Dobot Magician Project

Daniel Tobón, *2126550, IEE*
Universidad Autónoma de Occidente

***Abstract*—In the present report of the Dobot Magician project is realized, its history, its applications, its design in Solidworks software, which will be implemented in the Ubuntu operating system, creating an xacro file that will be the code that contains the information of the robot in terms of its structure, using ROS software (Robot Operating System) and be displayed on the Rviz platform, Gazebo simulator and Moveit.**

***Index Terms*—**Dobot**,** ROS**,** Xacro**.**

I. INTRODUCTION

The robotics community has made impressive progress in recent years. Reliable and inexpensive robot hardware base in mobile robots, to quadrotor helicopters, to humanoids is more widely available than ever before.

The community has also developed algorithms that help those robots run with increasing levels of autonomy. One of these algorithms is ROS.

The official description of ROS is:

ROS[1] is an open-source, meta-operating system for your robot. It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers.

This description is acurrate and correctly emphasizes that ROS does not replace, but instead works alongside a traditional operating system.

Many modern robot systems rely on software that spans many different processes and runs across several different computers. For Example:

- Some robots carry multiple computers, each of wich controls a subset of the robot's sensors or actuators.
- Even within a single computer, it's often a good idea to divide the robot's software into small, and stand-alone parts that cooperate to achieve the overall goal.
- Human users often send commands to a robot from a laptop, a desktop computer, or mobile device. We can think of this human interface as an extension of the robot's software.

One of the reasons that software development for robots is often more challenging than other kinds of development is that testing can be time consuming and error-prone. Physical robots may not always be available to work with, and when they are, the process is sometimes slow and finicky.

Working with ROS provides two effective workarounds to this problem:

1) Well designed ROS systems separate the low level direct control of the hardware and high level processing and decision making into separate programs. Because of this separation, we can temporarily replace those low level programs with a simulator, to test the behavior of the high level part of the system.

2) ROS also provides a simple way to record and play back sensor data and other kinds of messages. This facility means that we can obtain more leverage from the time we do spend operating a physical robot.

For this project, we are going to simulate de Dobot Magician Robot in Gazebo with Moveit through ROS. Dobot Magician Robot is development for Shenzhen Yueliang Technology [1]; is a company dedicated to development robot arm solution in China. Dobot is the first generation of robot arm debuted in 2015 in the worldwide.

The Dobot Magician manipulator has 4 degrees of freedom and the final effector has 2; That is, in 3D modeling of the robot it is necessary to generate 6 links with their respective dimensions in the archive urdf that describes the robot. What is intended is to model the robot using ROS to simulate the behavior of the same in a physical environment without having to get to build the real model of the robot.

One objective of the project is to model the robot using CAD drawing software; For this case, SolidWorks 2016 will be used. After the robot has been modeled, the necessary packages will be built in ROS to be able to interact with all the tools offered by this operating system, such as Gazebo and Moveit, two essential programs in the development of this project.

Also, it is intended to visualize the model of the robot in a virtual environment and verify each of the degrees of freedom of the robot.

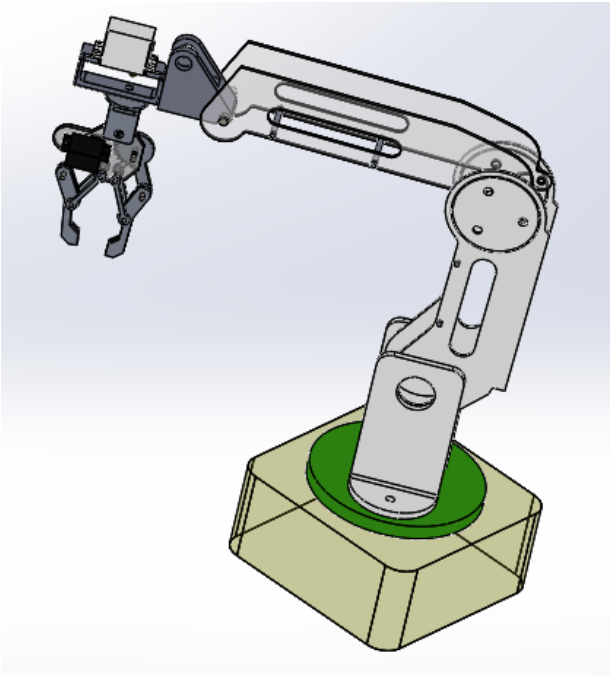
On the other hand, the physical measurements of the robot of each link for the modeling in SoliWorks were taken using a foot of king with 50 divisions in the vernier. Later the robot model is described using the Xacro format in XML and finally the step by step is detailed for loading and visualization of the model in Gazebo with Moveit

[1] http://wiki.ros.org/ROS/Introduction

II. SOLIDWORKS MODELING

With the physical measurements taken from the robot, each robot link was modeled in SolidWorks and each part was assembled. Figure 1 shows the assembly of the Dobot Magician robot model.

Figure 1. Robot model of Dobot Magician in Solidworks 2016.



In SolidWorks each link in .STL format was exported.

III. WORKING WITH ROS

The Dobot Magician manipulator has 7 degrees of freedom; That is, in 3D modeling of the robot it is necessary to generate 6 links with their respective dimensions in the xacro file that describes the robot.

Table 1 shows the most important technical specifications, including the selection of the gripper for pick and place objects as final effector.

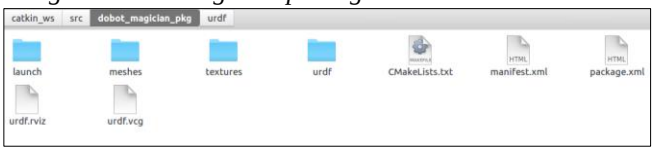
TABLE I
SPECIFICATIONS

Description	Value
Number of axes	4
Payload	500 g
Max.reach	320 mm
Joint 1 base range	-90° to +90°
Joint 2 rear arm range	0° to +85°
Joint 3 forear arm range	-10° to +95°
Joint 4 rotation servo	+90° to -90°
Joint 1 speed	320°/s
Joint 2 speed	320°/s
Joint 3 speed	320°/s
Joint 4 speed	480°/s
Net weight	8.0 kg
Base dimension	158 mm x 158 mm
Materials	Aluminum alloy 6061
Pen diameter	10 mm
Gripper range	27.5 mm
Gripper drive type	Pneumatic
Gripper force	8N

A. *Visualizing in Rviz*

To work with ROS the first step is to create the robot package in the catkin_ws folder. To do this, in the terminal of ubuntu catkin_create_pkg "*package name*" is used to create the package associated with the Dobot Magician Robot. Once the package is created, it must be built in the workspace catkin_ws; the *catkin_make* command allows you to build the Dobot Magician Robot package in catkin_ws. The image shows the package dobot_magician_pkg with the urdf, meshes and launch folders.

Image 1. Dobot magician package.



Inside of Dobot Magician package are 3 different sub-packages; urdf is the package that contains the description of the robot in a file in XML format, meshes is a folder that contains the robot geometry shell and the launch folder, it contains the ".launch" files that call the ROS Nodes. However, in order to be able to size each robot link in the xacro model, the measurements of each link were taken with a 50 division footprint on the nonio scale; The length and the thickness. The angles were measured with gauges in mm for radios and for those measures difficult to reach, a conventional meter was used. The description of the robot in the xacro file is shows in the figure 2.

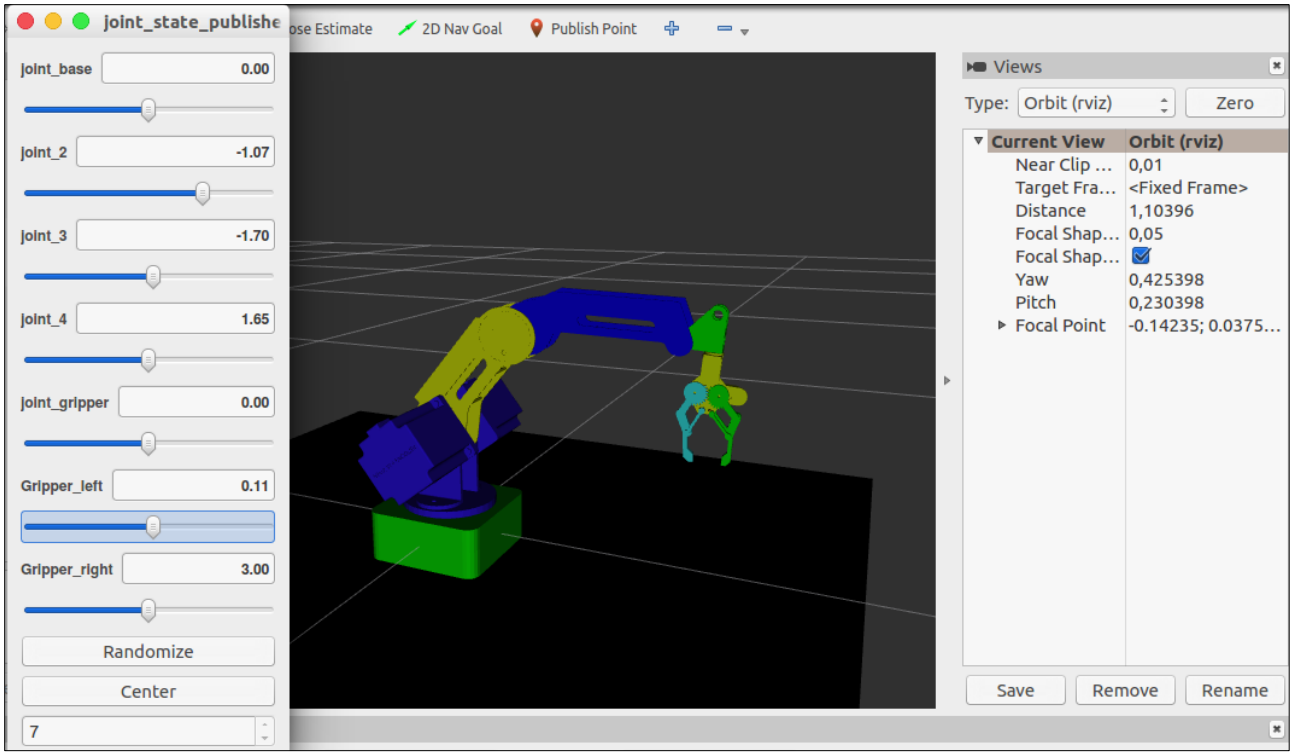
Figure 2. Dobot Magician xacro file.

```
<!--  ____________________________________________________-->
<!--  ____________________JOINT-BASE____________________-->
<!--  ____________________________________________________-->

 <joint name="joint_base" type="revolute">

    <origin xyz="0 0 0.101" rpy="0.03 0 0" />
    <parent link="base_link" />
    <child  link="link_1" />
    <axis   xyz="0 0 1" />
    <limit effort="300" velocity="0.4" lower="-3" upper="3"/>
    <dynamics damping="20" friction="1"/>

  </joint>

<!--  ____________________________________________________-->
<!--  ____________________LINK 1____________________-->
<!--  ____________________________________________________-->

  <link name="link_1">

    <visual>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh filename="package://dobot_magician_pkg/meshes/link1.dae"/>
      </geometry>
      <material name="Aluminum 6061">
        <color  rgba="0.89804 0.91765 0.92941 1" />
      </material>
    </visual>


     <inertial>
        <mass value="0.05" />
        <inertia ixx="1" ixy="0.0" ixz="0.0" iyy="1" iyz="0.0" izz="1" />
    </inertial>
```

Each link is composed of three tags; <*visual*> describes the link geometry, coordinate origin, and material. The <*inertial*> tag describes the inertia characteristics of the link. These values are specific to the geometry of each link. Because the model of the robot was also made in Solidworks, it is a tool that provides that type of information. Finally, the <*collision*> tag describes a geometry that makes up the link, generally of the same shape and dimensions, which allows to determine the possible collisions of the element with another.

One of the objective of modeling the robot in xacro format is to be able to visualize the robot in Rviz as shown in figure 3.

Figure 3. Dobot Magician Robot in Rviz.



Once created the Dobot Magician package and the xacro file with the description of the model, you can run the setup.bash in the devel folder of the catkin_ws to be able to view it in Rviz as follows. In the terminal of ubuntu:

$ source devel/setup.bash

After it has been executed, the model of the robot is visualized in Rviz with the command line roslaunch "name of the Package ""launch file ".

$ roslaunch dobot_magician_pkg view_dobot_magician_xacro.launch

Figure 4. Running the robot model in Rviz.

```
<launch>
        <arg name="model" />
        <!-- Parsing xacro and setting robot_description parameter -->
<!--    <param name="robot_description" command="$(find xacro)/xacro.py $(find mastering_ros_robot_description_pkg)/urdf/seven_dof_arm.xacro" /> -->
        <param name="robot_description" textfile="$(find dobot_magician_pkg)/urdf/dobot_magician_solidworks_model.xacro" />

        <!-- Setting gui parameter to true for display joint slider -->
        <param name="use_gui" value="true"/>
        <!-- Starting Joint state publisher node which will publish the joint values -->
        <node name="joint_state_publisher" pkg="joint_state_publisher" type="joint_state_publisher" />
        <!-- Starting robot state publish which will publish tf -->
        <node name="robot_state_publisher" pkg="robot_state_publisher" type="state_publisher" />
        <!-- Launch visualization in rviz -->
        <node name="rviz" pkg="rviz" type="rviz" args="-d $(find dobot_magician_pkg)/urdf.rviz" required="true" />
</launch>
```

After the Rviz has been executed correctly, the model of the robot can be visualized as shown in the figure 3. The figure 3, shows the model in Rviz of the Dobot Magician Robot with the node joint_state_publisher which allows to modify the articular behavior of each pair of links.

Rviz allows to visualize the model of robot, the axes of rotation, the origins of each frame of reference.

As you can see, the robot has / degrees of freedom and the end effector is effectively the gripper selected for pick and place. SolidWorks is a CAD software for 3D mechanical modeling, currently developed by SolidWorks Corp [4]. The program allows you to model parts and assemblies and extract them from both technical drawings and other information needed for production.

Alternatively, SolidWorks allows to perform static analysis to determine faults and deformations in the material used for the design of a mechanical element.

When the modeling requirements of the robot require a geometry more similar to the real model, you can take advantage of the model made in Solidworks by exporting the pieces in .STL format. STL is a native format file to the stereolithography CAD software (STL file format, Wikipedia). This format allows the visualization of each link that composes the robot with a more elaborate aspect; For this laboratory had problems in loading the STL files in Rviz because it did not allow the visualization of the geometry in the program, for this, it was necessary to convert each STL file in the DAE format that brings the modeling program by default CAD, Blender. Once the pieces are in the format that Rviz allows to display, you only need to define the geometry of each link with the <mesh> tag and indicating the address of the .DAE file that refers to that link.

A part of the code is shown below where the green line shows how to indicate the geometry of the link made in SolidWorks in .dae format and located in the meshes folder of the package dobot_magician_pkg

```
<link name="base_link">
    <visual>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
<mesh filename="package://dobot_magician_pkg/meshes/base.dae"/>
      </geometry>
      <material name="Aluminum 6061">
        <color  rgba="1 1 1 1" />
      </material>
    </visual>
    <inertial>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <mass   value="3.913" />
       <inertia ixx="0.0085988" ixy="9.7318E-20" ixz="6.3149E-19" iyy="0.014441"
   iyz="-5.4725E-20"              izz="0.0085988" />
    </inertial>
    <collision>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
<mesh filename="package://dobot_magician_pkg/meshes/base.dae"/>
      </geometry>
    </collision>
 </link>
```

B. Visualizing in Gazebo

Gazebo[2] is a 3D dynamic simulator with the ability to accurately and efficiently simulate populations of robots in complex indoor and outdoor environments. While similar to game engines, Gazebo offers physics simulation at a much higher degree of fidelity, a suite of sensors, and interfaces for both users and programs.

Typical uses of Gazebo include:

- ✓ Testing robotics algorithms
- ✓ Designing robots
- ✓ Performing regression testing with realistic scenarios
- ✓

A few key features of Gazebo include:

- ✓ Multiple physics engines
- ✓ A rich library of robot models and environments
- ✓ A wide variety of sensors
- ✓ Convenient programmatic and graphical interfaces

What is intended now is to be able to load the model of the robot in the graphical interface of gazebo and therefore, to be able to manipulate its articulations through the terminal of ubuntu. As a first step, you must create the launch file that allows loading and visualization of the xacro model in Gazebo.

To do this, a .launch file is created that has the basic structure of an openning launch in Gazebo, that is, it has the basic simulation parameters of the Gazebo world. In addition, since you want to control the movement of each robot joints in the Gazebo interface through the ubuntu terminal, you must specify the control file for each joint.

[2] http://gazebosim.org/tutorials?cat=guided_b&tut=guided_b1

The following figure shows the XML code of the launch file for loading and visualization of the model in Gazebo including the path of the control file of each joint.

Figure 5. Launch file for display in Gazebo with control joints.

```
<launch>

  <arg name="paused" default="false"/>
  <arg name="use_sim_time" default="true"/>
  <arg name="gui" default="true"/>
  <arg name="headless" default="false"/>
  <arg name="debug" default="true"/>

<rosparam file="$(find dobot_magician_pkg)/config/dobot_magician_gazebo_control.yaml" command="load"/>

<node name="controller_spawner" pkg="controller_manager" type="spawner" respawn="false"
        output="screen" ns="/dobot_magician" args="joint_state_controller
                                            joint1_position_controller
                                            joint2_position_controller
                                            joint3_position_controller
                                            joint4_position_controller
                                            joint5_position_controller
                                            joint6_position_controller
                                            joint7_position_controller"/>

  <arg name="model" default="$(find dobot_magician_pkg)/urdf/dobot_magician_solidworks_model.xacro"/>

  <include file="$(find gazebo_ros)/launch/empty_world.launch" >
  <arg name="debug" value="$(arg debug)" />
    <arg name="gui" value="$(arg gui)" />
    <arg name="paused" value="$(arg paused)"/>
    <arg name="use_sim_time" value="$(arg use_sim_time)"/>
    <arg name="headless" value="$(arg headless)"/>

  </include>

  <param name="robot_description" command="$(find xacro)/xacro --inorder $(arg model)" />

  <node name="urdf_spawner" pkg="gazebo_ros" type="spawn_model" respawn="false" output="screen" args="-urdf -model dobot_magician -param
robot_description"/>
```

As you can see in the image, the <rosparam> tag indicates where the control file of each joint is located and additionally, the node of the controller_manager package is loaded to specify the type of controller to use in Gazebo.

Figure 6 shows the XML code of the dobot_magician_control.yaml file, which indicates the type of controller to use at each joint. For this case, a PID controller was used to control the position of each joint.

Figure 6. Dobot magician gazebo control yaml file.

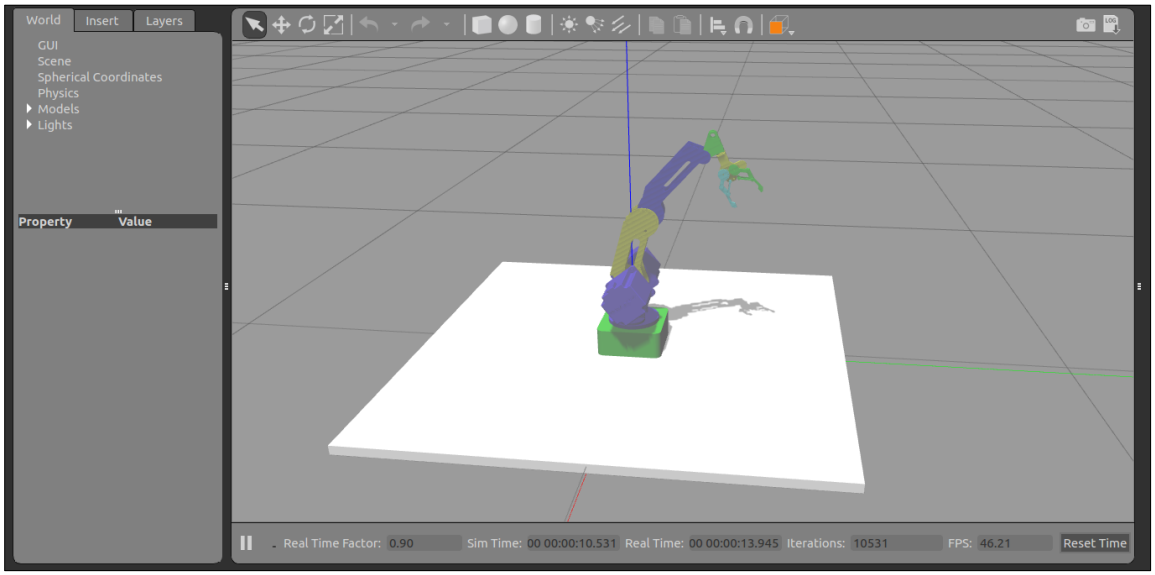
```
dobot_magician:
  # Publish all joint states -----------------------------------
  joint_state_controller:
    type: joint_state_controller/JointStateController
    publish_rate: 50

  # Position Controllers ---------------------------------------
  joint1_position_controller:
    type: position_controllers/JointPositionController
    joint: joint_base
    pid: {p: 100.0, i: 0.01, d: 10.0}
  joint2_position_controller:
    type: position_controllers/JointPositionController
    joint: joint_2
    pid: {p: 100.0, i: 0.01, d: 10.0}
  joint3_position_controller:
    type: position_controllers/JointPositionController
    joint: joint_3
    pid: {p: 100.0, i: 0.01, d: 10.0}
  joint4_position_controller:
    type: position_controllers/JointPositionController
    joint: joint_4
    pid: {p: 100.0, i: 0.01, d: 10.0}
  joint5_position_controller:
    type: position_controllers/JointPositionController
    joint: joint_gripper
    pid: {p: 100.0, i: 0.01, d: 10.0}
  joint6_position_controller:
    type: position_controllers/JointPositionController
    joint: Gripper_left
    pid: {p: 100.0, i: 0.01, d: 10.0}
  joint7_position_controller:
    type: position_controllers/JointPositionController
    joint: Gripper_right
    pid: {p: 100.0, i: 0.01, d: 10.0}
```

Now, once you have specified the driver to use you can load the launch file into the ubuntu terminal and display the model in gazebo. In the terminal of ubuntu it is written:

$ roslaunch dobot_magician_pkg dobot_magician_gazebo_world.launch

If all goes well you can see the model in Gazebo as you can see in figure 7.

Figure 7. Dobot magician in Gazebo.



Gazebo allows a more physical and real environment than Rviz since it allows to see the influence of inertia, weight and orientation of each link that composes to the model of the robot.

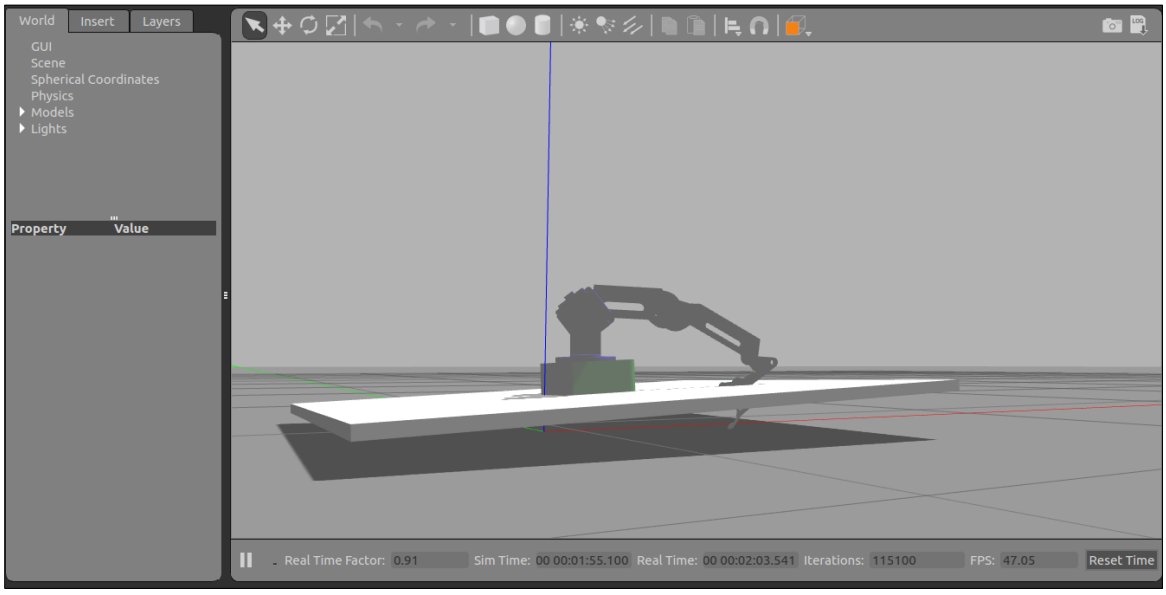
The next step is to be able to move each joint using the ubuntu terminal. To do this, you must publish a topic that indicates the type of command to send (in this case is a message), the model name of the robot, the number of the joint you want to move and the data in degrees that you want to articulate.

In a new terminal write the following:

$ rostopic pub /dobot_magician/joint2_position_controller/command std_msgs/Float64 “data: 1.0”

What is going to be done is to publish a topic that contains a message of type float with a data of 1.0 to move the joint2 of the robot Dobot Magician.

Figure 8. Moving a joint in Gazebo.



With this ends the control of each joint of the robot in gazebo. In the next section the robot model will be linked in Gazebo using Moveit, a very powerful virtual assistant that allows the planning and generation of trajectory of the end effector of the robot with respect to the base

C. Visualizing in Rviz with Moveit

MoveIt[3] is state of the art software for mobile manipulation, incorporating the latest advances in motion planning, manipulation, 3D perception, kinematics, control and navigation. It provides an easy-to-use platform for developing advanced robotics applications, evaluating new robot designs and building integrated robotics products for industrial, commercial, R&D and other domains.

MoveIt! is the most widely used open-source software.

To work with Moveit you must first install the software. For this, in the official page of Moveit is the code ubuntu package:

```
$ sudo apt-get install ros-kinetic-moveit
```

And setup your environment:

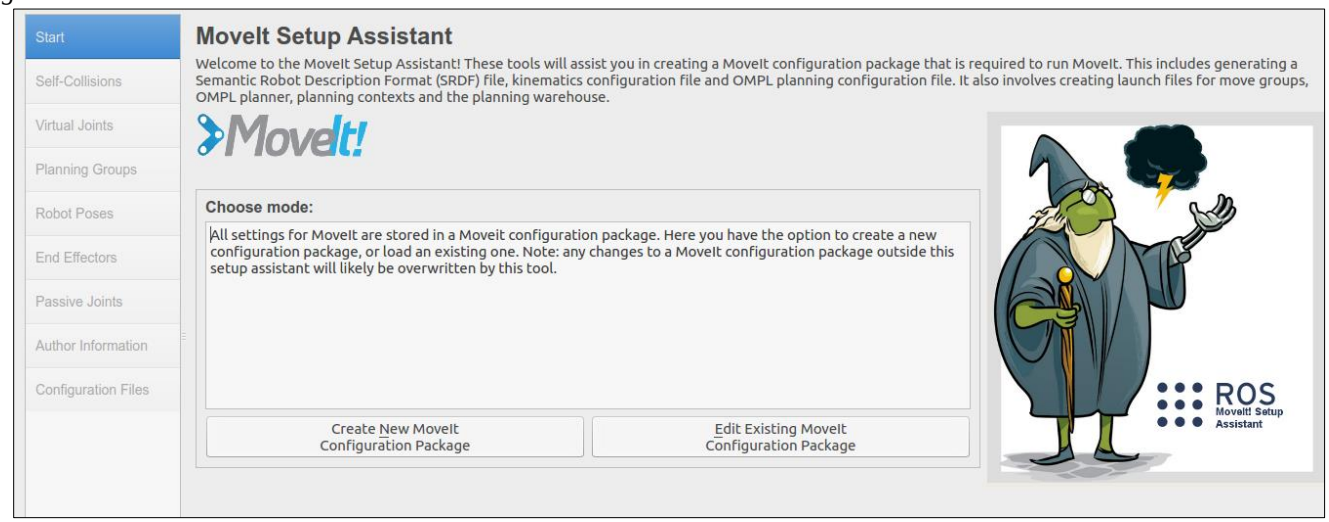
```
$ source /opt/ros/kinetic/setup.bash
```

Once installed the software runs the Moveit wizard to load the robot model. In the terminal of Ubuntu:

```
$ roslaunch moveit_setup_assistant setup_assistant.launch
```

If all goes well, you can see the assistant of Moveit as you can see in Figure 9.

Figure 9. Moveit assistant Wizard.

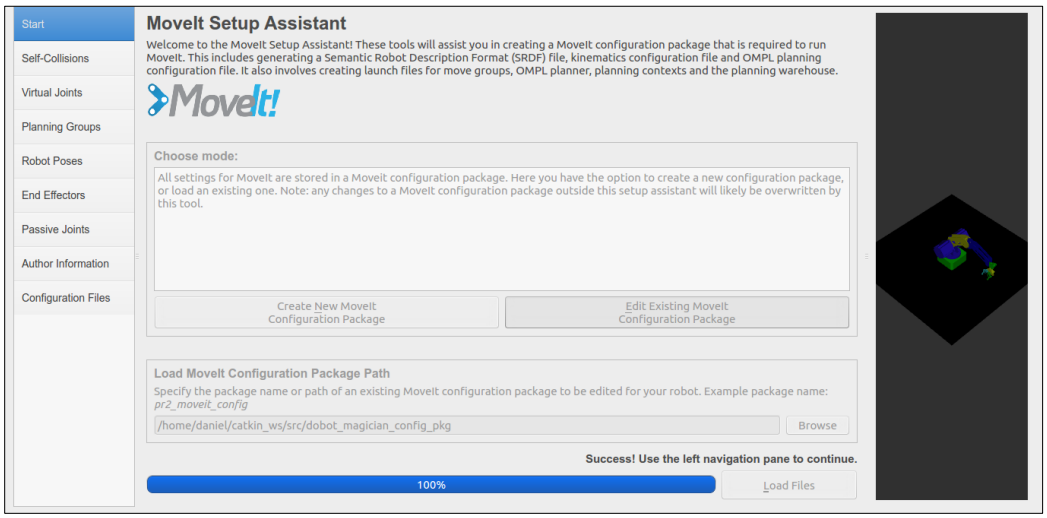


The objective is to load the description of the robot model into moveit. For this case, the xacro file must be loaded from the urd folder of the dobot_magician_pkg package containing the robot model description.

Clicking on create new Moveit configuration package, the dobot_magician_solidworks_model.xacro file is searched and selected.

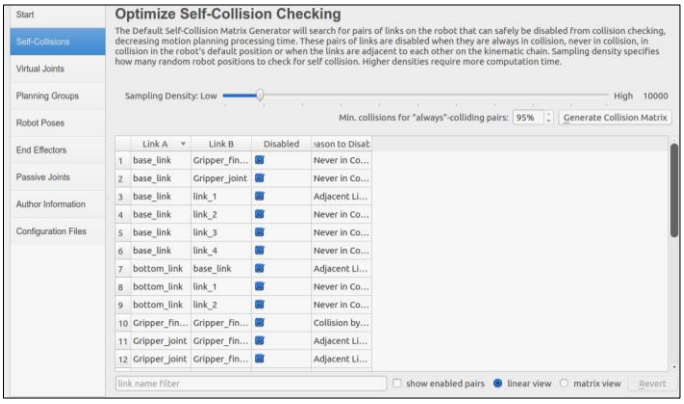
[3] http://moveit.ros.org/

Figure 10. Correct load of the model robot in Moveit.



If everything was loaded correctly, you can see the model of the robot. After the robot model has been loaded, you can continue to the next tab "self-collisions".

Figure 11. Self collisions Moveit.



Here you can generate collision matrices that allow you to identify when the element collides with another. The next tab is virtual joints; Because this robot is a manipulator it is not necessary to indicate the virtual joints since it is only necessary for mobile robots.

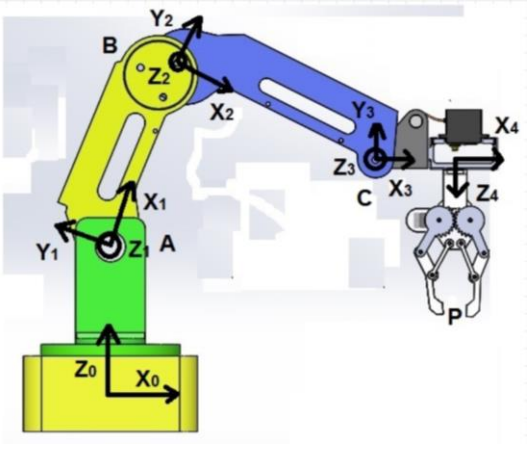
The planning groups (see figure 12) tab is the most important since it is the one that allows defining the direct kinetics of the robot and which of all the links that compose it is the final effector.

Figure 12. Planning groups.

At this point the contents of the robotics course can be evidenced because by the direct kinetics of a robot manipulator, we can know that it is the science that oversees studying how a robot can reach a desired position and orientation, given some values of articular angles and a defined geometry of the robot from a global frame of reference.

The direct kinetics of a robot starts from the homogeneous transformations of each link, from the end effector to the base and from the DH parameters of the robot.

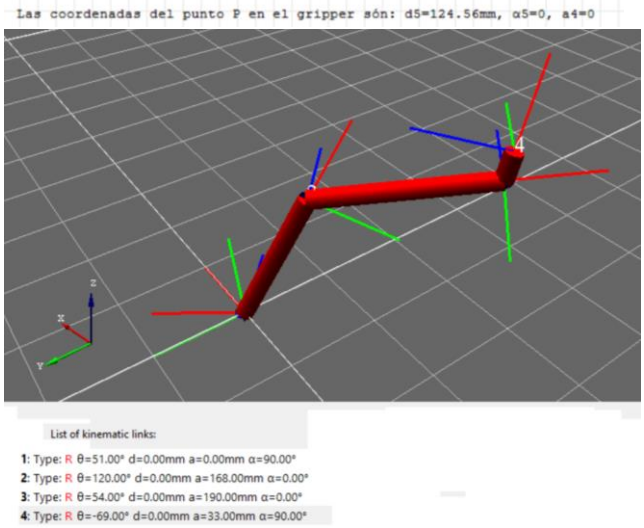
Figure 13. Coordinate system.



First the coordinate systems of each articulation of the robot were defined. From the theory of manipulating robotic arms, the DH parameters of the robot were calculated (Figure 13), and by means of specialized software to represent each DH parameter, the robot model was plotted to verify that it coincides with the robot.

Figure 14. DH parameters

i	α_{i-1}	a_{i-1}	d_i	θ_i
1	$\alpha_0 = 90°$	$a_0 = 0mm$	$d_1 = 0mm$	θ_1
2	$\alpha_1 = 0°$	$a_1 = 168.28mm$	$d_2 = 0mm$	θ_2
3	$\alpha_2 = 0°$	$a_2 = 190mm$	$d_3 = 0mm$	θ_3
4	$\alpha_3 = 90°$	$a_3 = 33.43mm$	$d_4 = 0mm$	θ_4



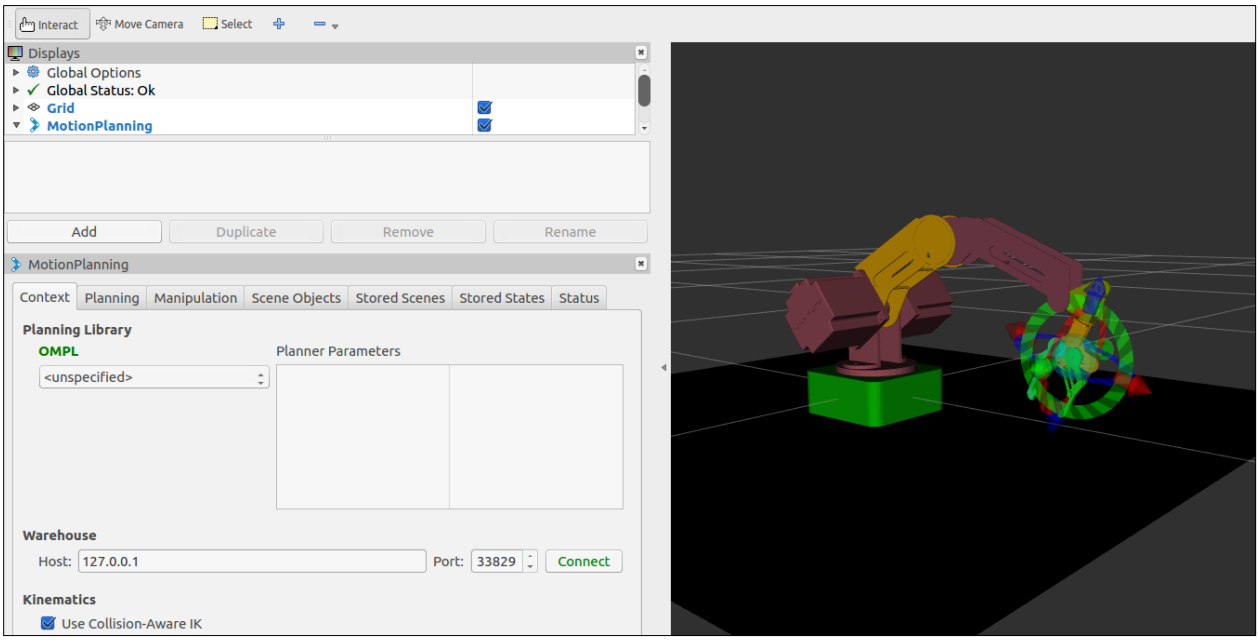
Continuing with the procedure of the configuration in Moveit defines which are the joints of the final effector and the links that compose it in the tab "End effectors" and later the package is saved in the src of catkin_ws.

To be able to visualize the model of the robot with the configurations of Moveit simply in a terminal of ubuntu:

```
$ roslaunch dobot_magician_config_pkg demo.launch
```

If all goes well, it can be seen that a window of Rviz (figure15) is opened with the model of the robot and a sphere of color in the final effector indicating the position and orientation with respect to the base.

Figure 15. Dobot magician configuration Moveit in Rviz.



In the planning tab you can select the initial state of the robot and the end, then the path is planned and therefore, the movement is executed.

D. Visualizing in Gazebo with Moveit

In the final stage of this project the planning of robot path in the graphical environment of gazebo using moveit. Therefore, you must create all the files and drivers necessary to link each interface. In the previous steps the position controller for gazebo had already been developed. What is missing is to perform the moveit position controller with gazebo. Figure 9 shows the control.yaml for gazebo with moveit.

Figure 16. Controllers yaml

```yaml
controller_manager_ns: controller_manager
controller_list:
  - name: dobot_magician/dobot_magician_joint_controller
    action_ns: follow_joint_trajectory
    type: FollowJointTrajectory
    default: true
    joints:
      - joint_base
      - joint_2
      - joint_3
      - joint_4
      - joint_gripper

  - name: dobot_magician/gripper_controller
    action_ns: follow_joint_trajectory
    type: FollowJointTrajectory
    default: true
    joints:
      - Gripper_left
      - Gripper_right
```

Indicates which are the joints that correspond to the body of the robot and which joints correspond to the final effector. Then the path controller is defined as follows:

Figure 17. Dobot magician trajectory controller.

```yaml
dobot_magician:
  dobot_magician_joint_controller:
    type: "position_controllers/JointTrajectoryController"
    joints:
      - joint_base
      - joint_2
      - joint_3
      - joint_4
      - joint_gripper

    gains:
      joint_base:   {p: 1000.0, i: 0.0, d: 0.1, i_clamp: 0.0}
      joint_2: {p: 1000.0, i: 0.0, d: 0.1, i_clamp: 0.0}
      joint_3:  {p: 1000.0, i: 0.0, d: 0.1, i_clamp: 0.0}
      joint_4:       {p: 1000.0, i: 0.0, d: 0.1, i_clamp: 0.0}
      joint_gripper:    {p: 1000.0, i: 0.0, d: 0.1, i_clamp: 0.0}

  gripper_controller:
    type: "position_controllers/JointTrajectoryController"
    joints:
      - Gripper_left
      - Gripper_right
    gains:
      Gripper_left:  {p: 50.0, d: 1.0, i: 0.01, i_clamp: 1.0}
      Gripper_right:  {p: 50.0, d: 1.0, i: 0.01, i_clamp: 1.0}

```

Also creates the joint state, which allows to determine the status of each joint of the robot and update it:

Figure 18. Dobot magician joint state controller.

```yaml
dobot_magician:
  # Publish all joint states -----------------------
  joint_state_controller:
    type: joint_state_controller/JointStateController
    publish_rate: 50

```

The main launch file that encompasses the entire gazebo interface with moveit is called dobot_magician_bringup_moveit.launch (figure 19) and this is the one that indicates all the files that relate moveit to gazebo.

Figure 19. Launch file to Gazebo with moveit.

```xml
<launch>


  <!-- Launch Gazebo  -->
  <include file="$(find dobot_magician_pkg)/launch/dobot_magician_gazebo.launch" />

  <!-- ros_control dobot magician launch file -->
  <include file="$(find dobot_magician_pkg)/launch/dobot_magician_gazebo_states.launch" />

  <!-- ros_control position control dobot magician launch file -->
  <!---include file="$(find dobot_magician_pkg)/launch/dobot_magician_gazebo_position.launch" /-->

  <!-- ros_control trajectory control dobot magician launch file -->
  <include file="$(find dobot_magician_pkg)/launch/dobot_magician_trajectory_controller.launch" />

  <!-- moveit launch file -->
  <include file="$(find dobot_magician_config_pkg)/launch/moveit_planning_execution.launch" />

<node name="joint_state_publisher" pkg="joint_state_publisher" type="joint_state_publisher" />
</launch>
```

In this file the joint_state_publisher node must be added in order to update the status of each joint. If this node is included, when executing a path in gazebo, this one is not going to move because the states of each joint are not being updated

For each control file, there must be a launch file that can execute it. Therefore, more launch files were created for each controller. Figure 20, 21 and 22 shows the launch files for each controller.

Figure 20. Launch file of gazebo position controller.

```xml
<launch>
  <!-- Load joint controller configurations from YAML file to parameter server -->
  <rosparam file="$(find dobot_magician_pkg)/config/dobot_magician_gazebo_joint_position.yaml" command="load"/>


  <!-- load the controllers -->
  <node name="controller_spawner" pkg="controller_manager" type="spawner" respawn="false"
        output="screen" ns="/dobot_magician" args="joint1_position_controller
                                                   joint2_position_controller
                                                   joint3_position_controller
                                                   joint4_position_controller
                                                   joint5_position_controller
                                                   joint6_position_controller
                                                   joint7_position_controller"/>

</launch>
```

Figure 21. Launch file of gazebo states controller.

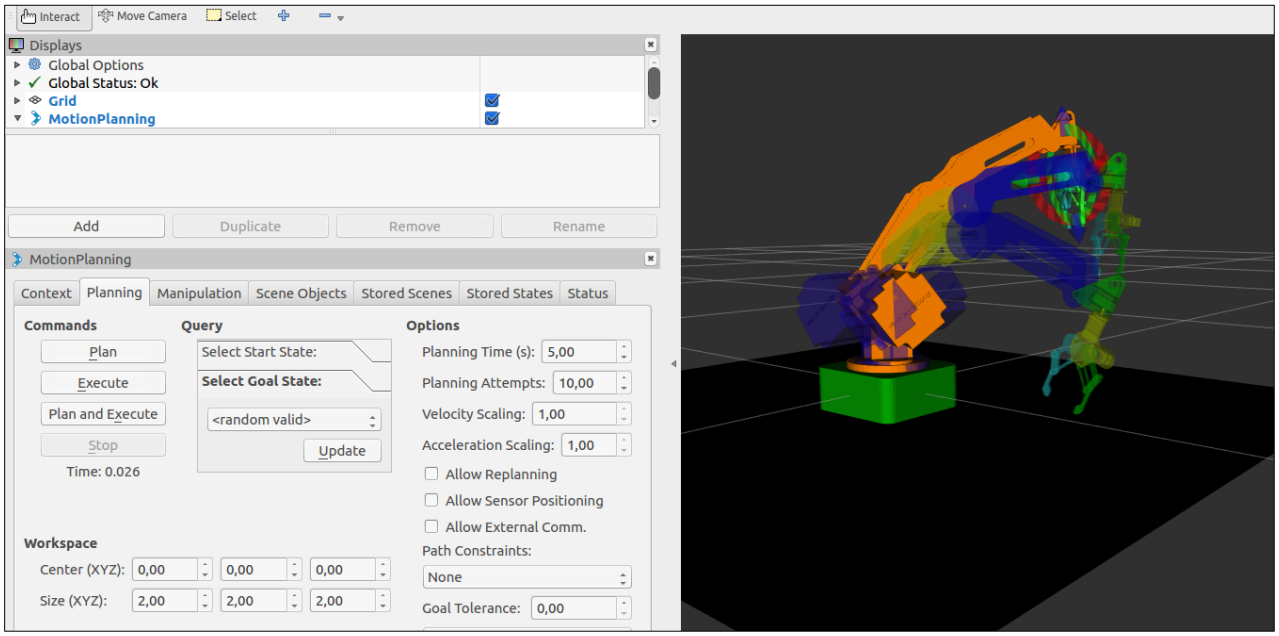
```xml
<launch>
  <!-- Load joint controller configurations from YAML file to parameter server -->
  <rosparam file="$(find dobot_magician_pkg)/config/dobot_magician_gazebo_joint_states.yaml" command="load"/>


  <node name="joint_controller_spawner" pkg="controller_manager" type="spawner" respawn="false"
                output="screen" ns="/dobot_magician" args="joint_state_controller" />

  <!-- convert joint states to TF transforms for rviz, etc -->
  <node name="robot_state_publisher" pkg="robot_state_publisher" type="robot_state_publisher"
        respawn="false" output="screen">
    <remap from="/joint_states" to="/dobot_magician/joint_states" />
  </node>

</launch>
```

Figure 22. Launch file of gazebo trajectory controller.

```xml
<launch>
        <rosparam file="$(find dobot_magician_pkg)/config/trajectory_control.yaml" command="load"/>

  <node name="dobot_magician_controller_spawner" pkg="controller_manager" type="spawner" respawn="false"
    output="screen" ns="/dobot_magician" args="dobot_magician_joint_controller       gripper_controller"/>

</launch>
```

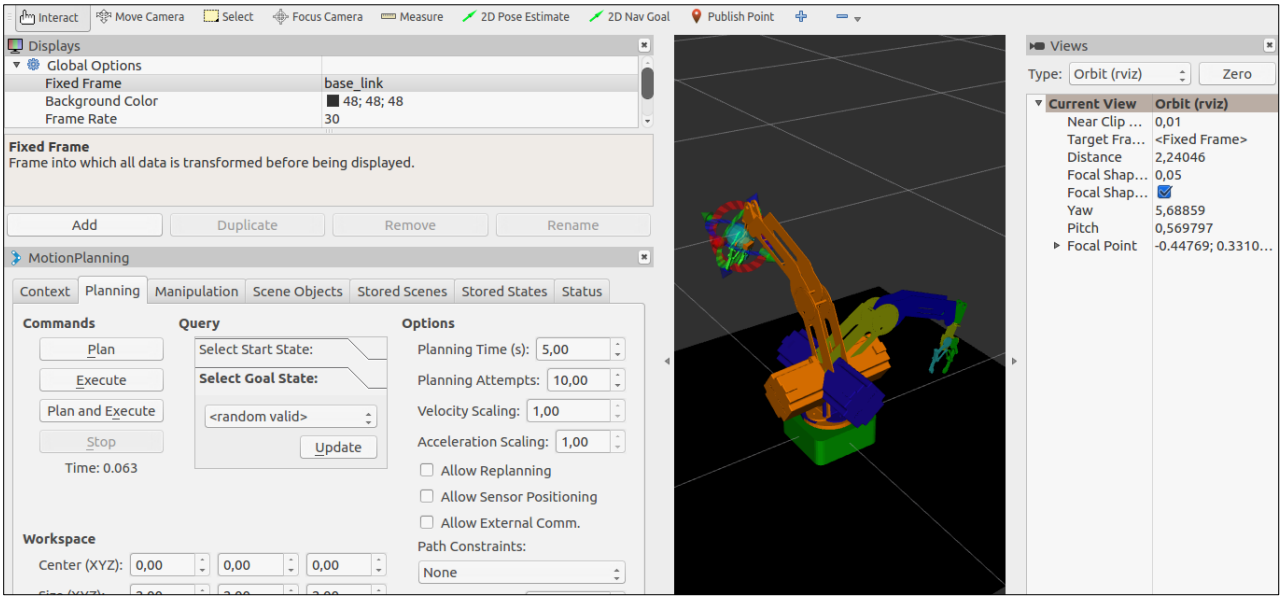
Finally, you can run the main launch file and view the Rviz interface and gazebo. In Rviz the base_link is selected as fixed frame and then a path planning is added (see figure 23).

Figure 23. Trajectory Planning.



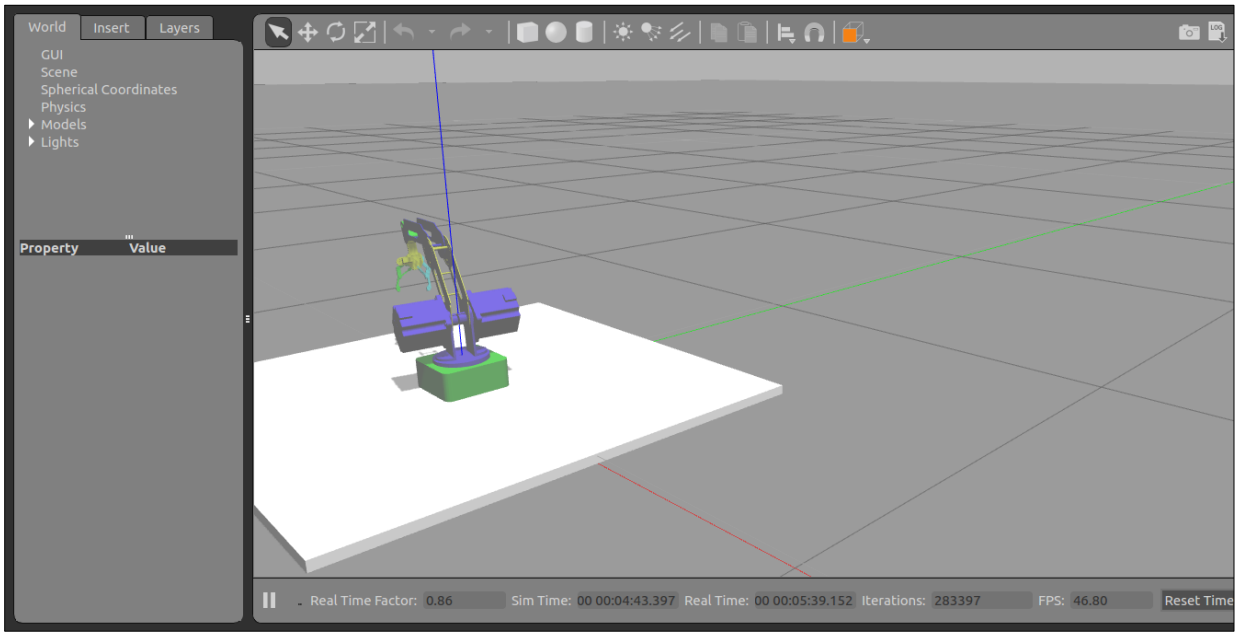
As you can effectively see the robot traces a path, plans it and then executes it.

Figure 24. Trajectory Planning.



When running the trajectory can be visualized in gazebo the movement of the robot (figure 25).

Figure 25. Trajectory Planning.



With this you can check the synchronization that has Rviz with gazebo using Moveit.

IV. Conclusion

The design and modeling of a robot using ROS is achieved by specifying and describing the geomechanically characteristics of the robot; To do this, there is a package containing the files with the robot descriptions. Depending on the application and the type of complexity that the robot has to design, you can model the robot with urfd or xacro file. The difference is that urdf files can model a simple robot that contains few links and joints; When the design requires more complexity it is appropriate to use xacro since it allows to use macro functions that allow to reduce the code of implementation, making more effective the design.

When passing the parts of the robot to the urdf you have to take into account the origins of the joints and the links. This is because they are not fixed as in the file of solidworks, for this you have to place the corresponding origin of each element, taking into account who is the father and son.

Moveit and Gazebo are two very powerful interfaces for simulating the robot model in a virtual environment with physical properties such as gravity and inertia. The advantage of being able to perform the planning of trajectory with Rviz through Moveit allows to deepen in the theory of the course of robotics complementing the subjects of the direct kinetics of the robot, the form that calculates the position and final orientation and then inversely, modifies The values of the articular angles in order to reach the desired position.

V. Anexo

As part of the annex is intended to indicate the problems that occurred during the development of the project and how they could be solved.

First, some loading errors of the model in Rviz were given since the part exported in SolidWorks was in .STL format. Rviz only recognized the .dae format so it was necessary to do a format conversion using the Blender drawing software.

As for Gazebo, most warnings that the compilation showed were due to the old syntax of ROS indigo, so it was necessary to update some code lines of some launch files to avoid these errors. Among them, the warning of deprecated xacro was solved by adding --inorder in the executing property of xacro.py

Also in the warnings of inertia in the base that does not support KDL was necessary to include a dummy link without inertia to solve these drawbacks. In the gazebo synchronization part with moveit, it was necessary to comment on a line of the main launch file since it must be defined if the position of the joints will be controlled or the trajectory will be controlled. This is why, you must have only one of the two controllers selected.

In the part of the incorporation of the transmission block in the xacro file, an old syntax error by the hardware_interface was presented. It was solved by adding the same hardware_interface / JointPosition tag to update the syntax of the same.

References

[1] Dobot Magician. (online), available in http://dobot.cc/dobot-magician/product-overview.html. (2017) Shenzhen Yuejiang Technology. Interface of Dobot Magician software. Dobot magician user manual, Pag.12

[2] Letin Joseph, Mastering ROS for Robotics Programming. Birmingham B3 2PB, UK. ISBN 978-1-78355-179-8.

[3] (2017). (online), available in https://es.wikipedia.org/wiki/SolidWorks