Hochschule Aachen

Development of a URDF file for simulation and programming of a delta robot using ROS

Thesis

TO ACHIEVE THE ACADEMIC DEGREE OF

Master in Mechatronics

BY

Rodrigo Torres Arrazate

Santiago de Querétaro, Qro., México, February 2017

Declaration of Authorship

I hereby declare that I have written the present work independently and that the sources used in this work are acknowledged as references.

The drawings or illustrations presented in this work are created by myself and if it is not the case, the images have a corresponding source proof.

This work was not previously presented to another examination board and has not been published.

Santiago de Querétaro, Qro., México, February 2017

Acknowledgements

First of all, I want to thank Prof. Dr. Ing. Jörg Wollert for the opportunity of working with this challenging project and his support during it. Also, I would like to thank M.Sc. Jesús Arturo Velarde Sánchez for his help and advices during the development of the project and the writing process of the thesis. Finally, thanks to the team members of the laboratory at the university in Aachen, to Victor Saldivar, Israel Alvarez, Alfonso Castro, Hector Zamora, Salvador Jimenez and B.Sc. Sebastian Rau that always helped me when I needed it.

Also, I want to express my gratitude to CIDESI and CONACYT for giving me the experience of studying and living abroad, without both institutions, this dream would never have come true.

To my parents, Luz del Rocío and Hugo, my sister, Rocío and my girlfriend, Lupita, that always encouraged me to keep going. Thanks for the advices, your help and for being with me during this process, this work came true thanks to all of you.

Abstract

This work presents the development of a 3D model of an ABB IRB340 delta robot using ROS. The motion of the 3D model is based on inverse kinematics, for this, it is necessary to provide the desired coordinates of the end effector. To accomplish this task, it is detailed the derivation of the inverse kinematics equations to obtain the robot's actuators state given the desired coordinates. Also, the results of the inverse kinematics computation are validated by means of forward kinematics equations. Once the inverse kinematics algorithm is validated, it is employed to send the control variables to the URDF model in ROS. In addition, it was created a robot motion routine to demonstrate the active operation of the 3D model in a graphical environment. It is worthwhile to mention that due to limitations on URDF joint properties the displacement of the robot's model is limited along one axis.

Contents

List of figures

List of tables

Chapter 1

Introduction

Robotics is an interdisciplinary field that can be define as the science and technology of robots, their manufacture, design and applications. Due to its nature, this field combines electrical and mechanical components as well as computer science. Robots are systems that contain control systems, sensors and manipulators, used to achieve tasks that must be performed repetitively whenever it is needed. Generally, robots are used in different industrial sectors such as automotive, pharmaceutical, food, packaging, etc.

The use of robots in industry is increasing constantly and rapidly due to the facility to work in different applications, such as, welding, handling operations, assembly, pick and place, packaging, drilling painting, chemical handling, among others. There are different reasons for using robots in industry, for instance, they can operate continuously without stopping the process, reduce the production time, provide high quality products, etc.

Nowadays, in industry there are many different types of robots like mobile robots, robotic arms, delta robots, among others. This work is focused on delta robots which have become a popular robot in industry. The major advantages of using delta robots are, high accuracy, faster response, high stiffness, high ratios of rigidity to weight, reliability and versatility, it works inside a smaller workspace, etc.

The spread use of robots has open the necessity of more capable robot applications. For this reason, different computational tools have been developed to ease the creation of such applications. One of these tools is Robot Operating System (ROS), which, provides a wide set of tools, libraries and conventions to simplify the creation of complex and robust algorithms for robots.

1.1 Justification

The use of delta robots optimize every task that must be done, due to its higher accuracy, faster response, greater reliability and versatility, being these key points to improve automation systems in various sectors of industry.

This project seeks to create a URDF file of a delta robot in order to use it to simulate the robot inside a graphical environment, in this way, using simulation and visualization tools, it will be possible to observe the performance of the robot when it is moving.

There are delta robots within the FH Aachen university facility that can be used for didactic purposes. Then, the students will be able of making tests using simulation and visualization tools before getting contact with the physical machinery. Once the students have sufficient knowledge about the behaviour of these robots, they will be able to get in contact with the real robots in order to work with them and carry out practical trainings.

Furthermore, the project pursues the use of ROS-I, which is a tool that leads to a versatile programming. This allows the use of diverse robots from different manufacturers and simplifies the development of complex systems. In such a way, the programming of an automation system can be implemented in another system with similar characteristics without the necessity of developing the programming from scratch.

1.2 Aim and objectives

1.2.1 Aim

This project aims to generate a URDF file for the simulation of a delta robot and develop a versatile programming routine for moving the robot observing its performance within a graphical environment.

1.2.2 Objectives

- Conduct literature review of concepts of Robot Operating System (ROS).
- Make practical examples of ROS to get familiar with the environment.
- Conduct literature review of Robot Operating System Industrial (ROS-I).
- Carry out practical examples of ROS-I for getting familiar with the environment.
- Conduct literature review of concepts of Unified Robot Description Format (URDF) files to get familiar with this files.
- Make practical examples of URDF files to get familiar with the concepts of this files.
- Conduct literature review of concepts of URDF files for industrial robots to get familiar with the concepts and standards used for its developing.
- Make practical examples of URDF files of industrial robots for getting familiar with the concepts and standards of this files.
- Investigate and select the kind of kinematic the delta robot will use in order to perform its movements.
- Design the URDF file of the delta robot.
- Make firsts tests for observing the delta robot inside the graphical environment.
- Create a routine for moving the delta robot and observe its performance inside a graphical environment.

Chapter 2

Background

2.1 Robotics

Over the time, humans have look for substitutes that would be able of imitate their behaviour in the interaction with the environment. The word robot was introduced in 1920 by Karel Capek, a Czech playwright, in his play *“Rossum’s Universal Robots”.* The term robot is derived from the Slav word *robota* that means executive labour [1].

The structure of a robot consists of rigid bodies (links) interconnected by means of articulations (joints) that allows the mobility of the robot. The mechanical structure of the robot or kinematic chain, can be represented by a wrist that gives dexterity, an arm that guarantee mobility and an end effector to perform the required task [1].

The robots used in industrial applications can be serial or open chain, and closed kinematic chain manipulators. A kinematic chain is open when just one sequence of links are connecting the two ends of the chain, whereas a kinematic chain is closed when a sequence of links forms a loop.

The serial manipulators cover a large workspace when it is fully extended, and their joints can be controlled independently. But, its big disadvantage is the low payload capacity comparing to the large structures and motors of each link, causing slow acceleration and deceleration.

A delta robot is considered a closed kinematic chain manipulator and it possess a great number of advantages when comparing it to serial manipulators. For instance, higher rigidity and small mobile mass, that provides faster acceleration without losing accuracy. Also, delta robots can move heavier payload relative to its body mass. The main drawback of delta robots is their limited range of motion comparing it to a serial manipulator.

2.2 Delta robots

In 1988, Professor Reymond Clavel, at the Swiss Federal Institute of Technology in Lausanne, developed an idea of using parallel robots with three translational and one rotational degree of freedom, creating the delta robots [2].

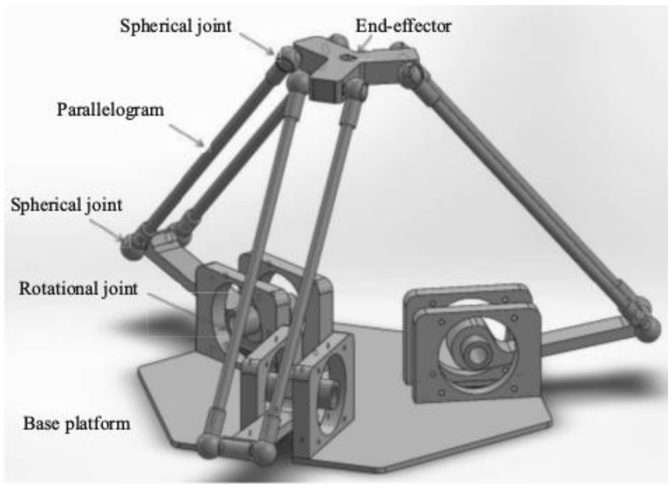


Figure 1 *3D Schematic of a delta robot made in SolidWorks. Image from [2].*

The idea of using delta robots is the use of parallelograms. The parallelograms allow an output link to remain at a fixed position with respect to the input link. The function of the three parallelograms is to restrain the end effector, remaining only with three translational degrees of freedom.

Forming a closed loop structure, each one of the arms are connected to the base platform via revolute joints [3], this joints permit only rotation of one of the bodies relative to another, thus, this joint has just one degree of freedom [4]. The position of the end effector is changing accordingly to the angular position of the motors on the base platform [5]. Finally, a fourth leg is used to transmit rotatory motion (Figure 2) from the base to the end effector mounted on the mobile platform [6].

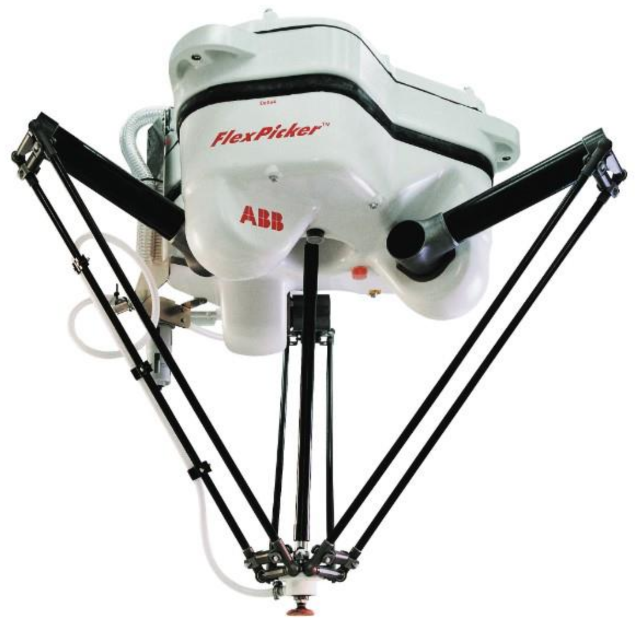


***Figure 2** Example of an ABB IRB 340 delta robot. Image from [7].*

The core advantage of the delta robot is its speed, the difference between the typical robot arms and delta robots is that a typical robot arm has to move the payload and also all the servos in each joint and a delta robot only moves its frame, which usually is made of a lightweight material. Due to this advantage, delta robots achieve accelerations of up to 12g in industrial applications [2], making it a perfect candidate for pick and place operations of light objects.

2.3 Kinematics

According to Craig in [8], "Kinematics is the science of motion which treats motion without regard to the forces that cause it. Within the science of kinematics one studies the position, velocity, acceleration, and all higher order derivatives of the position variables, with respect to time or any other variable or variables".

A manipulator can be represented as a kinematic chain of links connected using joints [1]. One end of the chain is attached to the base, while the other end of the chain of links is bounded to the end effector. The motion of the structure is obtained by composition of the elementary motions of each link. Therefore, in order to manipulate an object in space, it is necessary to describe the end effector position and orientation.

2.3.1 Modelling of delta robot

As mention in section 2.2, the position of the end effector changes accordingly to the angular position of the motors, so, by modifying these positions it is possible to control the location of the end effector. The effector is connected to the upper arms by means of universal joints [9].

Kinematics is used to describe the motion of a delta robot through mathematical equations, this with the intention to know how the end effector is led up to a desired position, leading to two possibilities, which are: forward or direct kinematics, and inverse kinematics.

2.3.2 Direct kinematics

Direct kinematics allows the end effector position and orientation to be expressed as a function of the joint variables of the mechanical structure with respect to a reference frame[1].

The position of the end effector can be calculated by the angular position of the motors [9], this can be done using motor encoders. When having the specific values of the encoders it is necessary to determine the position of the end effector. Another way to find the position of the end effector in a delta robot is by finding the intersection of three sphere equations, where the origin point of the spheres is determined by the dimensions of the upper arms and the angular position of motors [5].

The direct kinematics can provide the travelling plate centre for given three angles. The centre of the plate is formulated in a set of three coupled quadratic equations that must be solved simultaneously [3].

2.3.3 Inverse kinematics

Given a desired location of the end effector (points in the Cartesian coordinates $(X,\ Y, Z)$), it is necessary to determine the angular or translational positions of the robot's links to set them in the correct place to locate the end effector. This process is known as inverse kinematics. The solution to the inverse kinematics problem of a delta robot consists on the transformation of the end effector position into the angular positions of the three arms.

The inverse kinematics problem is complicated because the equations to solve are generally nonlinear, and thus multiple solutions may exist, it is not always possible to find a solution or there are no allowed solutions in view of the manipulator structure.

2.4 Robot Operating System (ROS)

ROS was developed by the Stanford Artificial Intelligence Laboratory in 2007. The development continues at Willow Garage which provided resources to extend ROS concepts and create tested implementations.

ROS is a framework for writing software for robots, designed for groups that collaborate and improve each other's work [10]. It provides a great variety of tools and libraries that simplify tasks for creating complex and robust robots. The libraries and tools that ROS contains can be used for writing, building and running code across multiple computers, etc. [11] [12]. This characteristics are helpful because it is possible to manipulate a single robot from different computers or even modify and improve the code running in the ROS system. ROS also provides standard operating system services such as hardware abstraction, device drivers, visualizers, package management, message passing between processes, and package management.

An advantageous point is that ROS was designed to be as modular as possible [13], this option is useful for the user or programmer, because it makes possible to implement a routine as and when it is desired. Another advantage of ROS is its license, called BSD license. This license is very permissive, people are free to take it and use it in proprietary, commercial and closed source products [14]. ROS is widely utilized by the research community for robotic applications because of its robustness and versatility. It can be applied to other areas, including industrial robotics (section 2.9). Finally, ROS is based on Linux system, so, all the characteristics and tools work correctly under Linux. It works as experimental on Mac OS and it has partial functionality under Windows [12].

Figure 3 *ROS framework. Image from [10].*

2.5 ROS concepts

ROS has three levels of concepts [15] [16], which are:

- File system level.
- Computation Graph level.
- Community level.

2.5.1 ROS file system level

ROS files are organized on the hard disk in a particular way. In this level it is possible to see how the files are organized on the disk. Below are shown the definition of the blocks in the file system:

Meta packages, it references one or more related packages which are grouped together.
Packages, are the most basic unit of the ROS software. It contains ROS nodes, configuration files, libraries, etc.

Package manifest, it contains information about the package, such as author name, license, dependencies and so on.

Messages, are the information sent from one ROS process to another one.

Services, it enables a request-reply interaction between processes or nodes.

Repositories, are collection of packages that share a common version control system (VCS) Some examples of repositories are: Git, mercurial, subversion, etc.

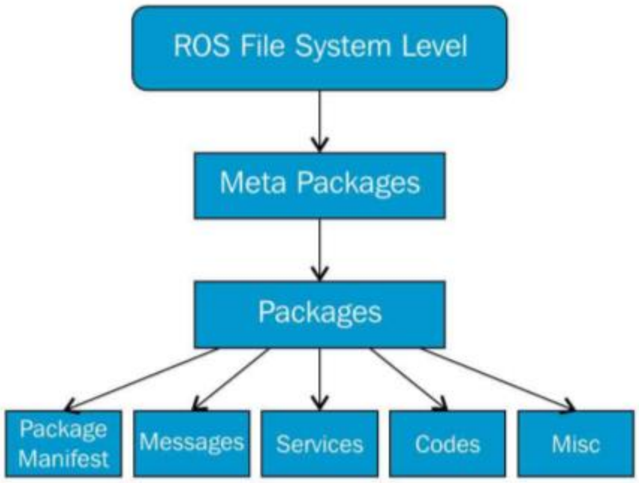
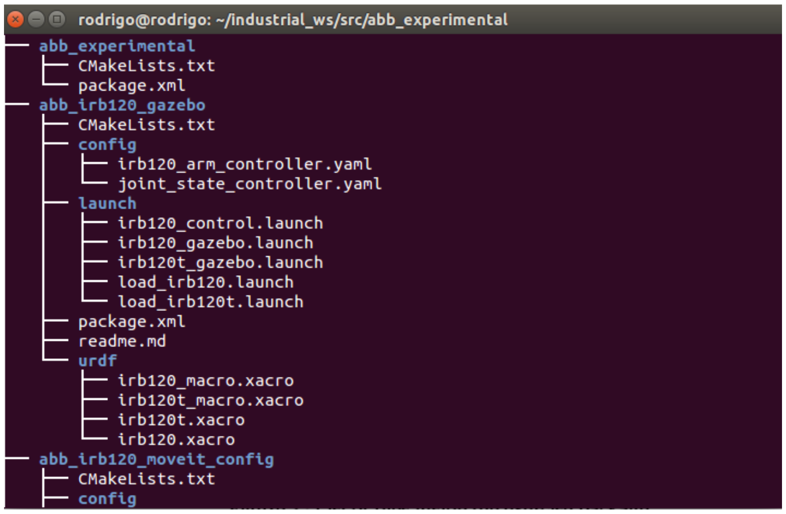


Figure 4 *ROS file system level. Image from [16].*



```
rodrigo@rodrigo: ~/industrial_ws/src/abb_experimental
── abb_experimental
   ├── CMakeLists.txt
   └── package.xml
── abb_irb120_gazebo
   ├── CMakeLists.txt
   ├── config
   │   ├── irb120_arm_controller.yaml
   │   └── joint_state_controller.yaml
   ├── launch
   │   ├── irb120_control.launch
   │   ├── irb120_gazebo.launch
   │   ├── irb120t_gazebo.launch
   │   ├── load_irb120.launch
   │   └── load_irb120t.launch
   ├── package.xml
   ├── readme.md
   └── urdf
       ├── irb120_macro.xacro
       ├── irb120t_macro.xacro
       ├── irb120t.xacro
       └── irb120.xacro
── abb_irb120_moveit_config
   ├── CMakeLists.txt
   ├── config
```

Figure 5 *List of files inside a package. This figure gives an idea of the basic files and folders contained inside a specific package.*

2.5.2 ROS computation graph level

The computation in ROS is done by using nodes, they are one of the basic concepts of ROS computation graph along with master, parameter server, messages, services, and bags. Each one of those provide data to the graph [15][16].

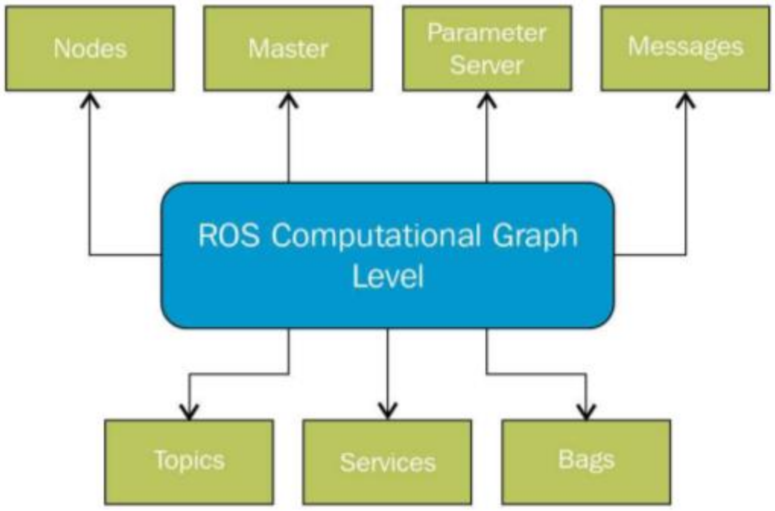


Figure 6 *ROS computational graph level. Image from [16]*

Nodes, are processes that perform computation and they are written with the use of ROS clients, such as roscpp or rospy.

ROS Master, provides name registration and lookup to the rest of the computation graph. If the master is not running, nodes would not be able to communicate or exchange messages.

Parameter Server, permits data to be stored in a central location and all nodes are able to access to it and modify it.

Messages, nodes communicate with each other by passing messages.

Topics, due to the publisher and subscriber communication, a node sends out a message by publishing it to a given topic, which is the name used to identify the content of the message.

Services, are defined by a pair of message structures: one for the request and one for the reply.

Bags, are a format for saving and playing back ROS message data.

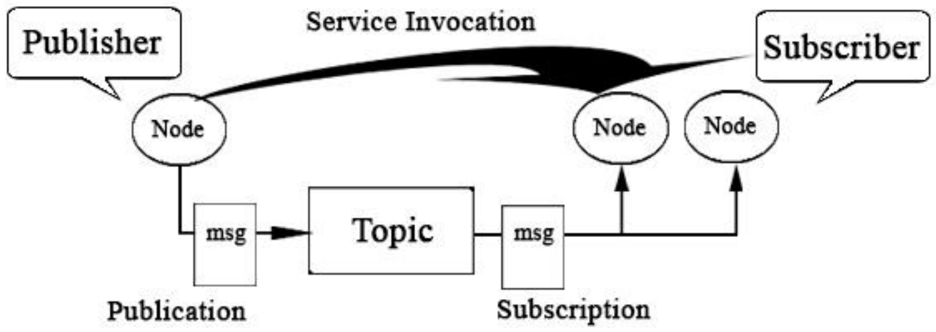


***Figure 7** Typical ROS communication. The figure shows the relationship between, nodes, publisher, subscriber and topic.*

2.5.3 ROS Community Level

In this level resources are used to exchange software and knowledge. These resources include: distributions, repositories, ROS wiki, ROS answer, etc. [16]. The goals of the community level are to let developers work against a stable code. Institutions or developers can release their own robot software and people can contribute by providing corrections or writing tutorials, and ask and answer questions related to ROS itself or a specific ROS software.

2.6 ROS Tools

As it was already told, one of the strongest features of ROS is its development toolset. These tools allow to diagnose problems easily [17], debugging, plotting, visualizing the state of the system, etc. Due to the publisher – subscriber communication it is possible to keep on track the flowing data through the system. Some of the most useful ROS tools are rqt_graph and RViz.

2.6.1 rqt_graph

It provides visualization of a ROS system, showing nodes and the connections between them [18] and it allows to debug and understand the running system and how it is structured.

In Figure 8, the ovals represent nodes, and the arrows represent publisher-subscriber relationships. It is possible to observe from the graph that the node named /joint_state_publisher publishes messages on a topic called /joint_states and the node named /robot_state_publisher subscribes to those messages.

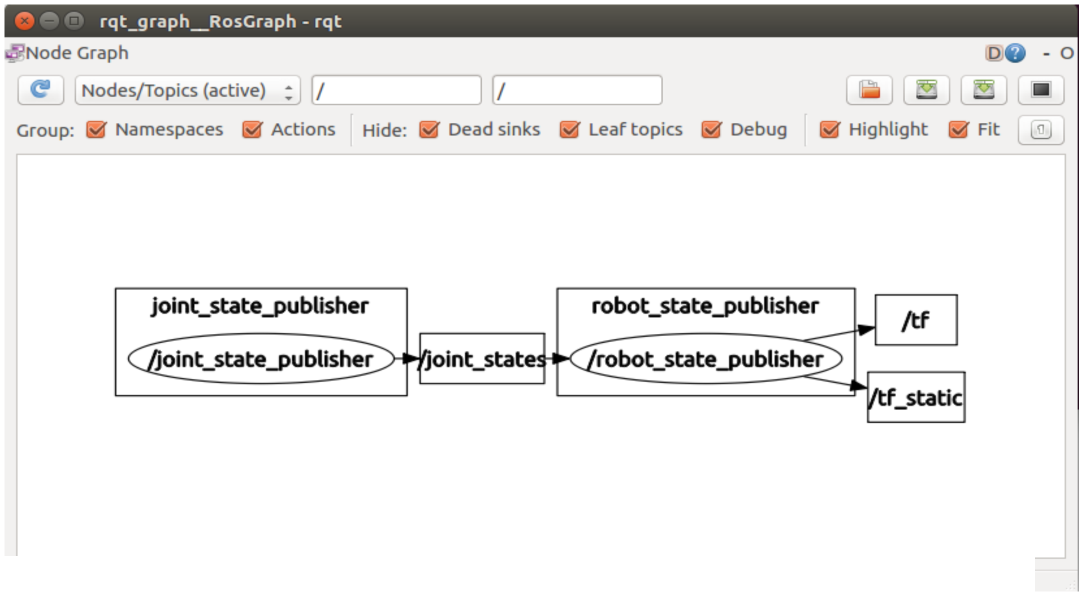
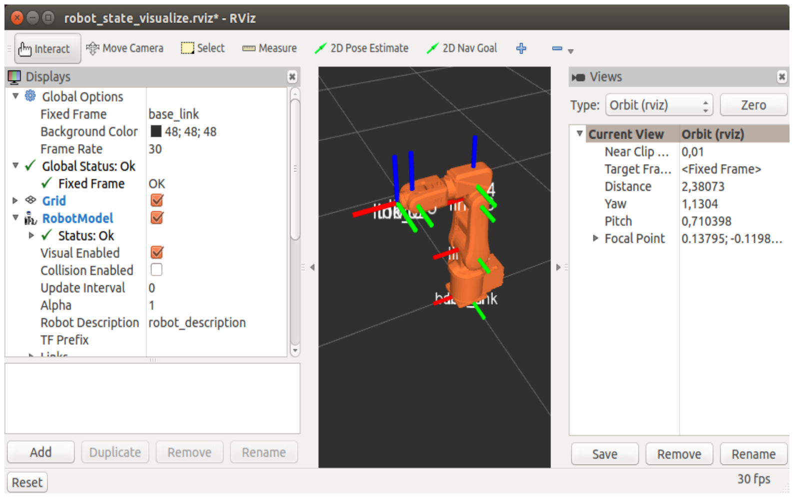


Figure 8 *rqt_graph interface showing the graph for a joint state publisher. The ovals represent the nodes, /joint_states, /tf and /tf_static are the topics.*

2.6.2 RViz

Some sensor devices like stereo cameras, 3D lasers, Kinect sensors, IMUs, etc. provide 3D data. ROS make available a graphical tool for displaying three dimensional data of different types of sensors. This tool is named RViz, which integrates an interface with a 3D world that stand for sensor data representation. To visualize correctly the sensor data type it is necessary to subscribe to an appropriate topic [17][18].

This tool also uses information from the tf library to show all of the sensor data in a common coordinate frame (section 2.7) along with a three dimensional model of the robot created via URDF files (section 2.8).



Global Options	
Fixed Frame	base_link
Background Color	■ 48; 48; 48
Frame Rate	30
✓ Global Status: Ok	
✓ Fixed Frame	OK
Grid	✓
RobotModel	✓
✓ Status: Ok	
Visual Enabled	✓
Collision Enabled	☐
Update Interval	0
Alpha	1
Robot Description	robot_description
TF Prefix	

Type: Orbit (rviz)

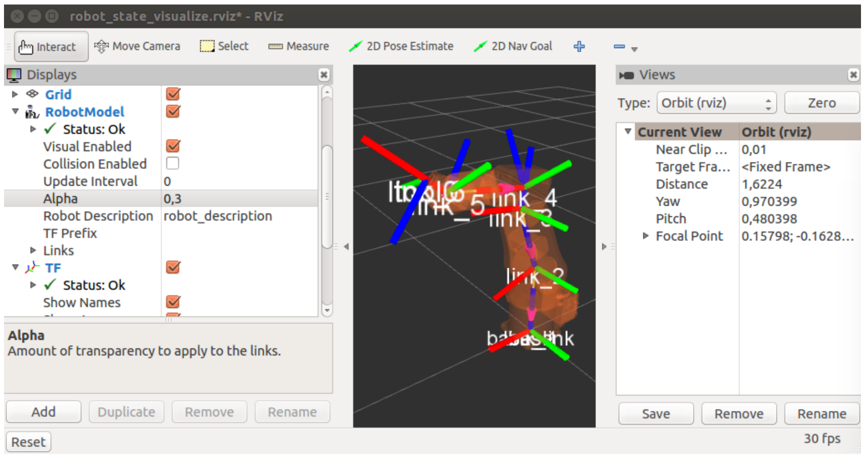
Current View	Orbit (rviz)
Near Clip ...	0,01
Target Fra...	<Fixed Frame>
Distance	2,38073
Yaw	1,1304
Pitch	0,710398
Focal Point	0.13795; -0.1198...

Figure 9 *RViz interface*

2.7 tf library

tf is a library designed to provide a standard way to keep track of the coordinate frames and transform data within the entire system. In this way, users know that the data is in the proper coordinate frame without requiring knowledge of all the coordinate frames in the system. tf library also maintains the relationship between coordinate frames in a tree structure and permits the user transform points, vectors, etc. between any two coordinate frames at any desired point in time. [19]

The tf library operates in a distributed system, the information corresponding to the coordinate frames of a robot is available to all ROS components on any computer in the system.



Grid	✓
RobotModel	✓
Status: Ok	
Visual Enabled	✓
Collision Enabled	☐
Update Interval	0
Alpha	0,3
Robot Description	robot_description
TF Prefix	
Links	
TF	✓
Status: Ok	
Show Names	✓

Alpha
Amount of transparency to apply to the links.

Type: Orbit (rviz)

Current View	Orbit (rviz)
Near Clip ...	0,01
Target Fra...	<Fixed Frame>
Distance	1,6224
Yaw	0,970399
Pitch	0,480398
Focal Point	0.15798; -0.1628...

30 fps

***Figure 10** Visualization of the tf frames in an IRB120 ABB robot. The RGB cylinders represent X, Y, and Z axes respectively.*

This library has two standard modules, broadcaster and listener. These modules are designed to integrate with the ROS ecosystem but are generally useful outside of ROS [20]. The broadcaster send updates periodically of the pose of coordinate frames to the rest of the system. The system is capable of having many broadcasters providing information about a different part of the robot [19][21]. The listener collects the received values and buffer all coordinate frames that are broadcasted in the system, and query for specific transforms between frames [21].

2.8 Unified Robot Description Format (URDF)

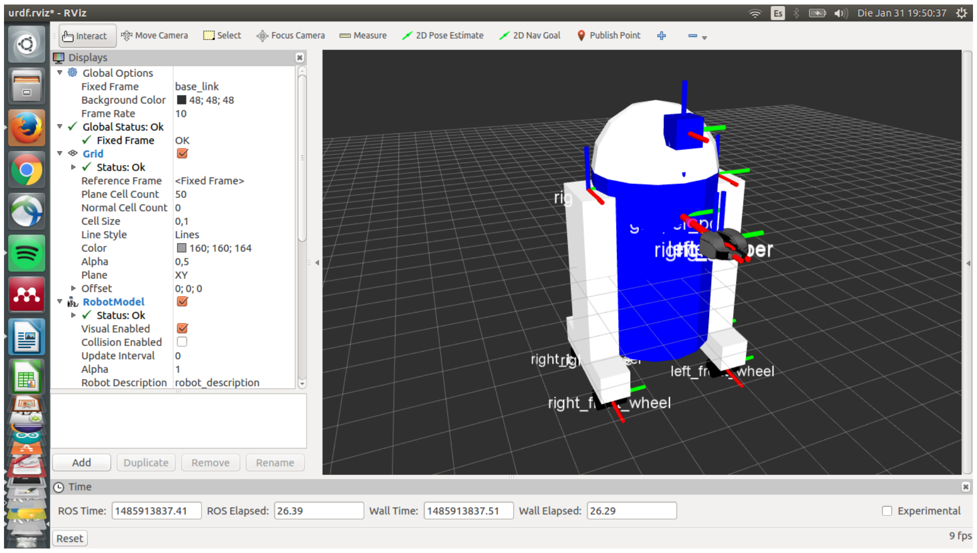
In ROS, it is possible to visualize a model of a robot by using Unified Robot Description Format files (URDF). The URDF files are based on XML, and it is designed for obtaining a 3D representation of a real robot [17]. Using this type of files it is possible to simulate diverse characteristics of robots such as, shape, colour, joints, etc.

The way of building and visualizing a robot model in URDF is writing and compiling the URDF file. While writing the file it is important to say that it should be divided into links and joints, and the connection relationship between parts is described by <parent> and <child> (Table 1).

Once the 3D representation of a robot is created by using a URDF file, it is possible to use such representation to simulate the motion of the robot. For this, the user needs to publish the robot conditions to tf, using a node (or nodes) to publish the transform information [22].

Command	Grammar
Name of the robot	robot name = “robot”
Define the part	link name = “base_link”/
Define the connection node	joint name = ”arm_1_joint” type = “revolute”
Define the connection relationship	parent link = “base_link”/ child link = “arm_1”/

Table 1 *Example of common commands used in URDF files.*



Global Options	
Fixed Frame	base_link
Background Color	■ 48; 48; 48
Frame Rate	10
✓ Global Status: Ok	
✓ Fixed Frame	OK
Grid	✓
✓ Status: Ok	
Reference Frame	<Fixed Frame>
Plane Cell Count	50
Normal Cell Count	0
Cell Size	0,1
Line Style	Lines
Color	■ 160; 160; 164
Alpha	0,5
Plane	XY
Offset	0; 0; 0
RobotModel	✓
✓ Status: Ok	
Visual Enabled	✓
Collision Enabled	☐
Update Interval	0
Alpha	1
Robot Description	robot_description

ROS Time: 1485913837.41 ROS Elapsed: 26.39 Wall Time: 1485913837.51 Wall Elapsed: 26.29 ☐ Experimental

Figure 11 *Visual model of a robot that look like R2D2. Done during URDF tutorials ROS wiki*

2.9 ROS Industrial (ROS-I)

Processes inside industry have been based on simple and repetitive tasks performed by robots. Nowadays, there has been an increasing demand for dynamic and intelligent robots able of realise more difficult and complex tasks. ROS-I offers high level tools and programs capable of solving this tasks.

This project began as a collaboration between Yaskawa Motoman Robotics, Southwest Research Institute, and Willow Garage in 2012 to support the use of ROS for industrial automation [23]. ROS-Industrial contains repositories, libraries and tools for industrial manipulators, grippers, sensors, etc. It is supported by an international consortium of industry and research members [24], having as its principal goal to extend the capabilities of ROS to industrial robots working in the production process, supporting the use of ROS for industrial automation and processes, enabling manufacturing robotic applications that were infeasible or cost prohibitive. [25]

Figure 12 *ROS Industrial Logo. Image from [26]*

The industry can benefit from the development tools that has been created around ROS. For instance, its framework provides tools for visualization and human machine interaction, which can ease the use of robotics systems. On the other hand, there are an increasing number of tools being improved by ROS-I, like tools for navigation, manipulation and visualization[25]. In this way, tools that were created and/or improved for its employment within industry sectors can also be used in ROS, improving existing tools.

In order to have a better overview of the advantages of using ROS-I the main goals and the benefits are mentioned below [16][25].

Main goals of ROS-I

- Combine strengths of ROS to the existing industrial technologies for exploring advanced capabilities of ROS in the manufacturing process.
- Provide an easy path to apply cutting edge research in industrial applications.
- Create a wide community supported by researchers and professionals in industrial robotics.
- Provide simple, easy to use APIs (Application program interface), which are a set of routines, protocols and tools for building software applications.

Benefits of ROS-I

- ROS-I is able to use all ROS features in industrial robots. For example it is possible to use different ROS tools such as RViz, Gazebo and rqt for visualization, simulation and debugging.
- ROS-I is an open source software that allows commercial use without restrictions.
- Provides simple, easy to use, well documented application programming interfaces.

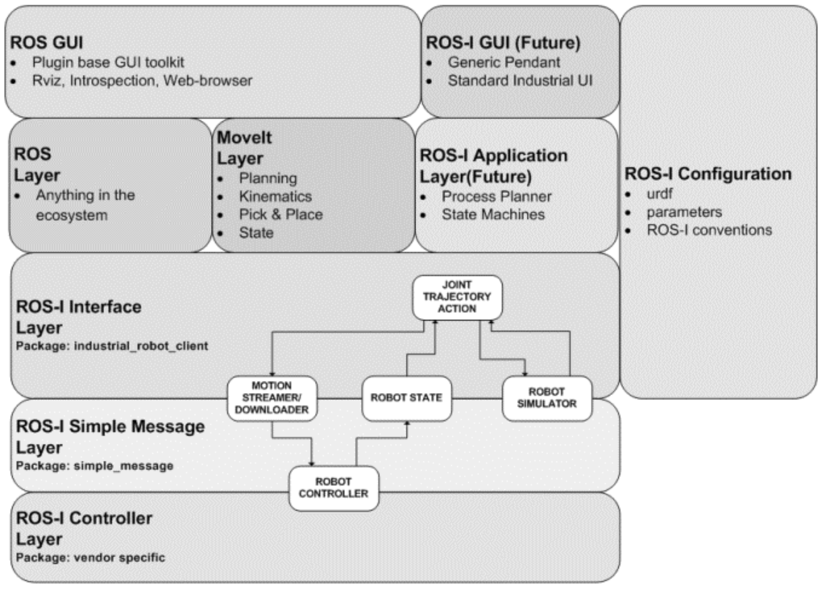


Figure 13 *ROS Industrial high level architecture. Image from [26]*

2.10 Unified Robot Description Format (URDF) for industrial robots

Create the URDF file for an ordinary robot and industrial robot are the same, both are XML files that describes a robot and its joints as well as the kinematics of the robot. But, for industrial robots certain standards should be followed during the creation of the URDF file[16].

- The URDF should be simple and readable.
- Develop a common design formula for all industrial robots by various vendors.
- The entire manipulator section is written as a macro using xacro.
- The base_link frame should be the first link and tool0 should be the end effector link.

Chapter 3

Inverse kinematics equations of a delta robot

This chapter presents a strategy based on inverse kinematics equations to transform a desired position of the end effector of a delta robot into joint parameters. The results obtained using inverse kinematics are validated by direct kinematics equations.

3.1 Problem statement

It is desired to express the motion of a delta robot given the location of the end effector. To fulfil this task, it is necessary to obtain the joint parameters that provide the desired position of the end effector. This is done by inverse kinematics, and in order to carry out such mathematical process it is essential to define the robot characteristics.

A delta robot is a closed kinematic chain manipulator. It consists in a fixed base and the end effector interconnected by three legs that form parallelograms. Therefore, the end effector is restricted to translational motion and its position is limited by the dimensions of the legs.

3.2 Methodology

3.2.1 Delta robot description

As shown in Figure 14, delta robots are composed of three identical legs in parallel between the fixed base and the end effector. Like the delta robot IRB340, the robot model in Figure 14 uses revolute joints for moving the upper arms. The control variables $\theta_i, i = 1, 2, 3$ are measured using the rule of the right hand and zero angle is defined when the actuated link is in the horizontal plane.

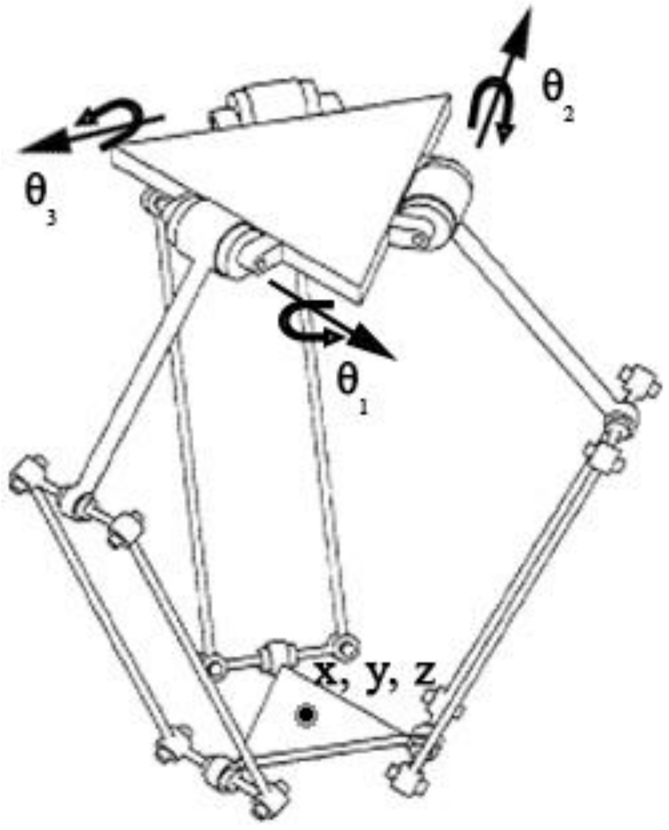


Figure 14 *Delta Robot Diagram.* θ_i $control\ variables$. *Image modified from [27]*

The delta robot is able to translate the end effector in the X, Y, Z coordinates. The legs are represented by the points: $B_i, i = 1, 2, 3$ the hips: $A_i, i = 1, 2, 3$, the knees; and $P_i, i = 1, 2, 3$, the ankles. The side length of the end effector is s_P, the length of the upper arms are defined as L while the length of the lower arms is l. For a better overview, these representations are displayed in Figure 15.

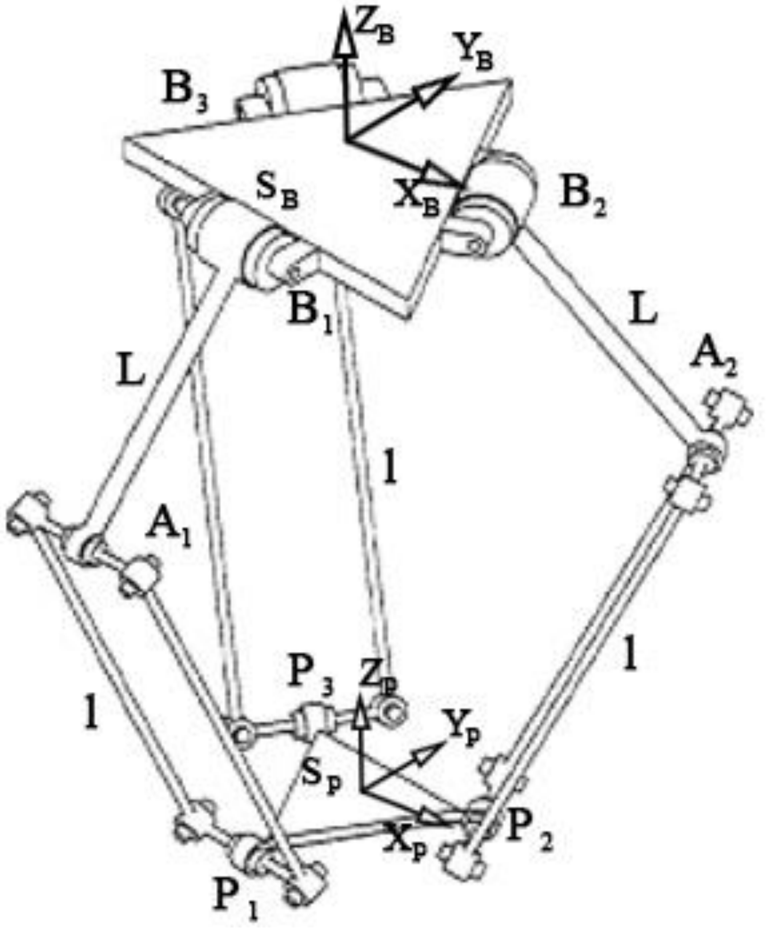


Figure 15 *Delta robot representations for kinematic analysis. B_i hips of the robot, A_i knees of the robot, P_i ankles of the robot, s_B side length of the fixed base, s_P side length of the end effector, L length of upper arms, l length of lower arms. Image modified from [27]*

From Figure 16 and 17, it can be observed that the reference frame of the fixed base is $\{B\}$, and $\{P\}$ is the reference frame of the end effector. Both origins are located at the centre of each triangle, therefore, the orientation of $\{P\}$ and $\{B\}$ are identical, leading to a constant rotation matrix $({}^B_P R = I_3)$. [28] The joint variables are $\Theta = \{\theta_1\ \theta_2\ \theta_3\}^T$, and the Cartesian coordinates are ${}^B P_P = \{x\ y\ z\}^T$.

The geometric details of the fixed base and the end effector are shown in the next figures:

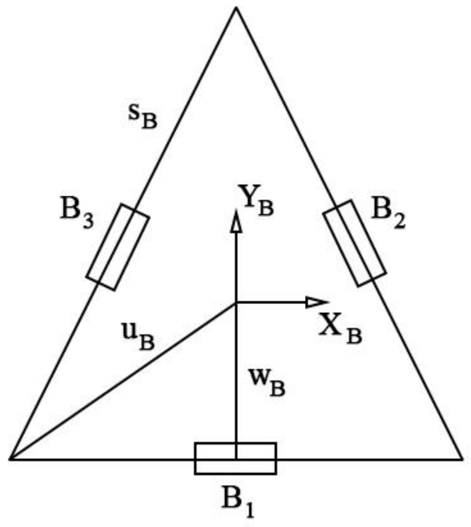


Figure 16 *Geometric details of the fixed base. w_B is the planar distance from {0} to near base side, u_B is the planar distance from {0} to a base vertex, $B_{i=1,2,3}$ are the position of the revolute joints in the fixed base.*

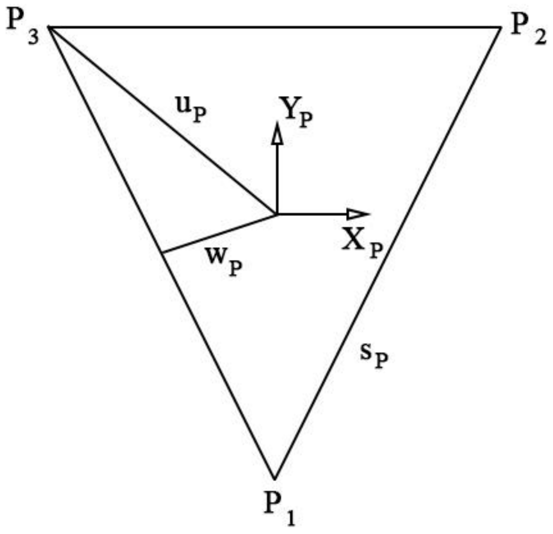


Figure 17 *Geometric details of the end effector base. w_P represents planar distance from {P} to near end effector side,u_P represents the planar distance from {P} to an end effector vertex, $P_{i=1,2,3}$ are the position where the lower arms are attached to the end effector.*

As shown in Figure 16, the points $B_i,\ i = 1, 2, 3$ are where the revolute joints are located in the fixed frame. And the points $P_i,\ i = 1, 2, 3$ are where the lower arms are attached to the end effector as shown in figure 17.

It can also be observed that the points B_i are constant in the base frame $\{B\}$ and the points P_i are constant in the frame $\{P\}$, therefore, the following data can be obtained:

$$ {}^{B}B_1 = \begin{Bmatrix} 0 \\ -w_B \\ 0 \end{Bmatrix} \qquad {}^{B}B_2 = \begin{Bmatrix} \frac{\sqrt{3}}{2} w_B \\ \frac{1}{2} w_B \\ 0 \end{Bmatrix} \qquad {}^{B}B_3 = \begin{Bmatrix} -\frac{\sqrt{3}}{2} w_B \\ \frac{1}{2} w_B \\ 0 \end{Bmatrix} \qquad \boldsymbol{eq.1} $$

$$ {}^{P}P_1 = \begin{Bmatrix} 0 \\ -u_P \\ 0 \end{Bmatrix} \qquad {}^{P}P_2 = \begin{Bmatrix} \frac{s_P}{2} \\ w_P \\ 0 \end{Bmatrix} \qquad {}^{P}P_3 = \begin{Bmatrix} -\frac{s_P}{2} \\ w_P \\ 0 \end{Bmatrix} \qquad \boldsymbol{eq.2} $$

The vertices of the fixed base are:

$$ {}^{B}b_1 = \begin{Bmatrix} \frac{s_B}{2} \\ -w_B \\ 0 \end{Bmatrix} \qquad {}^{B}b_2 = \begin{Bmatrix} 0 \\ u_B \\ 0 \end{Bmatrix} \qquad {}^{B}b_3 = \begin{Bmatrix} -\frac{s_B}{2} \\ -w_B \\ 0 \end{Bmatrix} \qquad \boldsymbol{eq.3} $$

Where:

$$ w_B = \frac{\sqrt{3}}{6} s_B \qquad u_B = \frac{\sqrt{3}}{3} s_B \qquad w_P = \frac{\sqrt{3}}{6} s_P \qquad u_P = \frac{\sqrt{3}}{3} s_P $$

It is important to take into account the real dimension of the ABB IRB340 because they define the particular solution of this manipulator. Figures 18 to 20 display the dimensions of the ABB robot and they are presented in Table 2.

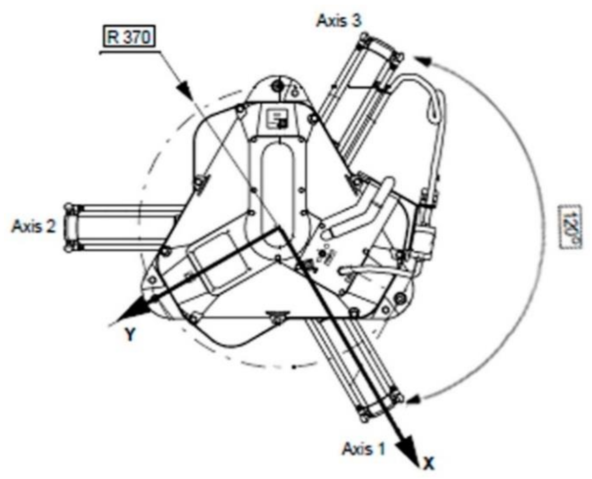


Figure 18 *Dimensions of the ABB IRB340 robot's fixed base. Modified from [29]*

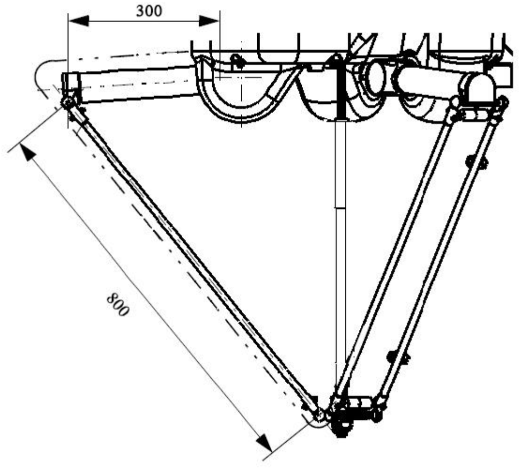


Figure 19 *Dimensions of the upper arms and lower arms. Modified from [29]*

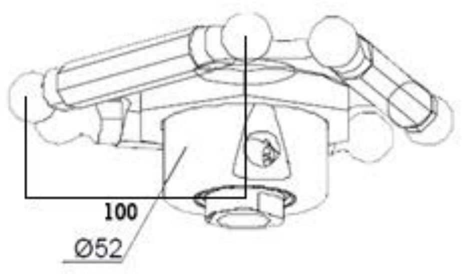


Figure 20 *Dimensions of the end effector. Modified from [29]*

Name	Meaning	Value (mm)
s_B	Fixed base equilateral triangle side.	640
s_P	End effector equilateral triangle side.	100
L	Upper legs length	300
l	Lower legs length	800
w_B	Planar distance from$\{0\}$ to near base side	180
u_B	Planar distance from $\{0\}$ to a base vertex	360
w_p	Planar distance from $\{P\}$ to near platform side	30
u_P	Planar distance from $\{P\}$ to a platform vertex	35

Table 2 *ABB IRB340 model measurements*

3.2.2 Inverse kinematics equations

From the kinematic diagram shown in Figure 15 a three vector loop closure equation for a delta robot can be found [28]:

$$\{{}^{B}B_i\} + \{{}^{B}L_i\} + \{{}^{B}l_i\} = \{{}^{B}P_P\} + [{}^{B}_{P}R]\{{}^{P}P_i\} = \{{}^{B}P_P\} + \{{}^{P}P_i\} \qquad i = 1, 2, 3 \qquad \boldsymbol{eq. 4}$$

Where $[{}^{B}_{P}R]$ = $[I_3]$, because there are no rotations allowed by the delta robot.

The lower leg lengths must have the correct constant length l (virtual length through the centre of each parallelogram)[28]:

$$l_i = \left\|\{{}^{B}l_i\}\right\| = \left\|\{{}^{B}P_P\} + \{{}^{P}P_i\} - \{{}^{B}B_i\} - \{{}^{B}L_i\}\right\| \qquad i = 1,2,3 \qquad \boldsymbol{eq.5}$$

In order to avoid square root in the Euclidean norms it is convenient to square both sides of eq. 5 [28]:

$$l_i^2 = \left\|\{{}^{B}l_i\}\right\|^2 = l_{ix}^2 + l_{iy}^2 + l_{iz}^2 \qquad i = 1,2,3 \qquad \boldsymbol{eq.6}$$

The vectors $\{{}^{B}L_i\}$ $i = 1,2,3$ are dependent on the joint variables given

$${}^{B}L_1 = \begin{Bmatrix} 0 \\ -Lcos\theta_1 \\ -Lsin\theta_1 \end{Bmatrix} \qquad {}^{B}L_2 = \begin{Bmatrix} \frac{\sqrt{3}}{2}Lcos\theta_2 \\ \frac{1}{2}Lcos\theta_2 \\ -Lsin\theta_2 \end{Bmatrix} \qquad {}^{B}L_3 = \begin{Bmatrix} -\frac{\sqrt{3}}{2}Lcos\theta_3 \\ \frac{1}{2}Lcos\theta_3 \\ -Lsin\theta_3 \end{Bmatrix} \qquad \boldsymbol{eq.7}$$

By substituting the equations given above into the vector loop equations (eq.4) it can be obtained:

$$\{{}^{B}l_1\} = \begin{Bmatrix} x \\ y + Lcos\theta_1 + a \\ z + Lsin\theta_1 \end{Bmatrix} \{{}^{B}l_2\} = \begin{Bmatrix} x - \frac{\sqrt{3}}{2}Lcos\theta_2 + b \\ y - \frac{1}{2}Lcos\theta_2 + c \\ z + Lsin\theta_2 \end{Bmatrix} \{{}^{B}l_3\} = \begin{Bmatrix} x + \frac{\sqrt{3}}{2}Lcos\theta_3 - b \\ y - \frac{1}{2}Lcos\theta_3 + c \\ z + Lsin\theta_3 \end{Bmatrix} \boldsymbol{eq.8}$$

Where:

$$a = w_B - u_P \qquad b = \frac{s_P}{2} - \frac{\sqrt{3}}{2}w_B \qquad c = w_P - \frac{1}{2}w_B$$

The three constraint equations yields the kinematics equations for the delta robot:

$$2L(y + a)cos\theta_1 + 2zLsin\theta_1 + x^2 + y^2 + z^2 + a^2 + L^2 + 2ya - l^2 = 0 \qquad \boldsymbol{eq.9}$$

$$-L\left(\sqrt{3}(x + b) + y + c\right)cos\theta_2 + 2zLsin\theta_2 + x^2 + y^2 + z^2 + b^2 + c^2 + L^2 + 2xb + 2y - l^2 = 0 \quad \boldsymbol{eq.10}$$

$$L\left(\sqrt{3}(x-b)-y-c\right)cos\theta_3+2zLsin\theta_3+x^2+y^2+z^2+b^2+c^2+L^2-2xb+2yc-l^2=0 \quad \boldsymbol{eq.11}$$

The three absolute vector knee points are found using ${}^{B}A_i = {}^{B}B_i + {}^{B}L_i,\ i=1,2,3$:

$${}^{B}A_1=\begin{Bmatrix}0\\-w_B-Lcos\theta_1\\-Lsin\theta_1\end{Bmatrix}\ {}^{B}A_2=\begin{Bmatrix}\frac{\sqrt{3}}{2}(w_B+Lcos\theta_2)\\\frac{1}{2}(w_B+Lcos\theta_2)\\-Lsin\theta_2\end{Bmatrix}\ {}^{B}A_3=\begin{Bmatrix}-\frac{\sqrt{3}}{2}(w_B+Lcos\theta_3)\\\frac{1}{2}(w_B+Lcos\theta_3)\\-Lsin\theta_3\end{Bmatrix}\ \boldsymbol{eq.12}$$

Referring to Figure 15, the inverse kinematics can be solved independently for each of the three upper arms. Geometrically each leg is the intersection between a known circle of radio L centred on the base of the joint point ${}^{B}B_i$ and a known sphere of radio l centred on the moving platform vertex ${}^{P}P_i$ [28]. Using the three constraint equations applied to the vector loop closure equation (eq. 4). The three independent scalar inverse position kinematics are of the form [28]:

$$E_i\cos\theta_1+F_i\sin\theta_1+G_i=0 \qquad i=1,2,3 \qquad \boldsymbol{eq.13}$$

Where:

$$E_1=2L(y+a)$$
$$F_1=2zL$$
$$G_1=x^2+y^2+z^2+a^2+L^2+2ya-l^2$$

$$E_2=-L(\sqrt{3}(x+b)+y+c)$$
$$F_2=2zL$$
$$G_2=x^2+y^2+z^2+b^2+c^2+L^2+2(xb+yc)-l^2$$

$$E_3=L(\sqrt{3}(x-b)-y-c)$$
$$F_3=2zL$$
$$G_3=x^2+y^2+z^2+b^2+c^2+L^2+2(-xb+yc)-l^2$$

Using the tangent half angle substitution [28]:

If we define:

$$t_i = tan\frac{\theta_i}{2} \qquad \boldsymbol{eq.14}$$

Then:

$$cos\theta_i = \frac{1 - t_i^2}{1 + t_i^2} \qquad \boldsymbol{eq.15}$$

And:

$$sin\theta_i = \frac{2t_i}{1 + t_i^2} \qquad \boldsymbol{eq.16}$$

Substituting the tangent half angle into the E, F, G equations:

$$E_i\left(\frac{1 - t_i^2}{1 + t_i^2}\right) + F_i\left(\frac{2t_i}{1 + t_i^2}\right) + G_i = 0$$

$$E_i(1 - t_i^2) + F_i(2t_i) + G_i(1 + t_i^2) = 0$$

$$(G_i - E_i)t_i^2 + (2F_i)t_i + (G_i + E_i) = 0 \qquad \boldsymbol{eq.17}$$

Yielding to a quadratic formula:

$$t_{1,2} = \frac{-F_i \pm \sqrt{E_i^2 + F_i^2 - G_i^2}}{G_i - E_i} \qquad \boldsymbol{eq.18}$$

Solve for θ_i by inverting the tangent half angle equation [28]:

$$\theta_i = 2tan^{-1}(t_i) \qquad \boldsymbol{eq.19}$$

From eq. 18 it is known that there are two possible solutions of the quadratic formula, both solutions are correct. Nevertheless, the solutions that are within the working range of the robot will be chosen. In Figure 21 and Table 3, the extreme positions of the arms are presented.

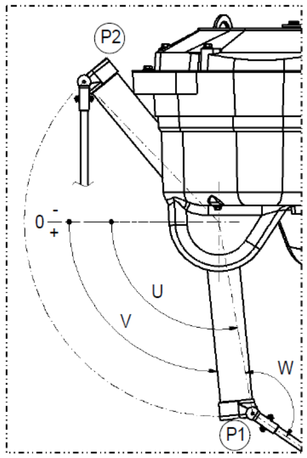


Figure 21 *Extreme positions of robot arms. Image from [29]*

Position P1
U = 100°
V = 95.5°
W = 134.5°

Position P2
U = -46.1°
V = -50.6°
W = 43.9°

Mechanical stop: When angle V = -57°

Table 3 *Extreme values of arm angles*

3.2.3 Forward kinematics equations

As mentioned in section 2.3.2, the position of the end effector can be calculated from the angular position of the upper arms. Hence, given the actuated joint angles, the resulting Cartesian coordinates of the end effector can be found.

According to [28], from the forward position kinematics diagram in Figure 22, it is possible to define three virtual sphere centres as ${}^{B}A_{iv} = {}^{B}A_{i} - {}^{P}P_{i}, i = 1, 2, 3$, obtaining:

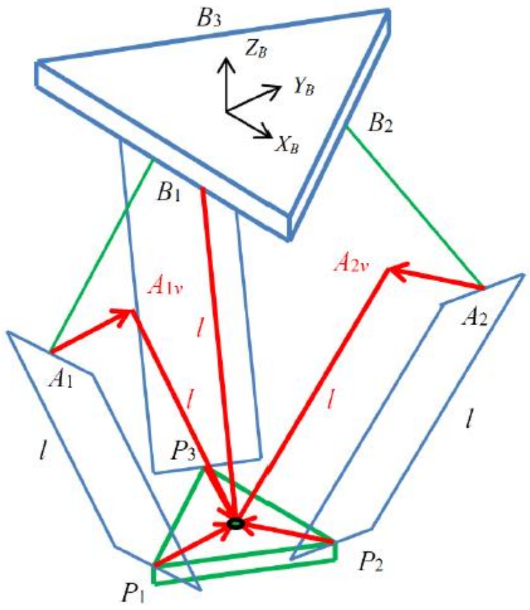


Figure 22 *Forward position kinematics diagram. The red lines show the vector centre point A_{iv} and the radius l of the sphere that allow the forward kinematics solution. Image from [28]*

$$ {}^{B}A_{1v} = \begin{Bmatrix} 0 \\ -w_B - Lcos\theta_1 + u_p \\ -Lsin\theta_1 \end{Bmatrix} \qquad {}^{B}A_{2v} = \begin{Bmatrix} \frac{\sqrt{3}}{2}(w_B + Lcos\theta_2) - \frac{s_p}{2} \\ \frac{1}{2}(w_B + Lcos\theta_2 - w_p \\ -Lsin\theta_2 \end{Bmatrix} $$

$$ {}^{B}A_{2v} = \begin{Bmatrix} -\frac{\sqrt{3}}{2}(w_B + Lcos\theta_3) + \frac{s_p}{2} \\ \frac{1}{2}(w_B + Lcos\theta_3 - w_p \\ -Lsin\theta_3 \end{Bmatrix} \quad \boldsymbol{eq.20} $$

Then, the forward kinematics solution is the intersection of three known spheres. Each sphere is define as a vector centre point $\{c\}$ and a scalar radius r. Thus, the sphere would be of the form:

$$(\{^{B}A_{1v}\}, l) \qquad (\{^{B}A_{2v}\}, l) \qquad (\{^{B}A_{3v}\}, l)$$

Having the vector centre point and the radius of the sphere, it is possible to derive the equations for the three spheres intersection [28]. This equations are presented in appendix A. It is worth mentioning that following the solution in Appendix A there will be a singularity that prevents a successful solution [28] when all the three sphere centres have the same height in Z, it is to say, the angular parameters θ_i are similar. A solution for this case [28] is presented in appendix B.

3.3 Inverse kinematics solution

In order to solve the inverse kinematics equations found in section 3.2.1 an algorithm is developed in MATLAB. As mentioned in section 3.2.2, the solution of inverse kinematics is given by eq. 18, which is quadratic, hence two possible solutions can be obtained. Table 4 presents the two possible solutions given proposed coordinates of the end effector.

As can be seen, the results obtained in the column with the name "Solution 2", are not within the working range of the robot (according to Table 3), therefore, the values shown in column "Solution 1" are the correct angles for moving the arms and bringing the end effector to the desired position.

Proposed coordinates	Solution 1	Solution 2

$(-0.2, 0.2, -0.6)$	Arm1 = 0.447175
	Arm2 = 0.109678
	Arm3 = -0.697459
		Arm1 = -2.545099
		Arm2 = -2.623097
		Arm3 = -2.938814

$(-0.5, -0.2, -0.8)$	Arm1 = 0.725058
	Arm2 = 1.694179
	Arm3 = 0.423712
		Arm1 = 2.279251
		Arm2 = 2.819121
		Arm3 = 2.201392

$(0.125, -0.367, -0.523)$	Arm1 = -1.070931
	Arm2 = 0.076695
	Arm3 = 0.566685
		Arm1 = -2.873512
		Arm2 = -2.497997
		Arm3 = -2.370386

$(0, 0, -0.75)$	Arm1 = 0.264188
	Arm2 = 0.220808
	Arm3 = 0.220808
		Arm1 = -3.023826
		Arm2 = -3.040674
		Arm3 = -3.040674

$(0.312, 0.349, -0.856)$	Arm1 = 1.607447
	Arm2 = 0.236494
	Arm3 = 1.231427
		Arm1 = 2.580972
		Arm2 = 2.183461
		Arm3 = 2.407590

Table 4 *Two possible solutions obtained using inverse kinematics equations.*

3.3.1 Validation of results

In order to validate the solutions given in Table 4, the forward kinematics equations are used. As mentioned in section 2.3.2, the position of the end effector can be calculated from the angular position of the upper arms. Hence, given the actuated joint angles, the resulting Cartesian coordinates of the end effector can be found. The angular parameters given in Table 4 are validated using them as input values in the forward kinematics calculations and the outcome is presented in Table 5.

Proposed coordinates	Calculated Angles	Obtained Coordinates

(-0.2, 0.2, -0.6)	θ_1 = 0.447175
	θ_2 = 0.109678
	θ_3= -0.697459
		x = -0.200000
		y = 0.200000
		z = -0.600000

(-0.5, -0.2, -0.8)	θ_1 = 0.725058
	θ_2 = 1.694179
	θ_3 = 0.423712
		x = -0.500000
		y = -0.200000
		z = -0.800000

(0.125, -0.367, -0.523)	θ_1 = -1.070931
	θ_2 = 0.076695
	θ_3 = 0.566685
		x = 0.125000
		y = -0.367000
		z = -0.523000

(0, 0, -0.75)	θ_1 = 0.264188
	θ_2 = 0.220808
	θ_3 = 0.220808
		x = 0.000000
		y = -0.013362
		z = -0.758460

(0.312, 0.349, -0.856)	θ_1 = 1.607447
	θ_2 = 0.236494
	θ_3 = 1.231427
		x = 0.312000
		y = 0.349000
		z = -0.856000

Table 5 *Solutions using Forward kinematics equations. This table shows resulting coordinates of the end effector given angles θ_i calculated by inverse kinematics. It can be seen, the obtained coordinates and proposed coordinates match each other.*

The data in Table 5 verify the correctness of the results obtained in section 3.3. Table 5 in line 4 shows the special case mentioned in section 3.2.3, where θ_i are similar. For this case in particular, the forward kinematics results and the proposed coordinates do not match exactly. There is a small inaccuracy and it is the result of the assumptions made in the mathematical solution.

3.3.2 Following trajectory

In this section a motion plan is proposed, called desired position. By using inverse kinematics the angles of the upper arms can be calculated and those angles can be used in direct kinematics equations to calculate positions, called obtained position. In this way, by comparing the desired and obtained positions it is possible to demonstrate the accuracy of the results. Appendix C shows the script for the comparison, obtained from [30].

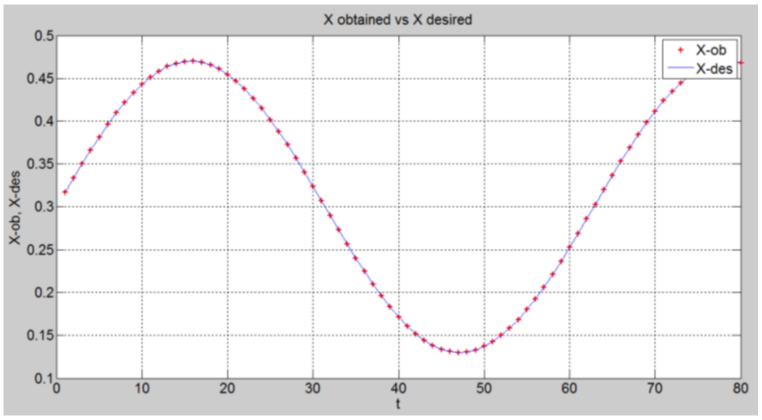


Figure 23 *Desired trajectory vs obtained trajectory in X axis against time. Blue line is the desired coordinates, red crosses are the obtained coordinates.*

In Figure 23 it can be observe that the desired position proposed in the X axis (blue line) and the obtain positions using kinematics equations (red crosses) match each other, in other words, the positions obtained by means of kinematics equations are calculated correctly.

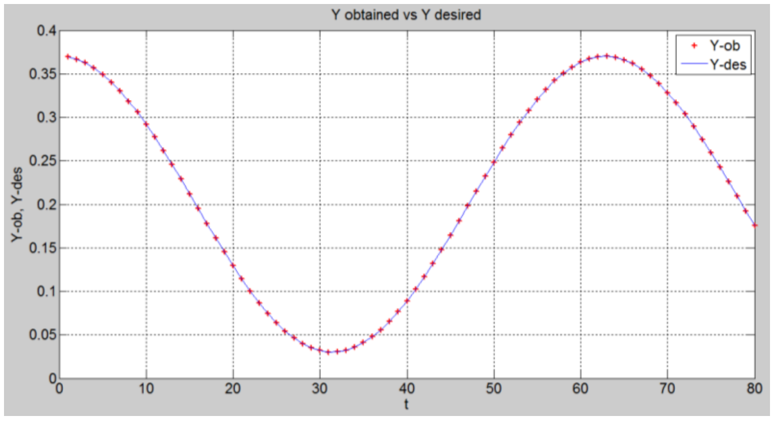


Figure 24 *Desired trajectory vs obtained trajectory in Y axis against time. Blue line is the desired coordinates, red crosses are the obtained coordinates.*

From Figure 24 the desired position in the Y axis (blue line) and the obtain positions (red crosses) match each other. The motion obtained via kinematics equations behave in the same manner than the desired position. From Figure 24 it is also proved that the obtained results are correct.

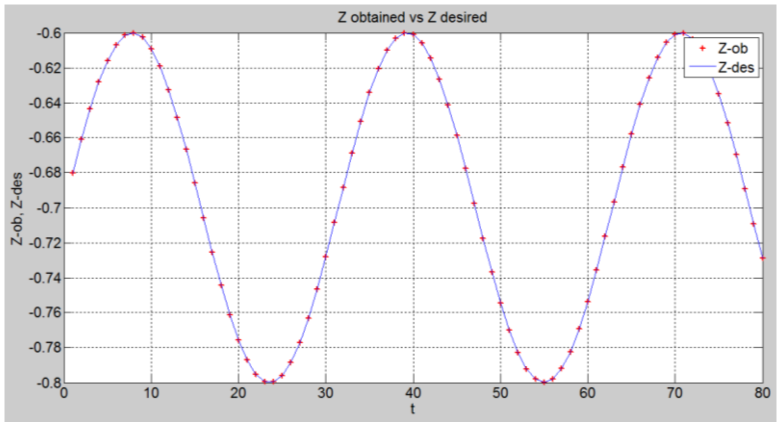


Figure 25 *Desired trajectory vs obtained trajectory in Z axis against time. Blue line is the desired coordinates, red crosses are the obtained coordinates.*

From Figure 25 the desired position in the Z axis (blue line) and the obtain positions (red crosses) match each other. From Figure 25 it can be seen that the obtained results are correct.

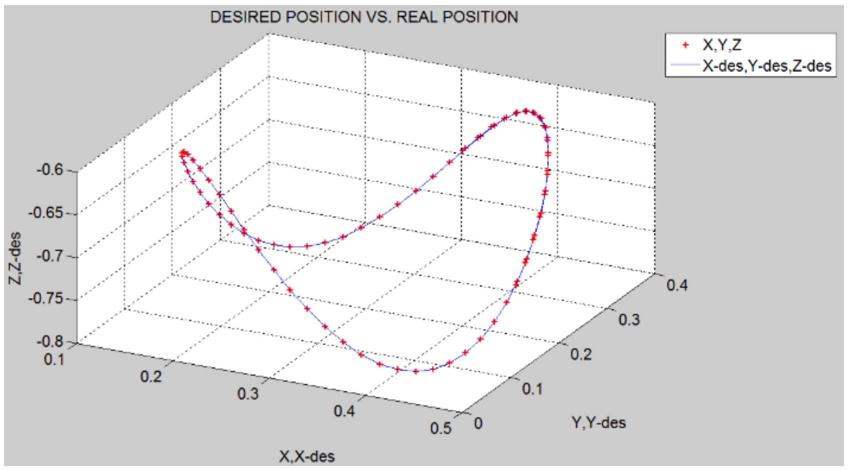


Figure 26 *Three dimensional plot of desired trajectory vs obtain trajectory given a set of points in X, Y, Z. Blue line is the desired coordinates, red crosses are the obtain coordinates.*

Figure 26 shows a 3 dimensional plot of the desired trajectory to follow in the X, Y and Z axes. This plot gives a better idea of the full motion desired of the robot and the full motion obtain according to the kinematics equations previously derived.

Chapter 4

Development of a URDF file for simulation and programming of a delta robot using ROS

This chapter presents the development of the URDF file of a delta robot for visualizing it in RViz. Also, a python script is developed to calculate inverse kinematics equations and to publish joint_states messages to move the robot model.

4.1 Problem statement

It is desired to create a 3D model of an IRB340 delta robot using the URDF package of ROS. Also, it is necessary the creation of a motion routine to visualize the active operation of the 3D model.

4.2 Methodology

4.2.1 URDF model

As explained in section 2.8, a URDF is used to obtain 3D representations of real robots. During the creation of a 3D model using URDF it is possible to customize characteristics such as shape and colour.

One of the limitations of URDF is that closed kinematic chains cannot be simulated. Because it is necessary to follow a tree structure when designing the model, that is, a parent link can have many children, but a child link cannot have many parents. It is known that parallel robots have a closed kinematic chain, thus a modification should be done to permit the creation of the model.

For this reason the robot is designed using three kinematics arms and only one arm is connected to the end effector.

In order to create a model based on the ABB IRB340 the data presented in Table 2 is used to assign the link parameters of the model. The thickness and width of the created links are constant (0.02 m). The base_link of the 3D model would be the fixed base of the robot, according to this, a link with cylindrical geometry and radius $w_B = 0.180$ is created as display in Figure 27.

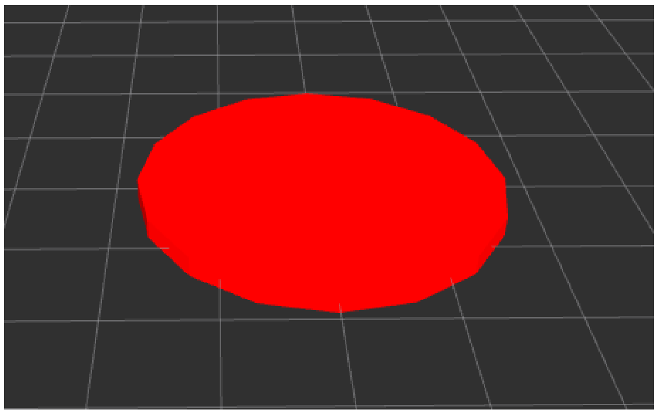


Figure 27 *Fixed base of the 3D model. It can be observed that the geometry is a cylinder and the radius of the base is* $w_B = 180$.

The next step is to add the three upper arms. These three links have the geometry of a box and their length is $L = 300$. As told in section 3.2.1, points B_i are where the arms are connected to the base through revolute joints. Therefore the position of the joints an upper arms is given by eq. 1. And the orientation is deducted considering the arms are separated 120 degrees of each other.

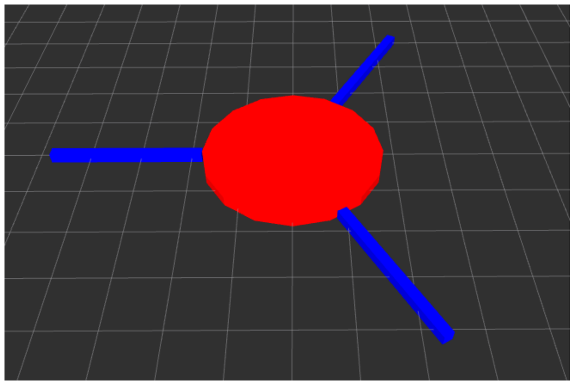


Figure 28 *Upper arms of the delta robot mode attached to the fixed base. The upper arms have a box geometry and its length is* $L = 300$.

The lower arms, like upper arms are links with box geometry and length $l = 800$.

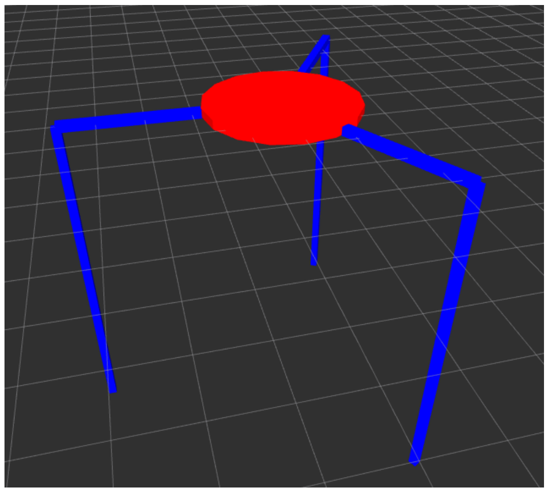


Figure 29 *Lower arms of the delta robot. It can be seen that the lower arms are attached to the upper arms and have box geometry with length* $l = 800$.

As told before in this section, just one arm can be connected to the end effector due to the obliged tree structure of the URDF models, the end effector has a cylindrical geometry of radius $s_p = 100$.

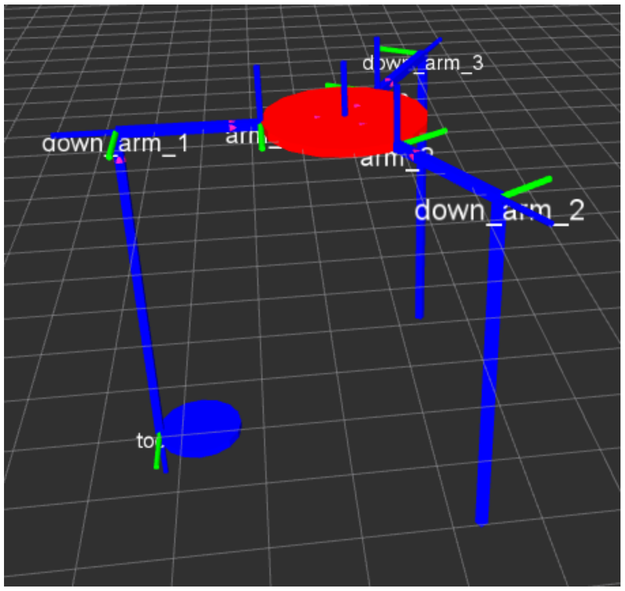


Figure 30 *3D model of the ABB IRB340 delta robot in RViz. In this model, it is observe that the three lower arms are separated and the end effector is attached to one lower arm of the robot. The model is created in this manner because a tree structure is needed for the creation of a URDF.*

The joints in URDF files define: the connecting nodes between parent and child links, the origin of the joint, the rotation around a fixed axis, safety limits, type of joints (revolute, prismatic, fixed, etc.), among others.

The IRB340 robot is composed by 9 joints, three of them are revolute joints and the other six are universal joints. A simple leg of the delta robot has a revolute joint attached to the fixed base, then the upper arm is connected with the lower arm via a universal joint and the lower arm is attached to the end effector using a universal joint. It is to say, one leg of the robot has one revolute joint and two universal joints to interconnect the fixed base, the upper arm, the lower arm and the end effector.

The URDF do not have the model of universal joints and for this reason this model uses only revolute joints. This type of joints rotate along a specified axis and has a limited range specified by the upper and lower limits. For a bigger overview of the 3D model of the robot developed in ROS, the complete URDF file is presented in appendix D.

4.2.2 Motion plan

In order to make the model to move, it is necessary the creation of a node that calculates the inverse kinematics and then send the calculated parameter to the joints of the URDF model. The algorithm used to calculate the inverse kinematics in ROS, is basically the same than the mentioned in section 3.3. But this one was created using python in order to be compatible with ROS. The outcome of the inverse kinematics calculations, the angles of joints, are sent to the URDF model using sensor_msgs/JointState message format through a joint_state_publisher. Appendix E shows the node described above.

4.3 Moving the 3D model inside RViz

Due to the physical characteristics of revolute joints, the joint displacement is limited to one axis. Therefore, the motion plan in this section is limited to the movement of the robot model along the Z axis. To observe the motion of the model, three proposed coordinates are given. The ROS node is in charge of calculating the desired angles and send these values to the joints in the 3D model to perform the motion task.

Desired position of end effector	Calculated angles of each arm

x = 0.0
y = 0.0
z = -0.5
	θ_1= -0.821975
	θ_2 = -0.938774
	θ_3 = -0.938774

x = 0.0
y = 0.0
z = -0.7
	θ_1 = 0.113260
	θ_2 = 0.065934
	θ_3 = 0.065934

x = 0.0
y = 0.0
z = -1.0
	θ_1 = 1.033750
	θ_2 = 0.998397
	θ_3 = 0.998397

Table 6 *Table presenting the desired position of the end effector and the calculated angles using inverse kinematics in ROS.*

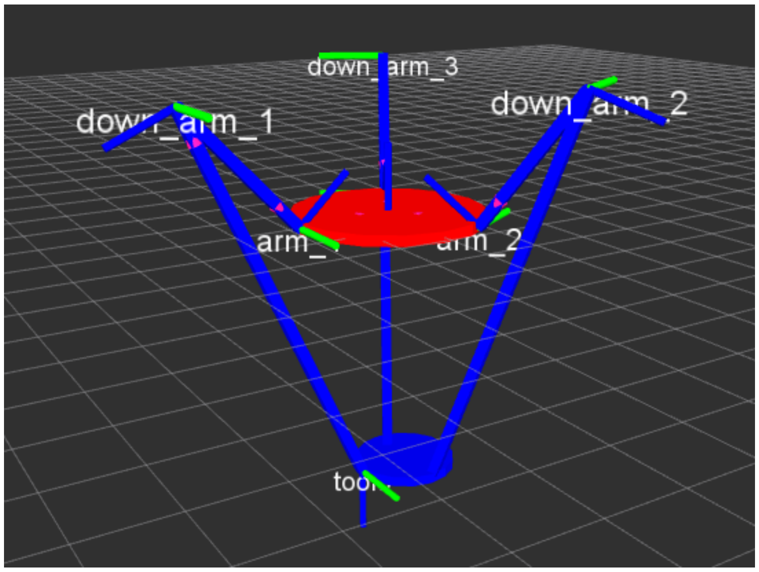


Figure 31 *Motion performance of the 3D model inside RViz. The coordinates given for this task are (0, 0,- 0.5)*

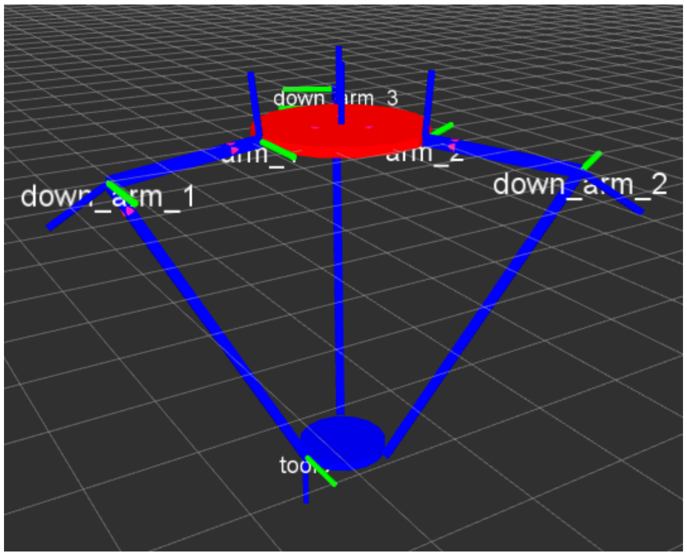


Figure 32 *Motion performance of the 3D model inside RViz. The coordinates given for this task are (0, 0,- 0.7)*

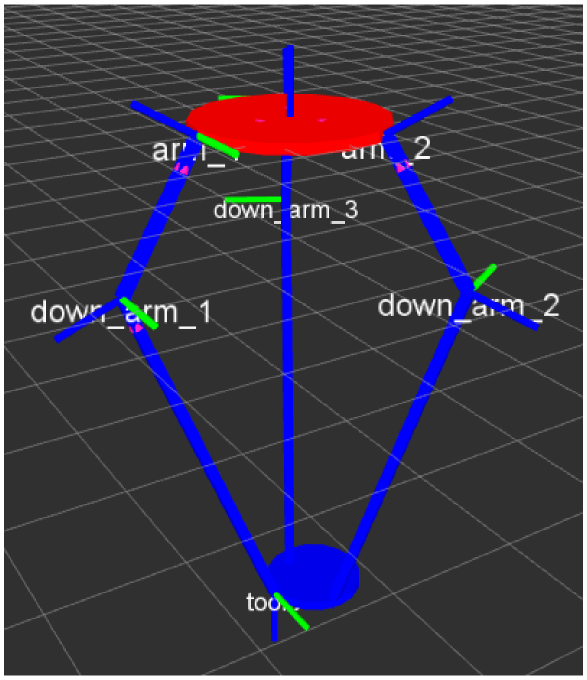


***Figure 33** Motion performance of the 3D model inside RViz. The coordinates given for this task are (0, 0,- 1.0)*

Figures 31 to 33 show the motion of the 3D model following the given positions in Table 6.

4.4 Delta robot model in MATLAB

A model of the delta robot is created in MATLAB in order to visualize the motion of the robot within its complete workspace. Appendix F shows the script of the routine.

Figures 34 to 38 show the motion of the model in its entire workspace. This motion plan contains the five desired positions mentioned in Table 4.

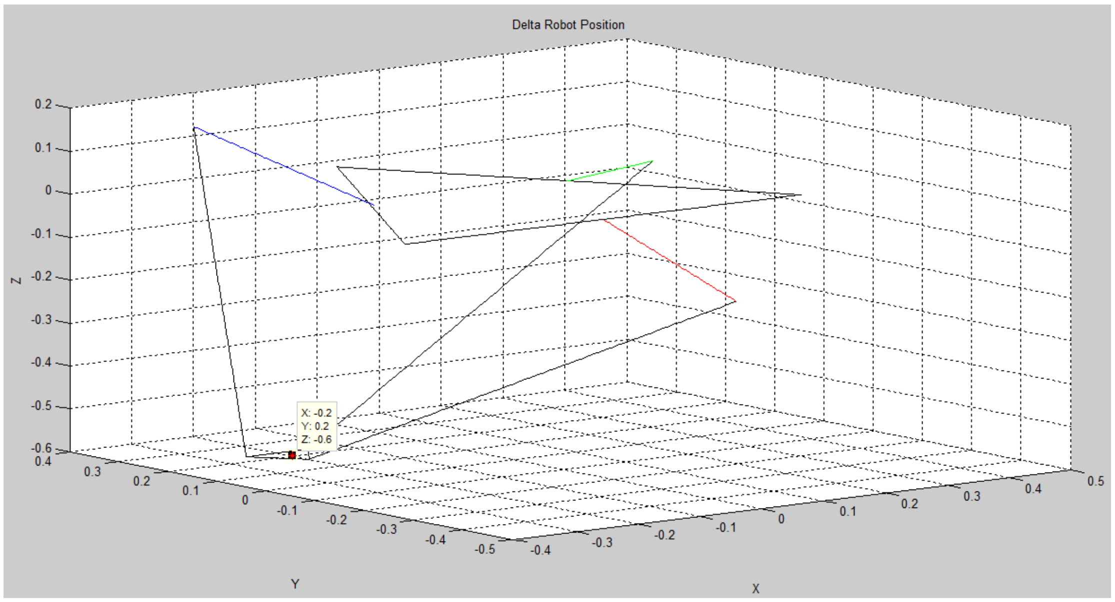


Figure 34 *3D model motion in MATLAB. The given coordinates of the end effector are (X=-0.2, Y=0.2, Z=-0.6).*

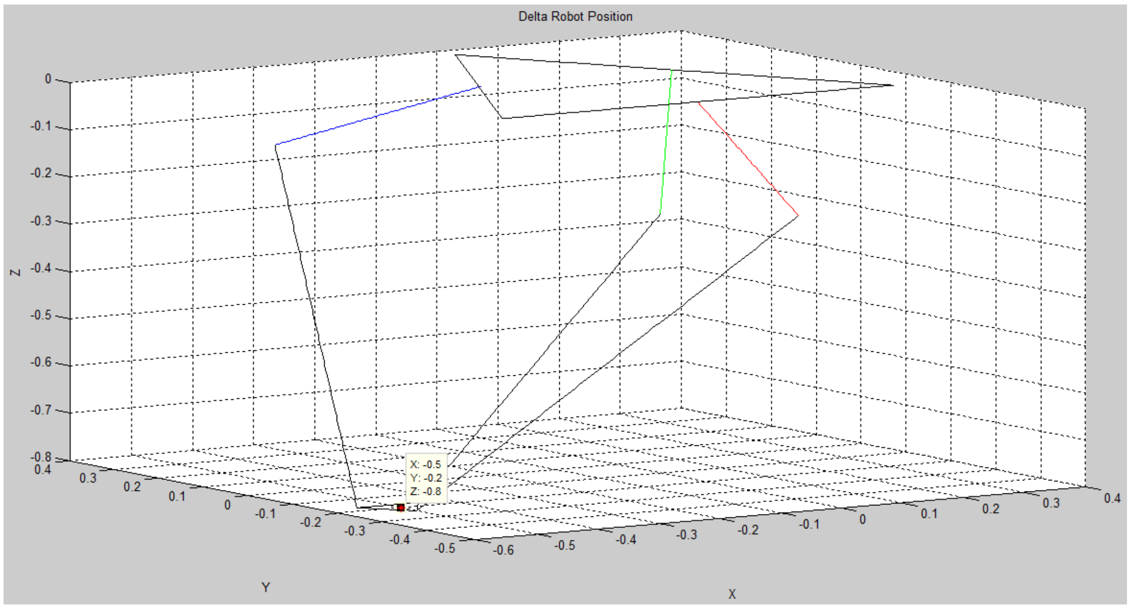


Figure 35 *3D model motion in MATLAB. The given coordinates of the end effector are (X=-0.5, Y=-0.2, Z=-0.8).*

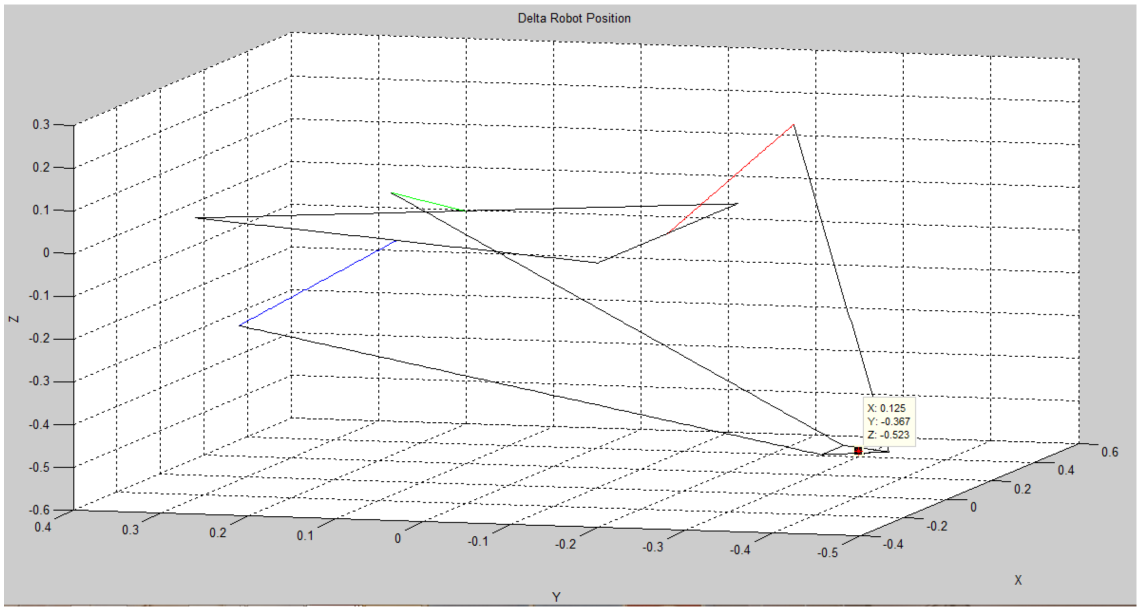


***Figure 36** 3D model motion in MATLAB. The given coordinates of the end effector are (X=-0.125, Y=-0.367, Z=-0.523).*

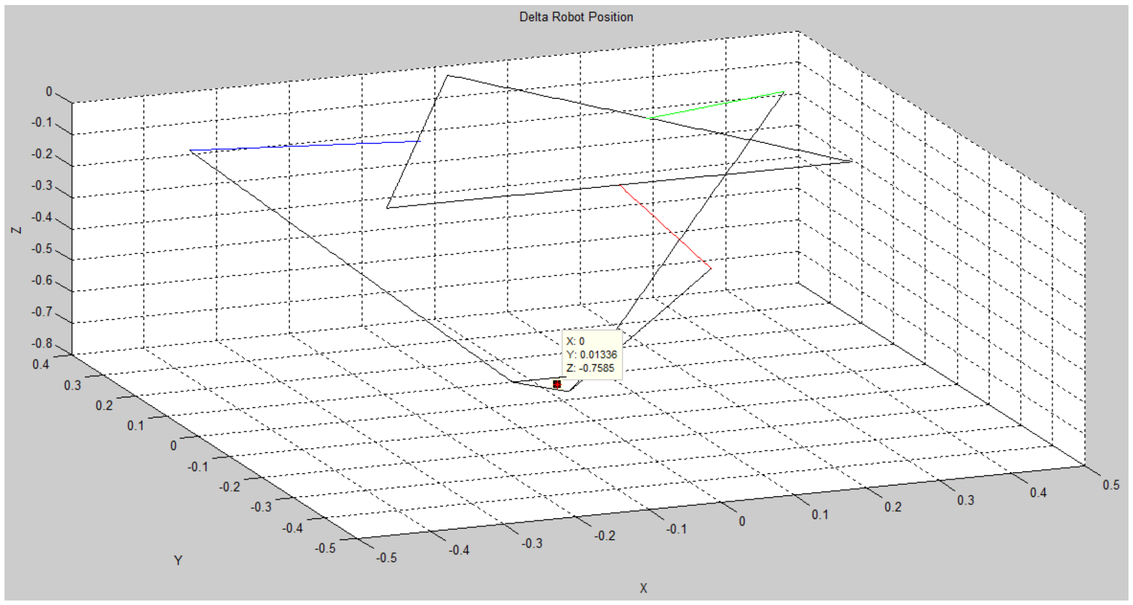


***Figure 37** 3D model motion in MATLAB. The given coordinates of the end effector are (X=-0.0, Y=0.013362, Z=-0.758460).*

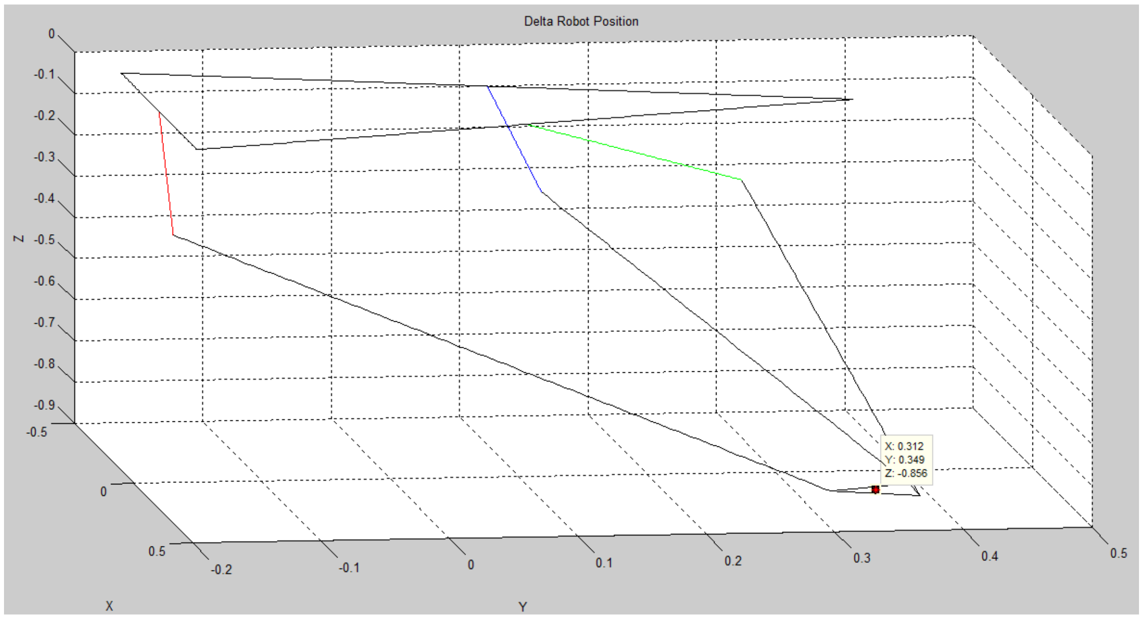


Figure 38 *3D model motion in MATLAB. The given coordinates of the end effector are (X=-0.312, Y=0.349, Z=-0.856).*

Chapter 5

Conclusions and future work

This work aims the creation of a 3D model of the IRB340 robot and the simulation of its operation. This objective was carried out by dividing the job into two tasks: the derivation of the robot’s kinematics and the creation of the model.

The robot motion problem was stated in such a way that the only information available to calculate the robot state was the position of the end effector. For this reason, the feasible option to solve the problem was the use of inverse kinematics. This was the tool used to describe the motion of the bodies that comprehend the robot’s body. The results obtained from the inverse kinematics calculations were validated and in this way confirm that the data used to create the motion plan of the robot was accurate.

The graphical design of the robot was done using ROS packages. This had the intention of implement a motion routine in the digital robot that can, in the future, be implemented in the real robot. As it was mentioned previously, ROS is in continuous development and there are some limitation, like for example the types of joints available in the URDF package and the tree structure of links that do not allow parallel structures.

The types of joints available in the URDF package represent a limitation to this project. As mentioned before the IRB340 is a robot that possess 9 joints, 6 of them are universal and this type of joints are not available in the URDF package. Therefore, it was not possible to add universal joints to the 3D model. Instead, revolute joints were used, which has only 1DOF. This had as consequence the limitation of the movement of the robot to the displacement along the Z axis. As an alternative, it was introduced a representation of the robot in MATLAB to observe the operation of the robot in its entire workspace.

Finally, according to the results mentioned above, it is possible to propose improvements or alternatives to the presented work.

The main limitation during the development of this project was the type of joints available in the URDF packages. In order to solve this problem, two alternatives are proposed. The first one consist in the modelling of the universal joint to be added to the URDF, as it is possible to make contributions to ROS packages. The second alternative is the use of floating joints instead of universal joints. This type of joints has 6DOF, but they cannot be controlled using the joint_state_publisher.

Another proposal is the control of the physical robot using ROS. The delta robot IRB340 uses an IRC5 controller to send desired motions tasks to the manipulator. Thus, it is possible to controller the real robot by sending the calculated angles to the controller using a specific ROS topic. There are projects integrating ROS and the IRC5 controller, but there is no testing implementations using delta robots.

Bibliography

[1] B. Siciliano, L. Sciavico, L. Villani, and J. Oriolo, *Robotics: Modeling, Planning and Control*. 2010.

[2] I. Bonev, “Delta parallel robot-the story of success,” *Newsletter*, no. 4, pp. 1–8, 2001.

[3] P. Prempraneerach, “Workspace and dynamic trajectory tracking of delta parallel robot,” *2014 Int. Comput. Sci. Eng. Conf. ICSEC 2014*, pp. 469–474, 2015.

[4] K. Waldron, K. Waldron, J. Schmiedeler, and J. Schmiedeler, “Kenneth Waldron, James Schmiedeler,” *Springer Handb. Robot.*, pp. 9–33, 2008.

[5] C. Mitsantisuk, S. Stapornchaisit, N. Niramitvasu, and K. Ohishi, “Force Sensorless Control with 3D Workspace Analysis for Haptic Devices based on Delta Robot,” pp. 1747–1752, 2015.

[6] M. Mustafa, R. Misuari, and H. Daniyal, “Forward kinematics of 3 degree of freedom delta robot,” *2007 5th Student Conf. Res. Dev. SCORED*, no. December, pp. 3–6, 2007.

[7] M. Applications, “Irb 340,” pp. 4–5.

[8] J. J. Craig, “Introduction to Robotics: Mechanics and Control 3rd,” *Prentice Hall*, vol. 1, no. 3. p. 408, 2004.

[9] S. Stapornchaisit, C. Mitsantisuk, S. Srisonphan, N. Chayopitak, and Y. Koike, “Micro-macro bilateral in task space for delta robot by using forward and inverse kinematic,” *IEEE Reg. 10 Annu. Int. Conf. Proceedings/TENCON*, vol. 2015–Janua, 2015.

[10] Open Source Robotics Foundation, “About ROS.” [Online]. Available: http://www.ros.org/about-ros/. [Accessed: 20-Oct-2016].

[11] Open Source Robotics Foundation, “Documentation.” [Online]. Available: http://wiki.ros.org/. [Accessed: 20-Oct-2016].

[12] Open Source Robotics Foundation, “Introduction.” [Online]. Available: http://wiki.ros.org/ROS/Introduction. [Accessed: 20-Oct-2016].

[13] Open Source Robotics Foundation, “Is ROS for me?” [Online]. Available: http://www.ros.org/is-ros-for-me/. [Accessed: 22-Oct-2016].

[14] M. Lucas, *Absolute BSD*, vol. 1. William Pollock, 2012.

[15] Open Source Robotics Foundation, “Concepts.” [Online]. Available:

http://wiki.ros.org/ROS/Concepts. [Accessed: 22-Oct-2016].

[16] L. Joseph, *Mastering ROS for Robotics Programming*. 2015.

[17] A. Martinez., E. Fernandez, A. Mahtani., and L. S. Crespo., *Learning ROS for Robotics Programming*. 2015.

[18] J. M. O'Kane, *A gentle introduction to ROS*. 2013.

[19] Open Source Robotics Foundation, "tf." [Online]. Available: http://wiki.ros.org/tf. [Accessed: 03-Nov-2016].

[20] M. Quigley *et al.*, "ROS: an open-source Robot Operating System," *Icra Work. Open Source Softw.*, vol. 3, no. Figure 1, p. 5, 2009.

[21] T. Foote, "tf : The Transform Library."

[22] L. H. Kubota, Naoyuki, kiguchi, Kazuo, *Intelligent Robotics and Applications*. 2016.

[23] ROS-Industrial, "Our Brief History." [Online]. Available: http://rosindustrial.org/briefhistory/.

[24] Open Source Robotics Foundation, "Integration with Other Libraries." [Online]. Available: http://www.ros.org/integration/. [Accessed: 28-Oct-2016].

[25] ROS-Industrial, "ROS-Industrial," 2016. [Online]. Available: http://rosindustrial.org/about/description/. [Accessed: 03-Jan-2017].

[26] Open Source Robotics Foundation, "Industrial." [Online]. Available: http://rosindustrial.org/about/description/. [Accessed: 05-Jan-2017].

[27] S. Staicu, "Recursive modelling in dynamics of Agile Wrist spherical parallel robot," *Robot. Comput. Integr. Manuf.*, vol. 25, no. 2, pp. 409–416, 2009.

[28] R. L. . Williams, "The Delta Parallel Robot : Kinematics Solutions," no. 4, pp. 1–46, 2015.

[29] ABB Company, "Product Specification IRB340."

[30] J. A. Velarde Sánchez, "Cotrol simultáneo de fuerza y posición de un robot industrial," CIDESI, 2011.

Appendix A – Three spheres intersection algorithm

In order to solve this algorithm it is necessary to have three spheres, $(c_1, r_1), (c_2, r_2), (c_3, r_3)$. For the solution of the algorithm is necessary to know the vectors centres c_i and radius r. The centre vectors are presented in eq.20 and the radius is l. The equations of the three spheres are:

$$\begin{aligned}(x-x_1)^2+(y-y_1)^2+(z-z_1)^2&=r_1^2\\(x-x_2)^2+(y-y_2)^2+(z-z_2)^2&=r_2^2 \qquad eq.1\\(x-x_3)^2+(y-y_3)^2+(z-z_3)^2&=r_3^2\end{aligned}$$

The first step is to square the left side terms, then it is needed to perform mathematical operations to eliminate the square unknown variables, yielding to:

$$a_{11}x+a_{12}y+a_{13}z=b_1 \qquad eq.2$$
$$a_{21}x+a_{22}y+a_{23}z=b_2 \qquad eq.3$$

Where:

$$a_{11}=2(x_3-x_1) \quad a_{21}=2(x_3-x_2) \qquad b_1=r_1^2-r_3^2-x_1^2-y_1^2-z_1^2+x_3^2+y_3^2+z_3^2$$
$$a_{12}=2(y_3-y_1) \quad a_{22}=2(y_3-y_2) \qquad b_2=r_2^2-r_3^2-x_2^2-y_2^2-z_2^2+x_3^2+y_3^2+z_3^2$$
$$a_{13}=2(z_3-z_1) \quad a_{23}=2(z_3-z_2)$$

Solving for z in eq.2 and eq.3:

$$z=\frac{b_1}{a_{13}}-\frac{a_{11}}{a_{13}}x-\frac{a_{12}}{a_{13}}y \qquad eq.4$$
$$z=\frac{b_2}{a_{23}}-\frac{a_{21}}{a_{23}}x-\frac{a_{22}}{a_{23}}y \qquad eq.5$$

From eq.4 and eq.5 it is possible to eliminate z and obtain $x=f(y)$

$$x = f(y) = a_4 y + a_5 \qquad eq.6$$

Where:

$$a_4 = -\frac{a_2}{a_1} \quad a_5 = -\frac{a_3}{a_1} \quad a_1 = \frac{a_{11}}{a_{13}} - \frac{a_{21}}{a_{23}} \quad a_2 = \frac{a_{12}}{a_{13}} - \frac{a_{22}}{a_{23}} \quad a_3 = \frac{b_2}{a_{23}} - \frac{b_1}{a_{13}}$$

Substituting eq.6 into eq.5 to eliminate x and obtain $x = f(y)$:

$$x = f(y) = a_6 y + a_7 \qquad eq.7$$

Where:

$$a_6 = \frac{-a_{21}a_4 - a_{22}}{a_{23}} \quad a_7 = \frac{b_2 - a_{21}a_5}{a_{23}}$$

Substitute eq.6 in eq. 7 into eq. 1 to eliminate x and z and obtain a quadratic equation only in y

$$ay^2 + by + c = 0 \qquad eq.8$$

Where:

$$a = a_4^2 + 1 + a_6^2$$
$$b = 2a_4(a_5 - x_1) - 2y_1 + 2a_6(a_7 - z_1)$$
$$c = a_5(a_5 - 2x_1) + a_7(a_7 - 2z_1) + x_1^2 + y_1^2 + z_1^2 - r_1^2$$

eq.8 yields to two possible solutions for y:

$$y_{\pm} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \qquad eq.9$$

Having the two possible values of y, those solutions are substitute in eq. 6 and eq.7, once this is done. The two possible solutions are found.

(x_+, y_+, z_+) and (x_-, y_-, z_-)

Appendix B – Three spheres intersection algorithm for identical vertical centre heights

In order to solve this algorithm it is necessary to have three spheres, $(c_1, r_1), (c_2, r_2), (c_3, r_3)$. For the solution of the algorithm is necessary to know the vectors centres c_i and radius r. The equations of the three spheres are:

$$(x - x_1)^2 + (y - y_1)^2 + (z - z_n)^2 = r_1^2 \qquad eq.1$$
$$(x - x_2)^2 + (y - y_2)^2 + (z - z_n)^2 = r_2^2 \qquad eq.2$$
$$(x - x_3)^2 + (y - y_3)^2 + (z - z_n)^2 = r_3^2 \qquad eq.3$$

Due to the sphere centre height in z are the same, $z_1 = z_2 = z_3 = z_n$, yielding to:

$$x^2 - 2x_1x + x_1^2 + y^2 - 2y_1y + y_1^2 + z^2 - 2z_nz + z_n^2 = r_1^2 \quad eq.4$$
$$x^2 - 2x_2x + x_2^2 + y^2 - 2y_2y + y_2^2 + z^2 - 2z_nz + z_n^2 = r_2^2 \quad eq.5$$
$$x^2 - 2x_3x + x_3^2 + y^2 - 2y_3y + y_3^2 + z^2 - 2z_nz + z_n^2 = r_3^2 \quad eq.6$$

It is possible to simplificate eq.4, eq.5 and eq.6 yielding to:

$$2(x_3 - x_1)x + 2(y_3 - y_1)y + x_1^2 + y_1^2 - x_3^2 - y_3^2 = r_1^2 - r_3^2 \quad eq.7$$
$$2(x_3 - x_2)x + 2(y_3 - y_2)y + x_2^2 + y_2^2 - x_3^2 - y_3^2 = r_2^2 - r_3^2 \quad eq.8$$

eq.7 and eq.8 are linear equations so, they can be expressed as:

$$\begin{bmatrix} a & b \\ d & e \end{bmatrix} \begin{Bmatrix} x \\ y \end{Bmatrix} = \begin{Bmatrix} c \\ f \end{Bmatrix} \qquad eq.9$$

Where:

$$a = 2(x_3 - x_1)x$$
$$b = 2(y_3 - y_1)$$
$$c = -x_1^2 - y_1^2 + x_3^2 + y_3^2 + r_1^2 - r_3^2$$
$$d = 2(x_3 - x_2)$$
$$e = 2(y_3 - y_2)$$
$$f = -x_2^2 - y_2^2 + x_3^2 + y_3^2 + r_2^2 - r_3^2$$

The solution for the unknown x, y is:

$$x = \frac{ce - bf}{ae - bd}$$
$$y = \frac{af - cd}{ae - bd}$$

To solve the unknown value of z it is necessary to applied:

$$Az^2 + Bz + C = 0$$

Where:

$$A = 1$$
$$B = -2z_n$$
$$C = z_n^2 - r_1^2 + (x - x_1)^2 + (y - y_1)^2$$

Because the values of x, y are known the quadratic formula can be used to find the value of z.

$$z_{\pm} = \frac{-B \pm \sqrt{B^2 - 4C}}{2}$$

This value also has two possible solutions. But because of the design of the robot the negative value of z is always used.

Appendix C – Following trajectory MATLAB script

```
clc
clear all
close all

L = 0.300;      % upper legs length
l = 0.800;      % lower legs parallelogram length
Wb = 0.180;     % planar distance from {0} to near base side
Wp = 0.030;     % planar distance from {P} to near platform side
Up = 0.035;     % planar distance from {P} to a platform vertex
Sp = 0.100;     % platform equilateral triangle side

%centre of the figure
hx=0.3;
ky=0.2;

%time
t0=0;
tf=4;

%sampling distance
p_1=.05;

%frequency
w1 = 2;
w2 = 4;

sampling=(tf-t0)/p_1;

for s=1:sampling;

t=s*p_1;

%equation for a obtaining lissajous figure
xd(s) = 0.17*sin(w1*t)+hx;
yd(s) = 0.17*cos(w1*t)+ky;
zd(s) = 0.1*sin(w2*t)-0.7;

% desired position
d = [xd(s), yd(s), zd(s)];

%IPK
a = Wb - Up;
b = (Sp/2.0) - ((sqrt(3.0)/2.0) * Wb);
c = Wp - (Wb/2.0);

E1 = (2.0 * L) * (d(2) + a);
F1 = (2.0 * d(3)) * L;
G1 = d(1)^2 + d(2)^2 + d(3)^2 + a^2 + L^2 + (2.0 * d(2) * a) - l^2;
```

```
E2 = -L * ((sqrt(3.0) * (d(1) + b)) + d(2) + c);
F2 = 2.0 * d(3) * L;
G2 = d(1)^2 + d(2)^2 + d(3)^2 + b^2 + c^2 + L^2 + (2.0 * ((d(1) * b) +
(d(2) * c))) - l^2;

E3 = L * ((sqrt(3.0) * (d(1) - b)) - d(2) - c);
F3 = 2.0 * d(3) * L;
G3 = d(1)^2 + d(2)^2 + d(3)^2 + b^2 + c^2 + L^2 + (2.0 * (-(d(1) * b) +
(d(2) * c))) - l^2;

t1 = (-F1 - sqrt(F1^2 - G1^2 + E1^2)) / (G1 - E1);
t2 = (-F2 - sqrt(F2^2 - G2^2 + E2^2)) / (G2 - E2);
t3 = (-F3 - sqrt(F3^2 - G3^2 + E3^2)) / (G3 - E3);

theta1 = 2*atan(t1);
theta2 = 2*atan(t2);
theta3 = 2*atan(t3);

%FPK
x1 = 0;
y1 = -Wb - (L*(cos(theta1))) + Up;
z1 = -L * (sin(theta1));
r1 = l;

x2 = (((sqrt(3))/2)*(Wb+(L*(cos(theta2))))) - (Sp/2);
y2 = 0.5*(Wb+(L*(cos(theta2)))) - Wp;
z2 = -L*(sin(theta2));
r2 = l;

x3 = (((-sqrt(3))/2)*(Wb+(L*(cos(theta3))))) + (Sp/2);
y3 = 0.5*(Wb+(L*(cos(theta3)))) - Wp;
z3 = -L*(sin(theta3));
r3 = l;

a11 = 2*(x3-x1);
a12 = 2*(y3-y1);
a13 = 2*(z3-z1);

a21 = 2*(x3-x2);
a22 = 2*(y3-y2);
a23 = 2*(z3-z2);

b1 = r1^2-r3^2-x1^2-y1^2-z1^2+x3^2+y3^2+z3^2;
b2 = r2^2-r3^2-x2^2-y2^2-z2^2+x3^2+y3^2+z3^2;

a1 = a11/a13 - a21/a23;

a2 = a12/a13 - a22/a23;

a3 = b2/a23 - b1/a13;

a4 = -a2/a1;
```

```
a5 = -a3/a1;

a6 = ((-a21*a4)-a22)/a23;

a7 = (b2-(a21*a5))/a23;

a = a4^2 + 1 + a6^2;
b = (2*a4)*(a5-x1) - (2*y1) + (2*a6)*(a7-z1);
c = a5*(a5-(2*x1)) + a7*(a7-(2*z1)) + x1^2 + y1^2 + z1^2 - r1^2;

yp = (-b + sqrt((b^2)-(4*a*c)))/(2*a);
yn = (-b - sqrt((b^2)-(4*a*c)))/(2*a);

xp = a4*yp + a5;
xn = a4*yn + a5;

zp = a6*yp + a7;
zn = a6*yn + a7;

%end effector's position
 px1(s) = xp;
 py1(s) = yp;
 pz1(s)= zp;

end

figure(1)
plot(px1,'+r','LineWidth', 2);
grid on
hold on
plot(xd,'b','LineWidth', 1);
hold on
grid on
title('X obtained vs X desired');legend('X-ob','X-
des');xlabel('t');ylabel('X-ob, X-des');

figure(2)
plot(py1,'+r','LineWidth', 2);
grid on
hold on
plot(yd,'b','LineWidth', 1);
hold on
grid on
title('Y obtained vs Y desired');legend('Y-ob','Y-
des');xlabel('t');ylabel('Y-ob, Y-des');

figure(3)
plot(pz1,'+r','LineWidth', 2);
grid on
hold on
plot(zd,'b','LineWidth', 1);
hold on
grid on
title('Z obtained vs Z desired');legend('Z-ob','Z-
des');xlabel('t');ylabel('Z-ob, Z-des');
```

```
figure(4)
plot3(px1,py1,pz1,'+r','LineWidth', 2)
grid on
hold on
plot3(xd,yd,zd,'b','LineWidth', 1)
grid on
hold on
title('DESIRED POSITION VS. REAL POSITION');legend('X,Y,Z','X-des,Y-des,Z-
des');xlabel('X,X-des');ylabel('Y,Y-des');zlabel('Z,Z-des');
```

Appendix D – URDF File

```
<?xml version="1.0"?>
<!-- Robot Name -->
<robot name="delta_robot">
<!-- Base -->
   <link name="base_link">
      <visual>
         <geometry>
            <cylinder length="0.02" radius="0.180"/>
         </geometry>
         <origin rpy="0 0 0" xyz="0 0 0"/>
      </visual>
   </link>

<!-- Arm 1 -->
   <link name="arm_1">
      <visual>
         <geometry>
            <box size="0.300 0.02 0.02"/>
         </geometry>
         <origin rpy="0 0 0" xyz="0.150 0 0"/>
         <material name="blue">
            <color rgba="0 0 1 1"/>
         </material>
      </visual>
   </link>

   <joint name="base_to_arm_1" type="revolute">
      <axis xyz="0 1 0"/>
      <limit effort="1000.0" lower="-1.5708" upper="1.5708" velocity="0.5"/>
      <parent link="base_link"/>
      <child link="arm_1"/>
      <origin rpy="0 0 -1.5708" xyz="0 -0.180 0"/>
   </joint>

<!-- Down Part Arm 1 -->
```

```
    <link name="down_arm_1">
        <visual>
            <geometry>
                <box size="0.800 0.02 0.02"/>
            </geometry>
            <origin rpy="0 0 0" xyz="0.400 0 0"/>
            <material name="blue">
                <color rgba="0 0 0.8 1"/>
            </material>
        </visual>
    </link>



    <joint name="down_arm_1_to_arm_1" type="revolute">
        <axis xyz="0 1 0"/>
        <limit effort="1000.0" lower="-1.5708" upper="1.5708" velocity="0.5"/>
        <parent link="arm_1"/>
        <child link="down_arm_1"/>
        <origin rpy="0 1.5708 0" xyz="0.300 0 0"/>
    </joint>

<!-- End efector -->
    <link name="tool0">
        <visual>
            <geometry>
                <cylinder length="0.02" radius="0.100"/>
            </geometry>
            <origin rpy="0 0 0" xyz="0.100 0 0"/>
            <material name="blue">
                <color rgba="0 0 0.8 1"/>
            </material>
        </visual>
    </link>

    <joint name="tool0_to_down_arm_1" type="revolute">
        <axis xyz="0 1 0"/>
        <limit effort="1000.0" lower="-1.5708" upper="1.5708" velocity="0.5"/>
        <parent link="down_arm_1"/>
        <child link="tool0"/>
        <origin rpy="0 1.5708 0" xyz="0.800 0 0"/>
    </joint>

<!-- Arm 2 -->
    <link name="arm_2">
        <visual>
            <geometry>
                <box size="0.300 0.02 0.02"/>
            </geometry>
            <origin rpy="0 0 0" xyz="0.150 0 0"/>
            <material name="blue">
                <color rgba="0 0 1 1"/>
            </material>
        </visual>
    </link>

    <joint name="base_to_arm_2" type="revolute">
        <axis xyz="0 1 0"/>
        <limit effort="1000.0" lower="-1.5708" upper="1.5708" velocity="0.5"/>
        <parent link="base_link"/>
        <child link="arm_2"/>
        <origin rpy="0 0 0.5236" xyz="0.155885 0.09 0"/>
    </joint>

<!-- Down Part Arm 2 -->
    <link name="down_arm_2">
        <visual>
            <geometry>
                <box size="0.800 0.02 0.02"/>
            </geometry>
            <origin rpy="0 0 0" xyz="0.400 0 0"/>
            <material name="blue">
                <color rgba="0 0 0.8 1"/>
            </material>
        </visual>
    </link>

    <joint name="down_arm_2_to_arm_2" type="revolute">
        <axis xyz="0 1 0"/>
        <limit effort="1000.0" lower="-1.5708" upper="1.5708" velocity="0.5"/>
        <parent link="arm_2"/>
        <child link="down_arm_2"/>
        <origin rpy="0 1.5708 0" xyz="0.300 0 0"/>
    </joint>

<!-- Arm 3 -->

    <link name="arm_3">
        <visual>
            <geometry>
                <box size="0.300 0.02 0.02"/>
            </geometry>
            <origin rpy="0 0 0" xyz="0.150 0 0"/>
            <material name="blue">
                <color rgba="0 0 1 1"/>
            </material>
        </visual>
    </link>

    <joint name="base_to_arm_3" type="revolute">
        <axis xyz="0 1 0"/>
        <limit effort="1000.0" lower="-1.5708" upper="1.5708" velocity="0.5"/>
        <parent link="base_link"/>
        <child link="arm_3"/>
        <origin rpy="0 0 2.618" xyz="-0.155885 0.09 0"/>
    </joint>

<!-- Down Part Arm 3 -->
    <link name="down_arm_3">
        <visual>
            <geometry>
                <box size="0.800 0.02 0.02"/>
            </geometry>
            <origin rpy="0 0 0" xyz="0.400 0 0"/>
            <material name="blue">
                <color rgba="0 0 0.8 1"/>
            </material>
        </visual>
    </link>

    <joint name="down_arm_3_to_arm_3" type="revolute">
        <axis xyz="0 1 0"/>
        <limit effort="1000.0" lower="-1.5708" upper="1.5708" velocity="0.5"/>
        <parent link="arm_3"/>
        <child link="down_arm_3"/>
        <origin rpy="0 1.5708 0" xyz="0.300 0 0"/>
    </joint>

</robot>
```

Appendix E – ROS node

```
#!/usr/bin/env python
#-------- Libraries --------#
import math
import numpy as np
#---------------------------#
#-------- ROS Libraries --------#
import rospy
from sensor_msgs.msg import JointState
from std_msgs.msg import Header
import geometry_msgs.msg
import tf
#-------------------------------#
#-------- Node initialization --------#
rospy.init_node('delta_robot') # Node initialization in ROS
broadcaster = tf.TransformBroadcaster()
odom_trans = geometry_msgs.msg.TransformStamped()
joint_state = JointState()
odom_trans.header.frame_id = 'odom'
odom_trans.child_frame_id = 'base_link'
joint_state.header = Header()
joint_state.name = ['base_to_arm_1', 'base_to_arm_2', 'base_to_arm_3',
'down_arm_1_to_arm_1', 'down_arm_2_to_arm_2',
'down_arm_3_to_arm_3','tool0_to_down_arm_1']
#-------------------------------------#
#-------- Variables --------#
L = 0.300 # Upper legs length
l = 0.800 # Lower legs length
wb = 0.180 # Planar distance from {0} to near base side
```

```
wp = 0.030 # Planar distance from {p} to near platform side
up = 0.035 # Planar distance from {p} to a platform vertex
sp = 0.100 # Platform equilateral triangle side
E = np.zeros(3) #Variable for calculate angle
F = np.zeros(3) #Variable for calculate angle
G = np.zeros(3) #Variable for calculate angle
t1 = np.zeros(3) #Variable for calculate angle
t2 = np.zeros(3) #Variable for calculate angle
angle_upper_arm_rad_1 = np.zeros(3) #First solution angle of motors in rad
angle_upper_arm_rad_2 = np.zeros(3) #Second solution angle of motors in rad
angle_upper_arm_deg_1 = np.zeros(3) #First solution angle of motors in degrees
angle_upper_arm_deg_2 = np.zeros(3) #Second solution angle of motors in degrees
final_angle_lower_arm1 = np.zeros(1)
final_angle_lower_arm2 = np.zeros(1)
final_angle_lower_arm3 = np.zeros(1)
end_effector = np.zeros(1)
k=0 #Flag for changing coordinates
#---------------------------#
#-------- Function block for calculating angles --------#
def calculate_delta_robot_angle():
    #-------- Calculate a, b, c --------#
    a = wb - up
    b = (sp/2.0) - ((math.sqrt(3.0)/2.0) * wb)
    c = wp - (wb/2.0)
    #-------- Calculate E[0], F[0], G[0] --------#
    E[0] = (2.0 * L) * (y + a)
    F[0] = (2.0 * z) * L
    G[0] = math.pow(x, 2) + math.pow(y, 2) + math.pow(z, 2) + math.pow(a, 2) + math.pow(L,
2) + (2.0 * y * a) - math.pow(l, 2)
    #-------- Calculate E[1], F[1], G[1] --------#
    E[1] = -L * ((math.sqrt(3.0) * (x + b)) + y + c)
    F[1] = 2.0 * z * L
    G[1] = math.pow(x, 2) + math.pow(y, 2) + math.pow(z, 2) + math.pow(b, 2) + math.pow(c,
2) + math.pow(L, 2) + (2.0 * ((x * b)+ (y * c))) - math.pow(l, 2)
    #-------- Calculate E[2], F[2], G[2] --------#
    E[2] = L * ((math.sqrt(3.0) * (x - b)) - y - c)
    F[2] = 2.0 * z * L
    G[2] = math.pow(x, 2) + math.pow(y, 2) + math.pow(z, 2) + math.pow(b, 2) + math.pow(c,
2) + math.pow(L, 2) + (2.0 * (-(x * b) + (y * c))) - math.pow(l, 2)
    #-------- Calculate angle in rad and degrees --------#
    for i in range(0, 3):
        t1[i] = (-F[i] + math.sqrt(math.pow(F[i],2) - math.pow(G[i],2) + math.pow(E[i],2))) /
(G[i] - E[i])
        t2[i] = (-F[i] - math.sqrt(math.pow(F[i],2) - math.pow(G[i],2) + math.pow(E[i],2))) /
(G[i] - E[i])
        angle_upper_arm_rad_1[i] = 2 * math.atan(t1[i])
        angle_upper_arm_rad_2[i] = 2 * math.atan(t2[i]) #Usada
        angle_upper_arm_deg_1[i] = math.degrees(angle_upper_arm_rad_1[i])
        angle_upper_arm_deg_2[i] = math.degrees(angle_upper_arm_rad_2[i])
#---------------------------------------------------------------#
def calculate_angle_lower_arms():
#####Calculo de angulos#####
    p0 = np.array([x, y, z])
    p1 = np.array([0, -up, 0])
    p2 = np.array([sp/2, wp, 0])
    p3 = np.array([-sp/2, wp, 0])
    a1 = np.array([0, -wb-L*math.cos(angle_upper_arm_rad_2[0]), -
L*math.sin(angle_upper_arm_rad_2[0])])
    a2 = np.array([((math.sqrt(3))/2)*(wb+(L*math.cos(angle_upper_arm_rad_2[1]))),
0.5*(wb+(L*math.cos(angle_upper_arm_rad_2[1]))), -
L*math.sin(angle_upper_arm_rad_2[1])])
```

```
    a3 = np.array([-((math.sqrt(3))/2)*(wb+(L*math.cos(angle_upper_arm_rad_2[2]))),
0.5*(wb+(L*math.cos(angle_upper_arm_rad_2[2]))), -
L*math.sin(angle_upper_arm_rad_2[2])])
    pf1 = p0 + p2
    pf2 = p0 + p3
    pf3 = p0 + p1
    #"transpuesta" para obtener cuadrados
    d1_square = ((pf1 - a1)*(pf1 - a1)).sum()
    d2_square = ((pf2 - a2)*(pf2 - a2)).sum()
    d3_square = ((pf3 - a3)*(pf3 - a3)).sum()
    D1 = math.acos((d1_square - math.pow(sp,2)-math.pow(l,2))/((-2)*sp*l))
    alpha1 = math.pi - D1
    beta1 = math.pi - angle_upper_arm_rad_2[0]
    final_angle_lower_arm1[0] = beta1 - alpha1 - (math.pi/2) + 0.15
    D2 = math.acos((d2_square - math.pow(sp,2)-math.pow(l,2))/((-2)*sp*l))
    alpha2 = math.pi - D2
    beta2 = math.pi - angle_upper_arm_rad_2[1]
    final_angle_lower_arm2[0] = beta2 - alpha2 - (math.pi/2)
    D3 = math.acos((d3_square - math.pow(sp,2)-math.pow(l,2))/((-2)*sp*l))
    alpha3 = math.pi - D3
    beta3 = math.pi - angle_upper_arm_rad_2[2]
    final_angle_lower_arm3[0] = beta3 - alpha3 - (math.pi/2) + 0.1
    end_effector[0] = (angle_upper_arm_rad_2[0] + final_angle_lower_arm1[0]) * -1
#-------- Function block for sending joint state to URDF --------#
def send_joint_state_to_urdf():
    joint_state.header.stamp = rospy.Time.now()
    joint_state.position = [upper_arm_1, upper_arm_2, upper_arm_3, lower_arm_1,
lower_arm_2,lower_arm_3, effector]
    odom_trans.header.stamp = rospy.Time.now()
    pub.publish(joint_state)
    broadcaster.sendTransform((0.0, 0.0, 0.0), (0.0, 0.0, 0.0, 1.0),
rospy.Time.now(),'base_link','odom')
#-------------------------------------------------------------#
#-------- Definition of publisher --------#
pub = rospy.Publisher('joint_states', JointState, queue_size=10)
#---------------------------------------#
#-------- Main program --------#
r = rospy.Rate(1)
#---------------------------#
#-------- Infinite loop --------#
while not rospy.is_shutdown():
#---------------------------#
#-------- Three coordinates changing --------#
    if k == 0:
        x = 0.0 ##Coordinates
        y = 0.0
        z = -0.5
    if k == 1:
        x = 0.0
        y = 0.0
        z = -0.7
    if k == 2:
        x = 0.0
        y = 0.0
        z = -1.0
#------------------------------------------#
    calculate_delta_robot_angle() #Call function calculate_delta_robot_angle
    calculate_angle_lower_arms()
    upper_arm_1 = angle_upper_arm_rad_2[0]
    upper_arm_2 = angle_upper_arm_rad_2[1]
    upper_arm_3 = angle_upper_arm_rad_2[2]
```

```
lower_arm_1 = final_angle_lower_arm1
lower_arm_2 = final_angle_lower_arm2
lower_arm_3 = final_angle_lower_arm3
effector = end_effector
send_joint_state_to_urdf() #Call function send_joint_state_to_urdf
k=k+1
if k == 3:
    k = 0
print angle_upper_arm_deg_2
print angle_upper_arm_rad_2
print final_angle_lower_arm1 #angulo lower arm
print final_angle_lower_arm2 #angulo lower arm
print final_angle_lower_arm3 #angulo lower arm
print end_effector
r.sleep()
```

Appendix F – Delta robot model in MATLAB

```
% Robot values

Sb = 0.640;     % base equilateral triangle side
Sp = 0.100;     % platform equilateral triangle side
L = 0.300;      % upper legs length
Wb = 0.180;     % planar distance from {0} to near base side
Ub = 0.360;      % planar distance from {0 } to a base vertex
Wp = 0.030;     % planar distance from {P} to near platform side
Up = 0.035;     % planar distance from {P} to a platform vertex

P = [0.0; 0.013362; -0.758460]; %Coordinates
t1 = 0.264188; %Angle upper arm 1
t2 = 0.220808; %Angle upper arm 2
t3 = 0.220808; %Angle upper arm 3

% fixed-base revolute joint points B (constant in the base frame B)
B1 = [0,-Wb,0];
B2 = [(sqrt(3)/2)*Wb, 0.5*Wb,0];
B3 = [-(sqrt(3)/2)*Wb, 0.5*Wb,0];

%edges of the base triangle
b1 = [Sb/2, -Wb, 0];
b2 = [0, Ub, 0];
b3 = [-Sb/2, -Wb, 0];

%knee points of the delta robot
A1 = [0;-Wb-L*cos(t1); -L*sin(t1)];
A2 = [sqrt(3)/2 * (Wb+L*cos(t2)); 1/2 * (Wb+L*cos(t2)); -L*sin(t2)];
A3 = [-sqrt(3)/2 * (Wb+L*cos(t3)); 1/2 * (Wb+L*cos(t3)); -L*sin(t3)];

% platform-fixed U-joint connection (constant in the base frame P)
P1 = [0; -Up; 0];
P2 = [Sp/2; Wp; 0];
P3 = [-Sp/2; Wp; 0];

% platform-fixed U-joint connection (when P is not located in {0 0 0})
Pf1 = P+P1;
Pf2 = P+P2;
Pf3 = P+P3;

% linspace--> to create the points to plot (lines)

%base triangle

bt1(:,1) = linspace(b1(1),b2(1));
bt1(:,2) = linspace(b1(2),b2(2));   % triangle side 1 components
bt1(:,3) = linspace(b1(3),b2(3));
```

```
bt2(:,1) = linspace(b2(1),b3(1));
bt2(:,2) = linspace(b2(2),b3(2));   % triangle side 2
bt2(:,3) = linspace(b2(3),b3(3));

bt3(:,1) = linspace(b3(1),b1(1));
bt3(:,2) = linspace(b3(2),b1(2));   % triangle side 3
bt3(:,3) = linspace(b3(3),b1(3));

plot3(bt1(:,1), bt1(:,2), bt1(:,3),'k');
hold on
plot3(bt2(:,1), bt2(:,2), bt2(:,3),'k');
hold on
plot3(bt3(:,1), bt3(:,2), bt3(:,3),'k');

% upper arms

ua1(:,1) = linspace(B1(1),A1(1));
ua1(:,2) = linspace(B1(2),A1(2));   % upper arm 1 components
ua1(:,3) = linspace(B1(3),A1(3));

ua2(:,1) = linspace(B2(1),A2(1));
ua2(:,2) = linspace(B2(2),A2(2));   % upper arm 2
ua2(:,3) = linspace(B2(3),A2(3));

ua3(:,1) = linspace(B3(1),A3(1));
ua3(:,2) = linspace(B3(2),A3(2));   % upper arm 3
ua3(:,3) = linspace(B3(3),A3(3));

plot3(ua1(:,1), ua1(:,2), ua1(:,3),'r');
hold on
plot3(ua2(:,1), ua2(:,2), ua2(:,3),'g');
hold on
plot3(ua3(:,1), ua3(:,2), ua3(:,3),'b');

% lower arms

la1(:,1) = linspace(A1(1),Pf1(1));
la1(:,2) = linspace(A1(2),Pf1(2));  % lower arm 1 components
la1(:,3) = linspace(A1(3),Pf1(3));

la2(:,1) = linspace(A2(1),Pf2(1));
la2(:,2) = linspace(A2(2),Pf2(2));  %lower arm 2
la2(:,3) = linspace(A2(3),Pf2(3));

la3(:,1) = linspace(A3(1),Pf3(1));
la3(:,2) = linspace(A3(2),Pf3(2));  %lower arm 3
la3(:,3) = linspace(A3(3),Pf3(3));

plot3(la1(:,1), la1(:,2), la1(:,3),'k');
hold on
plot3(la2(:,1), la2(:,2), la2(:,3),'k');
hold on
plot3(la3(:,1), la3(:,2), la3(:,3),'k');
```

```
% end-effector base

eet1(:,1) = linspace(Pf1(1),Pf2(1));
eet1(:,2) = linspace(Pf1(2),Pf2(2));
eet1(:,3) = linspace(Pf1(3),Pf2(3));

eet2(:,1) = linspace(Pf2(1),Pf3(1));
eet2(:,2) = linspace(Pf2(2),Pf3(2));  %end effector
eet2(:,3) = linspace(Pf2(3),Pf3(3));

eet3(:,1) = linspace(Pf3(1),Pf1(1));
eet3(:,2) = linspace(Pf3(2),Pf1(2));
eet3(:,3) = linspace(Pf3(3),Pf1(3));

plot3(eet1(:,1), eet1(:,2), eet1(:,3),'k');
hold on
plot3(eet2(:,1), eet2(:,2), eet2(:,3),'k');
hold on
plot3(eet3(:,1), eet3(:,2), eet3(:,3),'k');

hold on
grid on
title('Delta Robot Position');
xlabel('X');
ylabel('Y');
zlabel('Z');
plot3(P(1),P(2),P(3),'r*')
```