Working with 3D Robot Modeling in ROS

Daniel Tobón Collazos
2126550
Universidad Autónoma de Occidente
Cali Colombia

Abstract*—*The present work is the application of chapter 2 of mastering ros for robotic programming for the design and modeling of a robot using ROS. Based on the theory of robotics, it is intended to model the characteristics of three different robots; A pan and tilt robot, a robot manipulator of seven degrees of freedom and a mobile differential robot. On the other hand, it is mentioned the chosen subject as final project of course for the robotic subject of the autonomous university of the west.

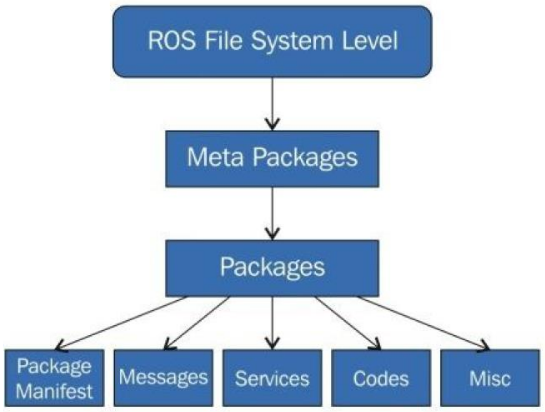
***Index Terms*—**ROS**,** Nodes**,** Urdf**.**

I. Introduction

ROS is an advanced software used for designing, building and simulating a robot model and interfacing it into real hardware; is a trending robot application development platform that provides various features such as message passing, distributed computing, code reusing, and so on. ROS is now an essential requirement for Robotic engineers; this paper introduces the topic about of chapter 2 of the Mastering ROS for Robotics Programming book.

The basic building blocks of the ROS software framework are ROS packages. Similar to an operating system, ROS files are also organized on the hard disk in a particular fashion. The following figure shows how ROS files and folder are organized on the disk.

Figure 1. Ros file system level.



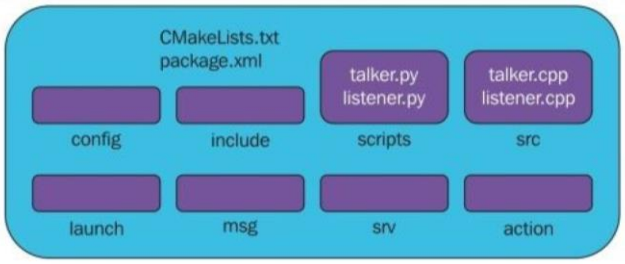
Source: ROS file system level. Chapter 1: Introduction to ROS and its packages management, Pag. 37.

The ROS packages are the most basic unit of the ROS software. It contains the ROS runtime process (nodes), libraries, configuration files, and so on, which are organized together as a single unit. The package manifest file is inside a package that contains information about the package, author, license, dependencies, compilation flags, and so on. The package.xml file inside the ROS package is the manifest file of that package.

The term meta package is used for a group of packages for a special purpose. The ROS messages are a type of information that is sent from one ROS process to the other. We can define a custom message inside the msg folder inside a package. The ROS service is a kind of request/reply interaction between processes. The reply and request data types can be defined inside the srv folder inside the package.

Most of the ROS packages are maintained using a *Version Control System* (*VCS*) such as Git, subversion (svn), mercurial (hg), and so on. A typical structure of ROS package is shown in the figure 2.

Figure 2. Structure of a typical Ros package.



Source: ROS package. Chapter 1: Introduction to ROS and its packages management, Pag. 39.

In this paper, we are going to discuss the designing of three robots. One is a pan and tilt mechanism with three links and two joints. Seven DOF (Degrees of Freedom) manipulator and the other is a differential drive robot. If we are planning to create the 3D model of the robot and simulate using ROS, is necessary learn about some ROS packages which helps in robot designing. ROS has a standard meta package for designing, and creating robot models called robot_model, which consists of a set of packages that help us to create the 3D robot model description with the exact characteristics of the real hardware.

Finally, the theme chosen for the final project of the course of Robotics is mentioned ().

II. Chapter 2: Working with 3D Robot Modeling in ROS

The first phase of robot manufacturing is its design and modeling. The main purposes of modeling robot is simulation. The robotic simulation tool can check the critical flaws in the robot design and can confirm the working of the robot before it goes to the manufacturing phase.

The virtual robot model must have all characteristics of real hardware, the shape of robot may or may not look like the actual robot, but it must be an abstract, which has all the physical characteristics of the actual robot.

Without any more preambles, let's get start!

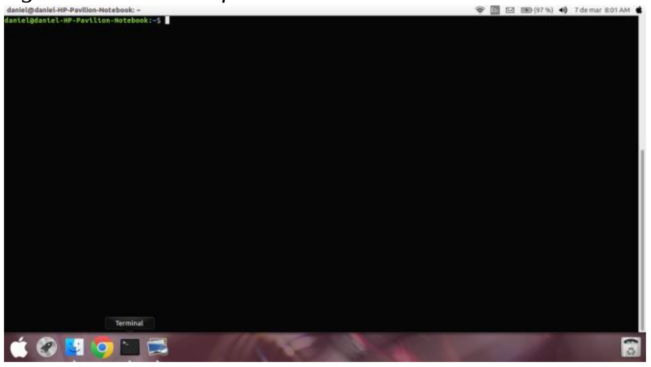
ROS has a standard meta package for designing, and creating robot model called robot_model, which consists of a set of packages called urdf, kdl_parser, robot_state_publisher, collada_urdf, and so on.

A. *Creating the ROS package for the robot description*

The first step is create the workspace for the robot model; the package catkin contains inside the urdf package. The URDF packages contains a C++ parser, which is an XML file to represent a robot model. We can define a robot model, sensors and a working environment using urdf. In summary, urdf package contains the description of the robot model and its characteristics.

For create catkin workspace package you have to open a terminal in Ubuntu pressing Ctrl + alt + t. The figure 3 shows the terminal of Ubuntu.

Figure 3. Terminal of Ubuntu.



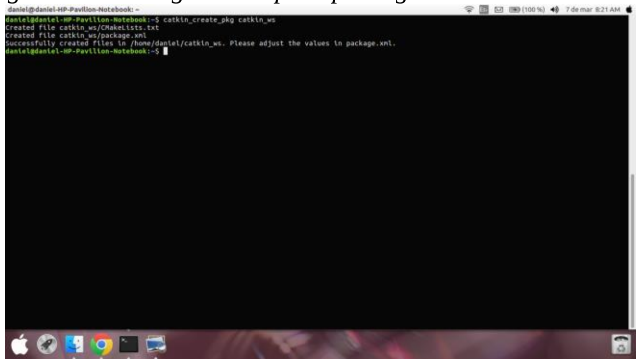
```
daniel@daniel-HP-Pavilion-Notebook:~$
```

The catkin workspace package is created using the following code line:

```
$ catkin_create_pkg catkin_ws
```

The first comand is the instruction to create a package and the second comand is the name of the package to create.

Figure 4. Creating a workspace package.



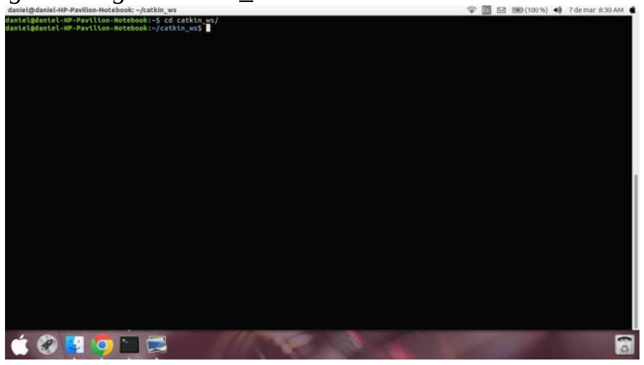
```
daniel@daniel-HP-Pavilion-Notebook:~$ catkin_create_pkg catkin_ws
Created file catkin_ws/CMakeLists.txt
Created file catkin_ws/package.xml
Successfully created files in /home/daniel/catkin_ws. Please adjust the values in package.xml.
daniel@daniel-HP-Pavilion-Notebook:~$
```

The terminal show successfully when the file is created. This package depends on urdf and xacro package, so is necessary to download de full package available on the Git repository:

```
$ git clone
https://github.com/qboticslabs/mastering_ros_robot_descriptio
n_pkg.git
```

Inside the catkin folder is necessary create another package that contains the source packages. In the terminal is written cd to sign in to the next stage:

Figure 4. Sign in catkin_ws.



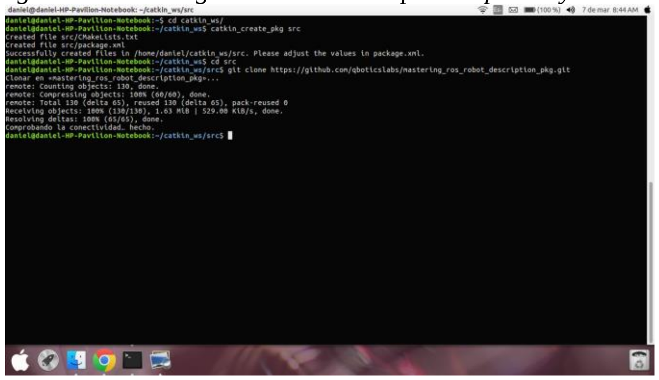
```
daniel@daniel-HP-Pavilion-Notebook:~$ cd catkin_ws/
daniel@daniel-HP-Pavilion-Notebook:~/catkin_ws$
```

In the catkin_ws folder:

```
$ catkin_create_pkg src
```

This code line creates the src folder inside of catkin_ws folder. Inside of src folder we download the full package of mastering ros robot description with the Git repository, see figure 5:

Figure 5. Mastering ROS robot description repository.

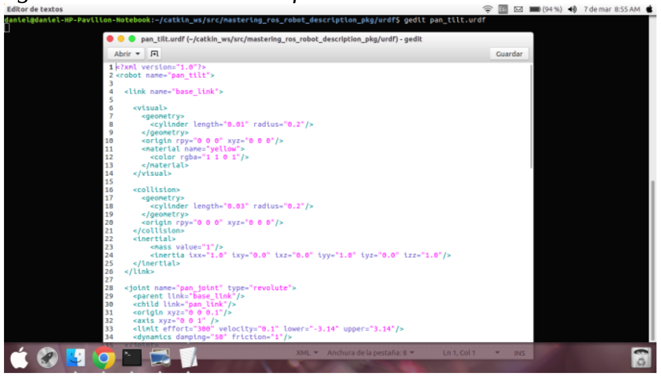


```
daniel@daniel-HP-Pavilion-Notebook:~$ cd catkin_ws/
daniel@daniel-HP-Pavilion-Notebook:~/catkin_ws$ catkin_create_pkg src
Created file src/CMakeLists.txt
Created file src/package.xml
Successfully created files in /home/daniel/catkin_ws/src. Please adjust the values in package.xml.
daniel@daniel-HP-Pavilion-Notebook:~/catkin_ws$ cd src
daniel@daniel-HP-Pavilion-Notebook:~/catkin_ws/src$ git clone https://github.com/qboticslabs/mastering_ros_robot_description_pkg.git
Clonar en «mastering_ros_robot_description_pkg»...
remote: Counting objects: 130, done.
remote: Compressing objects: 100% (60/60), done.
remote: Total 130 (delta 65), reused 130 (delta 65), pack-reused 0
Receiving objects: 100% (130/130), 1.63 MiB | 529.00 KiB/s, done.
Resolving deltas: 100% (65/65), done.
Comprobando la conectividad… hecho.
daniel@daniel-HP-Pavilion-Notebook:~/catkin_ws/src$
```

B. *Creating our first URDF model*

The first mechanism that we are going to design is a pan and tilt mechanism; inside of urdf package, the pan and tilt file describe the links and joints in a XML file. Typing in the terminal gedit pan_tilt.urdf we can see the XML file as show in the figure 6.

Figure 6. Pan and tilt XML file.



```
daniel@daniel-HP-Pavilion-Notebook:~/catkin_ws/src/mastering_ros_robot_description_pkg/urdf$ gedit pan_tilt.urdf
```

pan_tilt.urdf (~/catkin_ws/src/mastering_ros_robot_description_pkg/urdf) - gedit

```xml
<?xml version="1.0"?>
<robot name="pan_tilt">

  <link name="base_link">

    <visual>
      <geometry>
        <cylinder length="0.01" radius="0.2"/>
      </geometry>
      <origin rpy="0 0 0" xyz="0 0 0"/>
      <material name="yellow">
        <color rgba="1 1 0 1"/>
      </material>
    </visual>

    <collision>
      <geometry>
        <cylinder length="0.03" radius="0.2"/>
      </geometry>
      <origin rpy="0 0 0" xyz="0 0 0"/>
    </collision>
    <inertial>
        <mass value="1"/>
        <inertia ixx="1.0" ixy="0.0" ixz="0.0" iyy="1.0" iyz="0.0" izz="1.0"/>
    </inertial>
  </link>

  <joint name="pan_joint" type="revolute">
    <parent link="base_link"/>
    <child link="pan_link"/>
    <origin xyz="0 0 0.1"/>
    <axis xyz="0 0 1" />
    <limit effort="300" velocity="0.1" lower="-3.14" upper="3.14"/>
    <dynamics damping="50" friction="1"/>
```

The <robot> tag defines the name of the robot that we are going to create. Here, we named the robot pan_tilt. If we check the sections after the <robot> tag definition, we can see link and joint definitions of the pan and tilt mechanism.

The preceding code snippet is the base_link definition of the pan and tilt mechanism. The <visual> tag can describe the visual appearance of the link, which is shown on the robot simulation. We can define the link geometry (cylinder, box, sphere, or mesh) and the material (color and texture) of the link using this tag.

C. *Visualizing the Robot 3D model in RViz*

In the catkin_ws folder is necessary to build the packages for the workspace without adding nodes using:

```
$ catkin_make
```

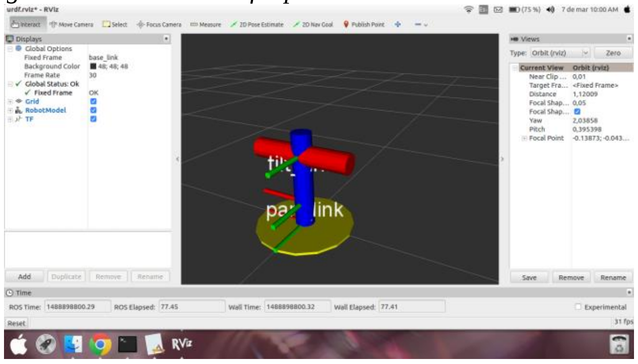
After building the empty workspace, we should set the environment of the current workspace to be visible by the ROS system. This process is called overlaying a workspace. We should add the package environment using the following command in the folder catkin_ws:

```
$ source devel/setup.bash
```

Then, executing the launch file view_pan_tilt_urdf.lauch we can see the RViz graphic:

```
$ roslaunch mastering_ros_robot_description_pkg view_pan_tilt_urdf.launch
```

Figure 7. Visualization of a pan and tilt mechanism in RViz.



urdf.rviz* - RViz

Displays
Global Options
Fixed Frame: base_link
Background Color: 48; 48; 48
Frame Rate: 30
Global Status: Ok
Fixed Frame: OK
Grid ✓
RobotModel ✓
TF ✓

Views
Type: Orbit (rviz)
Current View: Orbit (rviz)
Near Clip ...: 0,01
Target Fra...: <Fixed Frame>
Distance: 1,12009
Focal Shap...: 0,05
Focal Shap...: ✓
Yaw: 2,03858
Pitch: 0,395398
Focal Point: -0.13873; -0.043...

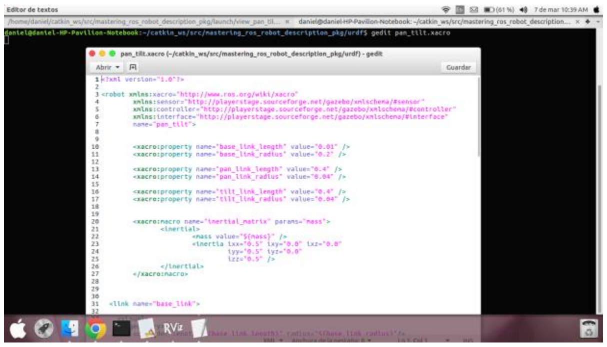
Time
ROS Time: 1488898800.29
ROS Elapsed: 77.45
Wall Time: 1488898800.32
Wall Elapsed: 77.41
31 fps

D. *Understanding robot modeling using xacro*

The flexibility of URDF reduces when we work with complex robot models. The robot modeling using xacro simplify URDF; Xacro is the cleaned up version of URDF. What it does is, it creates macros inside the robot description and reuses the macros. This can reduce the code length. Also, it can include macros from other files and make the code more readable, simpler, and modular. The xacro language support a simple programming statement in its description. There are variables, constants, mathematical expressions, conditional statements, and so on that make the description more intelligent and efficient.

The xacro file of pan and tilt mechanism is in the urdf folder with instead of .xacro.

Figure 8. pan_tilt.xacro file.



```
daniel@daniel-HP-Pavilion-Notebook:~/catkin_ws/src/mastering_ros_robot_description_pkg/urdf$ gedit pan_tilt.xacro
```

pan_tilt.xacro (~/catkin_ws/src/mastering_ros_robot_description_pkg/urdf) - gedit

```xml
<?xml version="1.0"?>

<robot xmlns:xacro="http://www.ros.org/wiki/xacro"
        xmlns:sensor="http://playerstage.sourceforge.net/gazebo/xmlschema/#sensor"
        xmlns:controller="http://playerstage.sourceforge.net/gazebo/xmlschema/#controller"
        xmlns:interface="http://playerstage.sourceforge.net/gazebo/xmlschema/#interface"
        name="pan_tilt">


        <xacro:property name="base_link_length" value="0.01" />
        <xacro:property name="base_link_radius" value="0.2" />

        <xacro:property name="pan_link_length" value="0.4" />
        <xacro:property name="pan_link_radius" value="0.04" />

        <xacro:property name="tilt_link_length" value="0.4" />
        <xacro:property name="tilt_link_radius" value="0.04" />


        <xacro:macro name="inertial_matrix" params="mass">
                <inertial>
                        <mass value="${mass}" />
                        <inertia ixx="0.5" ixy="0.0" ixz="0.0"
                                 iyy="0.5" iyz="0.0"
                                 izz="0.5" />
                </inertial>
        </xacro:macro>



  <link name="base_link">
```

Editor de textos
7 de mar 10:39 AM
/home/daniel/catkin_ws/src/mastering_ros_robot_description_pkg/launch/view_pan_til...
daniel@daniel-HP-Pavilion-Notebook: ~/catkin_ws/src/mastering_ros_robot_description...
Abrir
Guardar
RViz

We can use the value of the variable by replacing the hard coded value. The old value "0.4" is replaced with "{pan_link_length}", and "0.04" is replaced with " {pan_link_radius}". We can build mathematical expressions inside ${} using the basic operations such as + , −, ∗ , / , unary minus, and parenthesis.

E. *Creating the robot description for a seven DOF robot manipulator*

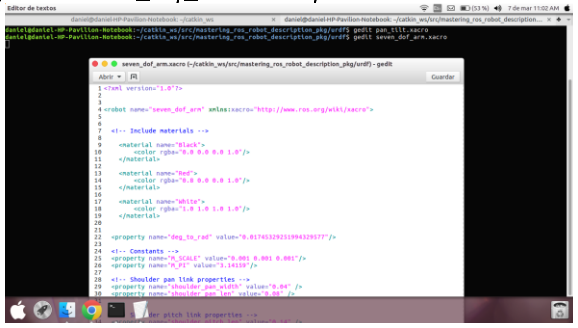
The preceding robot is described using xacro. We can navigate to the urdf folder and open the seven_dof_arm.xacro file with the comand tool gedit.

Figure 9. Seven_dof_arm.xacro file.

Now we can create some complex robots using URDF and xacro. The seven DOF robotic arm manipulator, is a serial link manipulator having multiple serial links. The seven DOF arm is kinematically redundant, which means it has more joints and DOF than required to achieve its goal position and orientation. The advantage of redundant manipulators are, we can have more joint configuration for a particular goal position and orientation. It will improve the flexibility and versatility of the robot movement and can implement effective collision free motion in a robotic workspace.

Let's start creating the seven DOF arm; in the urdf folder you can find the xacro file of the seven DOF. With the command tool gedit, we can edit the xacro file to specify features.

Figure 9. Seven_dof_arm.xacro file.



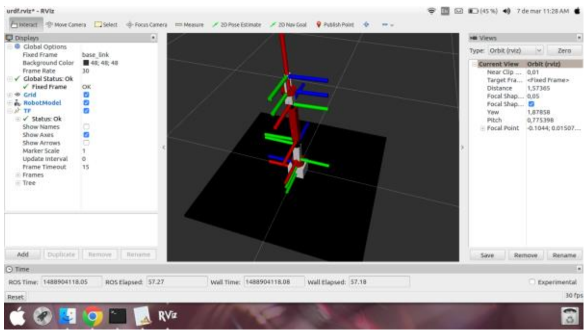
```
daniel@daniel-HP-Pavilion-Notebook:~/catkin_ws/src/mastering_ros_robot_description_pkg/urdf$ gedit pan_tilt.xacro
daniel@daniel-HP-Pavilion-Notebook:~/catkin_ws/src/mastering_ros_robot_description_pkg/urdf$ gedit seven_dof_arm.xacro
```

seven_dof_arm.xacro (~/catkin_ws/src/mastering_ros_robot_description_pkg/urdf) - gedit

```
<?xml version="1.0"?>


<robot name="seven_dof_arm" xmlns:xacro="http://www.ros.org/wiki/xacro">


  <!-- Include materials -->

    <material name="Black">
        <color rgba="0.0 0.0 0.0 1.0"/>
    </material>

    <material name="Red">
        <color rgba="0.8 0.0 0.0 1.0"/>
    </material>

    <material name="White">
        <color rgba="1.0 1.0 1.0 1.0"/>
    </material>


  <property name="deg_to_rad" value="0.01745329251994329577"/>

  <!-- Constants -->
  <property name="M_SCALE" value="0.001 0.001 0.001"/>
  <property name="M_PI" value="3.14159"/>

  <!-- Shoulder pan link properties -->
  <property name="shoulder_pan_width" value="0.04" />
  <property name="shoulder_pan_len" value="0.08" />
```

This file contains the description of the seven dof robot. With the setup.bash we can execute de Rviz program to visualize the seven dof robot. The figure 10 shows the seven dof robot in the Rviz program.

Figure 10. Seven dof robot in Rviz.



```
Fixed Frame          base_link
Background Color     48; 48; 48
Frame Rate           30
Global Status: Ok
  Fixed Frame        OK
Marker Scale         1
Update Interval      0
Frame Timeout        15

Type: Orbit (rviz)
Current View   Orbit (rviz)
Near Clip ...  0,01
Target Fra...  <Fixed Frame>
Distance       1,57365
Focal Shap...  0,05
Yaw            1,87858
Pitch          0,775398
Focal Point    -0.1044; 0.01507...

ROS Time: 1488904118.05  ROS Elapsed: 57.27  Wall Time: 1488904118.08  Wall Elapsed: 57.18
30 fps
```

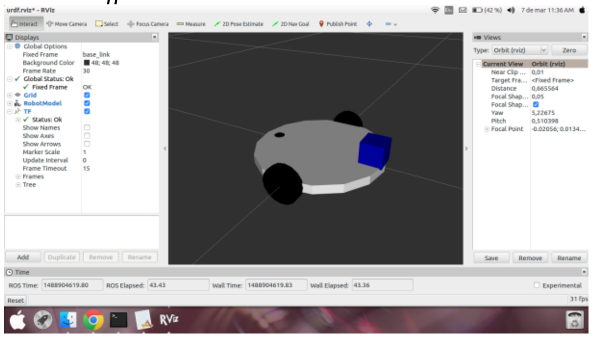
With the GUI of Rviz we can see the shape of the seven dof robot and interact with each link and joint.

F. *Creating a robot model for the differential drive mobile robot*

A differential wheeled robot will have two wheels connected on opposite sides of the robot chassis which is supported by one or two caster wheels. The wheels will control the speed of the robot by adjusting individual velocity. There are two supporting wheels called caster wheels that will support the robot and freely rotate according to the movement of the main wheels.

The procedure is the same as the seven dof robot. The urdf folder contains the XML file with the description of each features of the drive mobile robot. The figure 11 shows the differential drive mobile robot in Rviz program.

Figure 11. Differential drive mobile robot in Rviz.

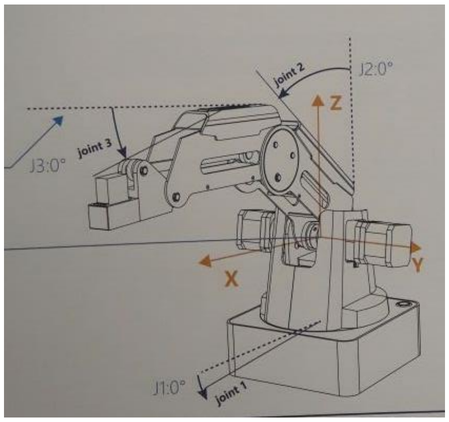


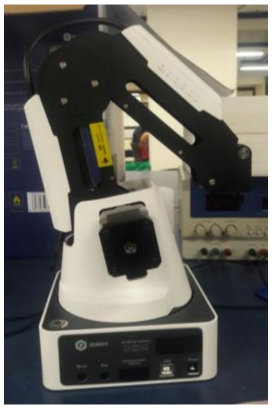
```
Fixed Frame          base_link
Background Color     48; 48; 48
Frame Rate           30
Global Status: Ok
  Fixed Frame        OK
Marker Scale         1
Update Interval      0
Frame Timeout        15

Type: Orbit (rviz)
Current View   Orbit (rviz)
Near Clip ...  0,01
Target Fra...  <Fixed Frame>
Distance       0,665564
Focal Shap...  0,05
Yaw            5,22675
Pitch          0,510398
Focal Point    -0.02056; 0.0134...

ROS Time: 1488904619.80  ROS Elapsed: 43.43  Wall Time: 1488904619.83  Wall Elapsed: 43.36
31 fps
```

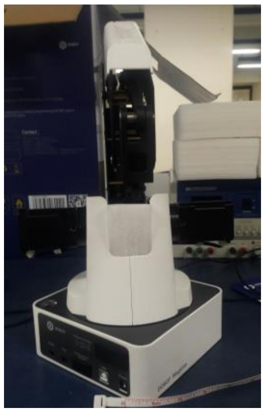
The joint state publisher can modify the orientation of the robot moving the scroll of any wheel joint.

III. Final Project:

The chosen robot for final project is the Dobot Magician Robot. (See the photography).







IV. Conclusion

The design and modeling of a robot using ROS is achieved by specifying and describing the geomechanical 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.

References

[1] Letin Joseph, "Design, build, and simulate complex robots using Robot operating system and master its out-of-the-box functionalities," Mastering ROS for Robotics Programming in *Packt open source*. Birmingham B3 2PB, UK. ISBN 978-1-78355-179-8.