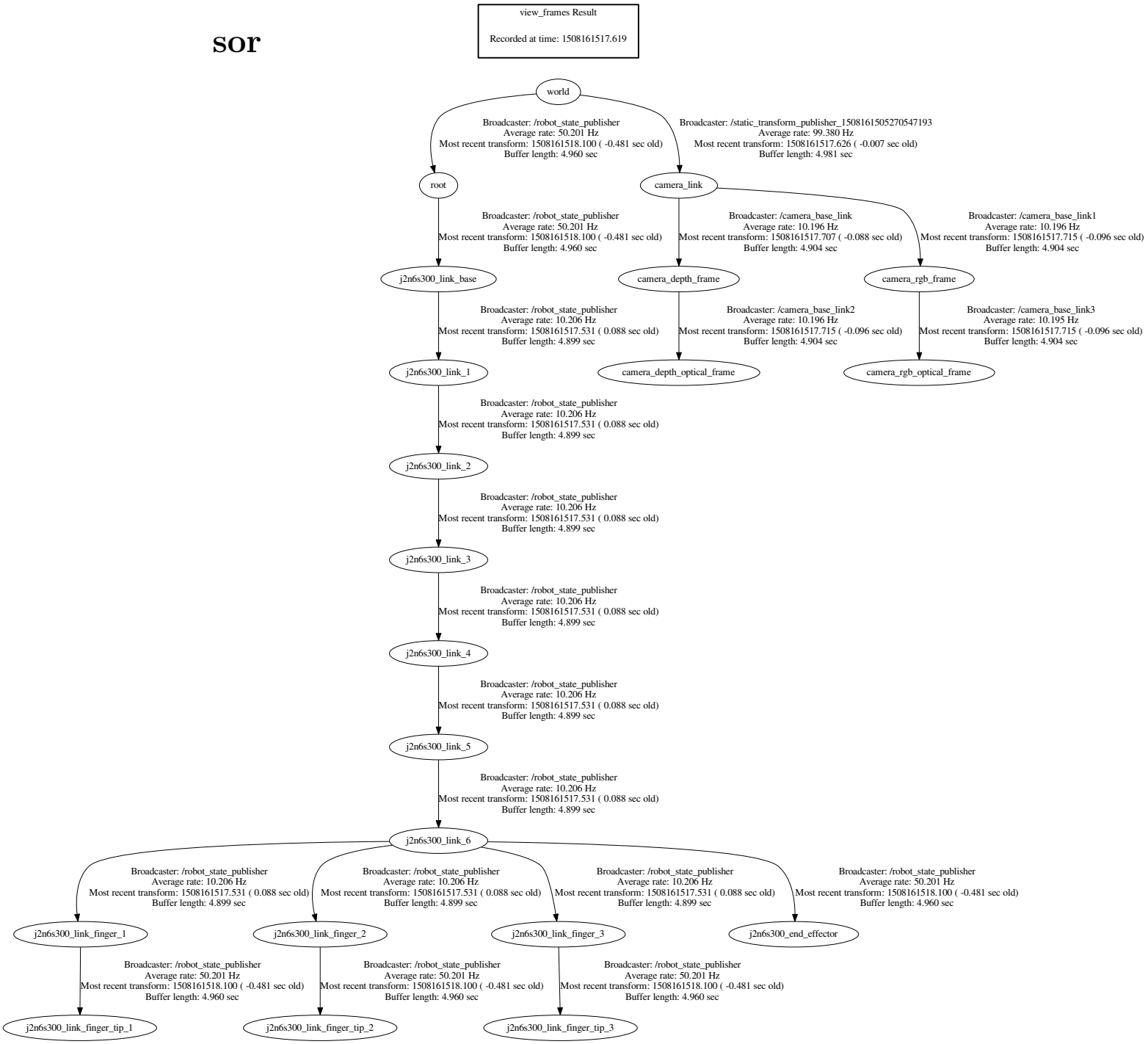
Jaco2 6DOF and UR5 arms frames with Kinect sensor

This Appendix D points to the frames generated during the configuration of the robotic arms used, in the simulations (UR5) and reality (Jaco2 6 DOF).

D.1 Frames of the UR5 arm with Kinect sensor



D.2 Frames of the Jaco2 6DOF arm with Kinect sensor



Appendix E

Description in HTML format of codes used

To start the system that avoids obstacles in UR5 handlers, simply follow the steps described here.

- Load the main program:

```
roslaunch myworkcell_moveit_config myworkcell_planning_execution.launch
```

- Load the depth map, the Cloud Points:

```
rosrun my_pcl_tutorial example input:=/camera/depth_registered/points
```

- Loads the depth map in Octree format:

```
roslaunch myworkcell_moveit_config octomap.launch
```

E.1 Code of MoveIt! and Rviz

The sequence of subsections details the codes of each file of the proposed system.

E.1.1 default_warehouse_db.launch

```
<launch>
  <arg name="reset" default="false"/>
  <arg name="moveit_warehouse_database_path"
   default="$(find myworkcell_moveit_config)/default_warehouse_mongo_db" />

  <include file="$(find myworkcell_moveit_config)/launch/warehouse.launch">
    <arg name="moveit_warehouse_database_path"
    value="$(arg moveit_warehouse_database_path)" />
  </include>

  <node if="$(arg reset)" name="$(anon moveit_default_db_reset)"
  type="moveit_init_demo_warehouse"
  pkg="moveit_ros_warehouse" respawn="false" output="screen" />
</launch>
```

E.1.2 demo.launch

```
<launch>
  <arg name="db" default="false" />
  <arg name="db_path"
   default="$(find myworkcell_moveit_config)/default_warehouse_mongo_db" />
  <arg name="debug" default="false" />
  <arg name="use_gui" default="false" />

  <include file="$(find myworkcell_moveit_config)
```

```
␣␣/launch/planning_context.launch">
    <arg name="load_robot_description" value="true"/>
  </include>

  <node name="joint_state_publisher" pkg="joint_state_publisher"
  type="joint_state_publisher">
    <param name="/use_gui" value="$(arg␣use_gui)"/>
    <rosparam param="/source_list">[/move_group/fake_controller_joint_states]
    </rosparam>
  </node>

  <node name="robot_state_publisher" pkg="robot_state_publisher"
  type="robot_state_publisher" respawn="true" output="screen" />

  <include file="$(find␣myworkcell_moveit_config)/launch/move_group.launch">
    <arg name="allow_trajectory_execution" value="true"/>
    <arg name="fake_execution" value="true"/>
    <arg name="info" value="true"/>
    <arg name="debug" value="$(arg␣debug)"/>
  </include>

  <include file="$(find␣myworkcell_moveit_config)/launch/moveit_rviz.launch">
    <arg name="config" value="true"/>
    <arg name="debug" value="$(arg␣debug)"/>
  </include>

  <include file="$(find␣myworkcell_moveit_config)
␣␣/launch/default_warehouse_db.launch" if="$(arg␣db)">
    <arg name="moveit_warehouse_database_path" value="$(arg␣db_path)"/>
```

```
  </include>
</launch>
```

E.1.3 fake_moveit_controller_manager.launch.xml

```
<launch>
  <param name="moveit_controller_manager"
  value="moveit_fake_controller_manager/MoveItFakeControllerManager"/>
  <rosparam file="$(find
  ␣␣myworkcell_moveit_config)/config/fake_controllers.yaml"/>
</launch>
```

E.1.4 joystick_control.launch

```
<launch>
  <arg name="dev" default="/dev/input/js0" />

  <node pkg="joy" type="joy_node" name="joy">
    <param name="dev" value="$(arg␣dev)" />
    <param name="deadzone" value="0.2" />
    <param name="autorepeat_rate" value="40" />
    <param name="coalesce_interval" value="0.025" />
  </node>

  <node pkg="moveit_ros_visualization" type="moveit_joy.py"
  output="screen" name="moveit_joy"/>
</launch>
```

E.1.5 move_group.launch

```
<launch>
  <include file="$(find␣myworkcell_moveit_config)
```

```
␣␣/launch/planning_context.launch" />

  <arg name="debug" default="false" />
  <arg unless="$(arg␣debug)" name="launch_prefix" value="" />
  <arg      if="$(arg␣debug)" name="launch_prefix"
  value="gdb␣-x␣$(find␣myworkcell_moveit_config)
␣␣/launch/gdb_settings.gdb␣--ex␣run␣--args" />
  <arg name="info" default="$(arg␣debug)" />
  <arg unless="$(arg␣info)" name="command_args" value="" />
  <arg      if="$(arg␣info)" name="command_args" value="--debug" />
  <arg name="allow_trajectory_execution" default="true"/>
  <arg name="fake_execution" default="false"/>
  <arg name="max_safe_path_cost" default="1"/>
  <arg name="jiggle_fraction" default="0.05" />
  <arg name="publish_monitored_planning_scene" default="true"/>

  <include ns="move_group" file="$(find␣myworkcell_moveit_config)
␣␣/launch/planning_pipeline.launch.xml">
    <arg name="pipeline" value="ompl" />
  </include>

  <include ns="move_group" file="$(find␣myworkcell_moveit_config)
␣␣/launch/trajectory_execution.launch.xml"
   if="$(arg␣allow_trajectory_execution)">
    <arg name="moveit_manage_controllers" value="true" />
    <arg name="moveit_controller_manager" value="myworkcell"
     unless="$(arg␣fake_execution)"/>
    <arg name="moveit_controller_manager" value="fake"
     if="$(arg␣fake_execution)"/>
```

```
  </include>

  <include ns="move_group" file="$(find␣myworkcell_moveit_config)
␣␣/launch/sensor_manager.launch.xml" if="$(arg␣allow_trajectory_execution)">
    <arg name="moveit_sensor_manager" value="myworkcell" />
  </include>

  <node name="move_group" launch-prefix="$(arg␣launch_prefix)"
  pkg="moveit_ros_move_group"  type="move_group" respawn="false"
  output="screen" args="$(arg␣command_args)">
    <env name="DISPLAY" value="$(optenv␣DISPLAY␣:0)" />
    <param name="allow_trajectory_execution"
    value="$(arg␣allow_trajectory_execution)"/>
    <param name="max_safe_path_cost" value="$(arg␣max_safe_path_cost)"/>
    <param name="jiggle_fraction" value="$(arg␣jiggle_fraction)" />
        <param name="capabilities"
        value="move_group/MoveGroupCartesianPathService
␣␣␣␣␣␣␣␣move_group/MoveGroupExecuteTrajectoryAction
␣␣␣␣␣␣␣␣move_group/MoveGroupKinematicsService
␣␣␣␣␣␣␣␣move_group/MoveGroupMoveAction
␣␣␣␣␣␣␣␣move_group/MoveGroupPickPlaceAction
␣␣␣␣␣␣␣␣move_group/MoveGroupPlanService
␣␣␣␣␣␣␣␣move_group/MoveGroupQueryPlannersService
␣␣␣␣␣␣␣␣move_group/MoveGroupStateValidationService" />
    <param name="planning_scene_monitor/publish_planning_scene"
    value="$(arg␣publish_monitored_planning_scene)" />
    <param name="planning_scene_monitor/publish_geometry_updates"
    value="$(arg␣publish_monitored_planning_scene)" />
    <param name="planning_scene_monitor/publish_state_updates"
```

```
    value="$(arg publish_monitored_planning_scene)" />
    <param name="planning_scene_monitor/publish_transforms_updates"
    value="$(arg publish_monitored_planning_scene)" />
  </node>
</launch>
```

E.1.6 myworkcell_moveit_controller_manager.launch.xml

```
<launch>
  <arg name="moveit_controller_manager"
  default="moveit_simple_controller_manager/MoveItSimpleControllerManager"/>
  <param name="moveit_controller_manager"
  value="$(arg moveit_controller_manager)"/>
  <rosparam file="$(find myworkcell_moveit_config)/config/controllers.yaml"/>
</launch>
```

E.1.7 myworkcell_planning_execution.launch

```
<launch>
  <rosparam command="load" file="$(find myworkcell_moveit_config)
  /config/joint_names.yaml"/>

  <arg name="sim" default="true" />
  <arg name="robot_ip" unless="$(arg sim)" />

  <include file="$(find myworkcell_moveit_config)
  /launch/planning_context.launch" >
    <arg name="load_robot_description" value="true" />
  </include>

  <group if="$(arg sim)">
```

```
    <include file="$(find␣industrial_robot_simulator)
␣␣␣␣/launch/robot_interface_simulator.launch" />
  </group>

  <group unless="$(arg␣sim)">
    <include file="$(find␣ur_bringup)/launch/ur5_bringup.launch" />
  </group>

  <!--include file="$(find␣openni_launch)/launch/openni.launch" >
    <arg name="depth_registration" value="true" />
  </include>
  <node pkg="tf" type="static_transform_publisher"
  name="link1_broadcaster" args="2.5␣0.0␣-0.8␣3.14␣1.3␣0.0
␣␣/camera_link␣/base_link␣10" /-->

  <include file="$(find␣freenect_launch)/launch/freenect.launch" >
    <arg name="depth_registration" value="true" />
  </include>
  <node pkg="tf" type="static_transform_publisher"
  name="link1_broadcaster" args="2.5␣0.0␣-1.0␣3.14␣1.53␣0.0
␣␣camera_link␣kinect␣10" />

  <!-- publish the robot state (tf transforms) -->
  <node name="robot_state_publisher" pkg="robot_state_publisher"
  type="robot_state_publisher" />
  <!--arg name="gui" default="true"/>
  <node name="joint_state_publisher" pkg="joint_state_publisher"
  type="joint_state_publisher" >
   <param name="use_gui" value="$(arg␣gui)"/>
```

```
  </node-->

  <include file="$(find␣myworkcell_moveit_config)
␣␣/launch/move_group.launch">
    <arg name="publish_monitored_planning_scene" value="true" />
  </include>

  <include file="$(find␣myworkcell_moveit_config)
␣␣/launch/moveit_rviz.launch">
    <arg name="config" value="true"/>
  </include>
</launch>
```

E.1.8 octomap.launch

```
<launch>
  <node pkg="octomap_server" type="octomap_server_node"
  name="octomap_server">

    <!-- resolution in meters per pixel -->
    <param name="resolution" value="0.025" />

    <!-- name of the fixed frame, needs to be "/map" for SLAM -->
    <param name="frame_id" type="string" value="kinect" />

    <!-- max range / depth resolution of the kinect in meter -->
    <param name="sensor_model/max_range" value="5.0" />
    <param name="latch" value="true" />

    <!-- max/min height for occupancy map, should be in meters -->
```

```
    <param name="pointcloud_max_z" value="10" />
    <param name="pointcloud_min_z" value="0.5" />

    <!-- topic from where pointcloud2 messages are subscribed -->
    <remap from="cloud_in" to="/output" />

  </node>
</launch>
```

E.1.9 ompl_planning_pipeline.launch.xml

```
<launch>
  <arg name="planning_plugin" value="ompl_interface/OMPLPlanner" />

  <arg name="planning_adapters"
  value="default_planner_request_adapters/AddTimeParameterization
  default_planner_request_adapters/FixWorkspaceBounds
  default_planner_request_adapters/FixStartStateBounds
  default_planner_request_adapters/FixStartStateCollision
  default_planner_request_adapters/FixStartStatePathConstraints" />

  <arg name="start_state_max_bounds_error" value="0.1" />

  <param name="planning_plugin" value="$(arg planning_plugin)" />
  <param name="request_adapters" value="$(arg planning_adapters)" />
  <param name="start_state_max_bounds_error"
  value="$(arg start_state_max_bounds_error)" />

  <rosparam command="load" file="$(find myworkcell_moveit_config)
  /config/ompl_planning.yaml"/>
```

```
</launch>
```

E.1.10 planning_context.launch

```
<launch>
  <arg name="load_robot_description" default="false"/>
  <arg name="robot_description" default="robot_description"/>

  <param if="$(arg load_robot_description)"
  name="$(arg robot_description)" command="$(find xacro)
  /xacro --inorder '$(find myworkcell_support)
  /urdf/workcell.xacro'"/>
  <param name="$(arg robot_description)_semantic"
   textfile="$(find myworkcell_moveit_config)/config/myworkcell.srdf" />

  <group ns="$(arg robot_description)_planning">
    <rosparam command="load" file="$(find myworkcell_moveit_config)
    /config/joint_limits.yaml"/>
  </group>

  <group ns="$(arg robot_description)_kinematics">
    <rosparam command="load" file="$(find myworkcell_moveit_config)
    /config/kinematics.yaml"/>
  </group>
</launch>
```

E.1.11 planning_pipeline.launch.xml

```
<launch>
  <arg name="pipeline" default="ompl" />
```

```
  <include file="$(find␣myworkcell_moveit_config)
␣␣/launch/$(arg␣pipeline)_planning_pipeline.launch.xml" />
</launch>
```

E.1.12 run_benchmark_ompl.launch

```
<launch>
  <arg name="cfg" />

  <include file="$(find␣myworkcell_moveit_config)
␣␣/launch/planning_context.launch">
    <arg name="load_robot_description" value="true"/>
  </include>

  <include file="$(find␣myworkcell_moveit_config)/launch/warehouse.launch">
    <arg name="moveit_warehouse_database_path"
    value="moveit_ompl_benchmark_warehouse"/>
  </include>

  <node name="$(anon␣moveit_benchmark)" pkg="moveit_ros_benchmarks"
  type="moveit_run_benchmark" args="$(arg␣cfg)␣--benchmark-planners"
  respawn="false" output="screen">
    <rosparam command="load" file="$(find␣myworkcell_moveit_config)
␣␣␣␣/config/kinematics.yaml"/>
    <rosparam command="load" file="$(find␣myworkcell_moveit_config)
␣␣␣␣/config/ompl_planning.yaml"/>
  </node>
</launch>
```

E.1.13 sensor_manager.launch.xml

```
<launch>
  <arg name="moveit_sensor_manager" default="myworkcell" />
  <include file="$(find␣myworkcell_moveit_config)
␣␣/launch/$(arg␣moveit_sensor_manager)_moveit_sensor_manager.launch.xml" />
  <param name="octomap_frame" type="string" value="world_frame" />
  <param name="octomap_resolution" type="double" value="0.5" />
  <param name="max_range" type="double" value="5.0" />
  <rosparam command="load" file="$(find␣myworkcell_moveit_config)
␣␣/config/sensors_kinect.yaml" />
</launch>
```

E.1.14 setup_assistant.launch

```
<launch>
  <arg name="debug" default="false" />
  <arg unless="$(arg␣debug)" name="launch_prefix" value="" />
  <arg     if="$(arg␣debug)" name="launch_prefix"
  value="gdb␣--ex␣run␣--args" />

  <node pkg="moveit_setup_assistant" type="moveit_setup_assistant" n
  ame="moveit_setup_assistant"
        args="--config_pkg=myworkcell_moveit_config"
        launch-prefix="$(arg␣launch_prefix)"
        output="screen" />
</launch>
```

E.1.15 state.launch

```
<launch>
  <arg name="gui" default="true"/>
  <param name="robot_description" command="$(find␣xacro)/xacro
```

```
␣␣--inorder␣'$(find␣myworkcell_support)/urdf/workcell.xacro'" />
  <node name="robot_state_publisher" pkg="robot_state_publisher"
  type="robot_state_publisher"/>
  <node name="joint_state_publisher" pkg="joint_state_publisher"
  type="joint_state_publisher">
    <param name="use_gui" value="$(arg␣gui)"/>
  </node>
</launch>
```

E.1.16 trajectory_execution.launch.xml

```
<launch>
  <arg name="moveit_manage_controllers" default="true"/>
  <param name="moveit_manage_controllers"
  value="$(arg␣moveit_manage_controllers)"/>

  <param name="trajectory_execution/allowed_execution_duration_scaling"
  value="1.2"/> <!-- default 1.2 -->
  <param name="trajectory_execution/allowed_goal_duration_margin"
  value="0.5"/> <!-- default 0.5 -->
  <param name="trajectory_execution/allowed_start_tolerance" value="0.01"/>
  <!-- default 0.01 -->

  <arg name="moveit_controller_manager" default="myworkcell" />
  <include file="$(find␣myworkcell_moveit_config)/launch/$
␣␣(arg␣moveit_controller_manager)_moveit_controller_manager.launch.xml" />
</launch>
```

E.1.17 warehouse.launch

```
<launch>
```

```
  <arg name="moveit_warehouse_database_path" />

  <include file="$(find␣myworkcell_moveit_config)
␣␣/launch/warehouse_settings.launch.xml" />

  <node name="$(anon␣mongo_wrapper_ros)" cwd="ROS_HOME"
  type="mongo_wrapper_ros.py" pkg="warehouse_ros_mongo">
    <param name="overwrite" value="false"/>
    <param name="database_path" value="$(arg␣moveit_warehouse_database_path)"
  </node>
</launch>
```

E.1.18 warehouse_settings.launch.xml

```
<launch>
  <arg name="moveit_warehouse_port" default="33829" />
  <arg name="moveit_warehouse_host" default="localhost" />

  <param name="warehouse_port" value="$(arg␣moveit_warehouse_port)"/>
  <param name="warehouse_host" value="$(arg␣moveit_warehouse_host)"/>
  <param name="warehouse_exec" value="mongod" />
  <param name="warehouse_plugin"
  value="warehouse_ros_mongo::MongoDatabaseConnection" />
</launch>
```

E.2 Code to set the Kinect

```
sensors:
 - sensor_plugin: occupancy_map_monitor/PointCloudOctomapUpdater
   point_cloud_topic: /kinect/depth_registered/points
```

```
    max_range: 5.0
    frame_subsample: 1
    point_subsample: 1
    shape_offset: 0
    shape_scale: 1.0
    shape_padding: 0.05
```

E.3 Code for the Point Cloud

```
#include <ros/ros.h>
// PCL specific includes
#include <sensor_msgs/PointCloud2.h>
#include <pcl_conversions/pcl_conversions.h>
#include <pcl/point_cloud.h>
#include <pcl/point_types.h>

ros::Publisher pub;

void cloud_cb (const sensor_msgs::PointCloud2ConstPtr& input)
{
  // Create a container for the data.
  sensor_msgs::PointCloud2 output;

  // Do data processing here...
  output = *input;

  // Publish the data.
  pub.publish (output);
}
```

```
int main (int argc, char** argv)
{
  // Initialize ROS
  ros::init (argc, argv, "my_pcl_tutorial");
  ros::NodeHandle nh;

  // Create a ROS subscriber for the input point cloud
  ros::Subscriber sub = nh.subscribe ("input", 1, cloud_cb);

  // Create a ROS publisher for the output point cloud
  pub = nh.advertise<sensor_msgs::PointCloud2> ("output", 1);

  // Spin
  ros::spin ();
}
```