****7

Multiple purpose simulation environment for robotics applications

Author: Luis Marzo Román

Supervisor: Begoña C. Arrue Ullés

Seville, 2018

Master´s Degree Final Project

Electronic, Robotic and Automatic Engineering

Department of Systems and Automatic

Higher Technical School of Engineering

University of Seville

Master´s Degree Final Project

Electronic, Robotic and Automatic Engineering

**Multiple purpose simulation environment for robotics applications.**

Author:

Luis Marzo Román

Supervisor:

Begoña C. Arrue Ullés

Department of Systems and Automatic

Higher Technical School of Engineering

University of Seville

Sevilla, 2018

*No es la ciudad,*

*son las personas.*

Acknowledgements

I would like to express my gratitude to my family for their support during these five long and endless years of university career and my friends who have accompanied me on this complicated journey.

This work would not have been possible without the great knowledge of Pablo and Begoña, whom I appreciate for all the time they have invested in me.

Por último y no menos importante, me gustaría agradecer a toda la gente que he conocido este último año, tanto a mis compañeros del master, los del laboratorio y a la familia canaria. Gracias por haber cambiado mi vida y mi forma de ser.

Que la aventura os acompañe. Thank you all.

*Luis Marzo Román*

*Seville, 2018*

Resumen

El presente trabajo consiste en el desarrollo de una herramienta HMI (Human-Machine Interface) para coordinar y controlar procesos en entornos industriales. Esta idea ha surgido a raíz del proyecto europeo TERRINET, el cual tienen como objetivo construir una red europea de infraestructuras robóticas donde investigadores con talento, tanto en el ámbito académico como industrial, tengan acceso para explorar nuevas ideas y establecer proyectos personales; ponerse en contacto e inspirarse con científicos líderes, creativos, tecnólogos, expertos y representantes industriales para compartir información y obtener el suficiente conocimiento para impulsar su investigación científica y en conclusión mejorar su potencial para la innovación tecnológica.

Concretamente el grupo de la Universidad de Sevilla que participa en el proyecto liderado por Anibal Ollero, va a poner a la disposición del proyecto robots móviles (aéreos y terrestres) y robots manipuladores. En concreto este trabajo se ha centrado en manipuladores industriales del tipo Staubli RX90 del Departamento de Ingeniería y Sistemas de la Escuela Técnica Superior de Ingeniería.

Abstract

The present work consists in the development of an HMI tool (Human-Machine Interface) to coordinate and control processes in industrial environments. This idea has arisen as a result of the European Robotics Research Infrastructures Network (TERRINET). This project aims at building a world-class network with harmonised services and complementary capabilities where talented researchers from academia and industry worldwide will have access and will be able to explore new ideas and establish personal and joint projects; to get in contact with and be inspired by leading and creative scientists, technologists, experts and industrial representatives; to share information and gain knowledge for boosting their scientific research and potential for technological innovation. The plan of TERRINET is to accomplish its vision by providing not only an organised and well orchestrated set of facilities, but also databases, tools and methodologies which will remain operational after the end of the project, so to provide a long-term service to Robotics science and industry in Europe and worldwide.

More specifically, the group from the University of Seville that participates in the project led by Anibal Ollero, will provide mobile robots (air and land) and robot manipulators. In particular, this work has focused in the Staubli RX90 robot manipulator.

Index

Acknowledgements ix

Resumen xi

Abstract xiii

Index xiv

Index of tables xvi

Index of figures xvii

1 Introduction *11*

1.1. Introduction **11**

1.2. Goals **12**

1.3. Methodology **12**

2 Types of systems *15*

2.1. Wind turbine **15**

2.1.1 Wind turbine model ***16***

2. 2. Photovoltaic cell **19**

2.2.1 Mathematical model of a photovoltaic cell **20**

2.2.2 Simulink implementation **22**

2. 3. Gas microturbine **23**

2.3.1 Gas microturbine model **24**

2. 4. PEM electrolyser **28**

2.4.1 PEM electrolyser model **29**

2.4.2 Hydrogen flow rate **31**

2.4.3 Simulink implementation **31**

2. 5. Metal hydride **33**

2.5.1 Metal hydride model **34**

2.5.2 Mass and energy balance **35**

2.5.3 Process kinetic **37**

2.5.4 Simulink implementation **38**

2. 6. Fuel cell **41**

2.6.1 Fuel cell model **43**

2.6.2 Electrochemical model **43**

2.3.1 Simulink implementation **45**

2. 7. Stationary battery **45**

2.7.1 Stationary battery model **48**

2.7.2 Simulink implementation **49**

3 Complete system and user interface *51*

3.1.Energy demand model **51**

3.2. Control system model **52**

3.3. Complete model **54**

3.4. User interface **55**

4 Analysis and conclusions *57*

4.1. Analysis **57**

4.2. Conclusions **62**

Bibliography *63*

# **Index of Tables**

# **Index of Figures**

*Figure 1. Horizontal axis wind turbines*

CHAPTER 1

**Introduccion(no te olvides de epezar los capitulos en páginas impares)**

Index

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

* 1. Introduccion. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11
  2. ROS . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
  3. Gazebo . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
  4. Rviz . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

1.5. Goals . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12

1.6. Methodology. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**1.1. Introduccion**

Habla de los sistemas unix como Linux y de ROS, que es como funciona el sistema de compilación etc. Fíjate en el TFG de Azahara. Habla del simulador gazebo y del simulador rviz

**1.2. Goals**

**Hacer un simulador para el control de los brazos robóticos Staubli que hay en el laboratorio. Comentar TERRINET**

**Di que el objetivo es hacer el simulador para que nadie se carge los rx90 y luego mandas las ordenes**

**1.3. Methodology**

The methodology is organized in the followings points:

1. Study and analysis of ROS and linux system.
2. Modelo del brazo a partir de las texturas
3. Introducción del brazo en simulación gazebo
4. Controladores del brazo
5. Estudio del puerto serie
6. Comunicación simulaccion y robot real

* **FIRST STAGE: Study and analysis.**
* **SECOND STAGE: Mathematical model.**

**1.4. Instalation**

En la siguiente sección vamos a describir todo el software necesario para el correcto funcionamiento. You can find the whole documentation in the following websites, but I will make a summary:

1. Ubuntu 16.6. [ [*https://www.ubuntu.com/download/desktop*](https://www.ubuntu.com/download/desktop%20) ]
2. ROS kinetic [ [*http://wiki.ros.org/kinetic/Installation/Ubuntu*](http://wiki.ros.org/kinetic/Installation/Ubuntu)]:

Just type step by step the commands below in the terminal in order to install it.

* 1. Setup your sources and keys:

sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb\_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'

sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --recv-key 421C365BD9FF1F717815A3895523BAEEB01FA116

* 1. Installation

sudo apt-get update

sudo apt-get install ros-kinetic-desktop-full

* 1. Environment setup

echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc

source ~/.bashrc

* 1. Dependencies

sudo apt-get install python-rosinstall python-rosinstall-generator python-wstool build-essential

1. Gazebo 7.x.x: [[*http://gazebosim.org/tutorials?tut=ros\_installing*](http://gazebosim.org/tutorials?tut=ros_installing) ]:

3.1 Setup your computer to accept software from packages.osrfoundation.org

sudo sh -c 'echo "deb http://packages.osrfoundation.org/gazebo/ubuntu-stable `lsb\_release -cs` main" > /etc/apt/sources.list.d/gazebo-stable.list'

3.2 Setup keys

wget http://packages.osrfoundation.org/gazebo.key -O - | sudo apt-key add -

3.3 Install Gazebo

sudo apt-get update

sudo apt-get install gazebo7

sudo apt-get install libgazebo7-dev

3.4 Install gazebo-ros-packages

sudo apt-get install ros-kinetic-gazebo-ros-pkgs ros-kinetic-gazebo-ros-control

1. Libserial: [ [https://sourceforge.net/projects/libserial/files/?source=navbar](https://sourceforge.net/projects/libserial/files/?source=navbar%20) ]

Inside the folder type:

./configure

make

make install

1. ROS packages

5.1 MoveIt!

sudo apt-get install ros-kinetic-moveit

source /opt/ros/kinetic/setup.bash

5.2 Ros\_control package

sudo apt-get install ros-kinetic-ros-control

sudo apt-get install ros-kinetic-ros-controllers

* 1. Joint\_state\_publisher package

sudo apt-get install ros-kinetic-robot-model

1. Rx90 package: [ [*https://github.com/luismarzo/rx90*](https://github.com/luismarzo/rx90%20) ]
   1. Creating your own catkin workspace

mkdir –p ~/ws\_rx90/src

cd ~/ws\_rx90/

catkin\_make

Sourcing the next file will overlay this workspace on top of your environment, so you have to do it every time you will open a new terminal.

source devel/setup.bash

* 1. Download package

cd ~/ws\_rx90/src/

git clone https://github.com/luismarzo/rx90

* 1. Installation

cd ~/ws\_rx90/

catkin\_make

The last step is to copy the “models” folder in *~/ws\_rx90/src/rx90/models* and copy it into your gazebo path models.

CHAPTER 2

**Montaje del brazo RX90**

Index

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

2.1. Wind turbine . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 15

2.2. Photovoltaic cell . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 19

2.3. Gas microturbine . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 23

2.4. PEM electrolyser . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 28

2.5. Metal hydride . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 33

2.6. Fuel cell . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 41

2.7. Stationary battery . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 46

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**2.1. Construcción del robot**

El modelo de las piezas individuales nos las proporciona el fabricante. Para crear un robot y moverlo con ROS usaremos un tipo de archivo llamado Unified Robot Description Format (URDF) basado en lenguaje XML.

The XML specifications that we are going to use are the followings: (You can find more specification on the ROS website)

* ***Link***: Describes the kinematic and dynamic properties of a link.

**<inertial>** *:*The inertial properties of the link.

**<origin>** *:*This is the pose of the inertial reference frame, relative to the link reference frame. The origin of the inertial reference frame needs to be at the center of gravity. The axes of the inertial reference frame do *not* need to be aligned with the principal axes of the inertia.

**xyz** *:* Represents the x,y,z offset.

**rpy** *:* Represents the fixed axis roll, pitch and yaw angles in radians.

**<mass>**: The mass of the link is represented by the value attribute of this element

**<inertia>:** The 3x3 rotational inertia matrix, represented in the inertia frame. Because the rotational inertia matrix is symmetric, only 6 above-diagonal elements of this matrix are specified here, using the attributes ixx, ixy, ixz, iyy, iyz, izz.

**<visual>** *:*The visual properties of the link. This element specifies the shape of the object (box, cylinder, etc.) for visualization purposes.

**<origin>**: The reference frame of the visual element with respect to the reference frame of the link.

**xyz** *:* Represents the x,y,z offset.

**rpy** *:* Represents the fixed axis roll, pitch and yaw angles in radians.

**<geometry>**: The shape of the visual object. This can be *one* of the following:

**<box>**

**Size:** attribute contains the three side lengths of the box. The origin of the box is in its center.

**<cylinder>**

Specify the **radius** and **length**. The origin of the cylinder is in its center.

**<sphere>**

Specify the **radius**. The origin of the sphere is in its center.

**<mesh>**

A trimesh element specified by a **filename**, and an optional **scale** that scales the mesh's axis-aligned-bounding-box. The recommended format for best texture and color support is Collada .dae files, though .stl files are also supported. The mesh file is not transferred between machines referencing the same model. It must be a local file.

**<material>** *:* The material of the visual element. It is allowed to specify a material element outside of the 'link' object, in the top level 'robot' element. From within a link element you can then reference the material by name.

**<color>**

**rgba** The color of a material specified by set of four numbers representing red/green/blue/alpha, each in the range of [0,1].

**<texture>** The texture of a material is specified by a filename

**<collision>** The collision properties of a link. Note that this can be different from the visual properties of a link, for example, simpler collision models are often used to reduce computation time.

**<origin>** The reference frame of the collision element, relative to the reference frame of the link.

**xyz** :Represents the x,y,z offset.

**rpy** *:*Represents the fixed axis roll, pitch and yaw angles in radians.

**<geometry> :**See the geometry description in the above visual element.



*Figura 1. Staubli Rx90*

* ***Transmission***: Transmissions link actuators to joints and represents their mechanical coupling

**<type>** :Specifies the transmission type.

**<joint>** :A joint the transmission is connected to. The joint is specified by its **name** attribute, and the following sub-elements:

**<hardwareInterface>** :Specifies a supported joint-space hardware interface.

**<actuator>** :An actuator the transmission is connected to. The actuator is specified by its **name** attribute, and the following sub-elements:

**<mechanicalReduction>** :Specifies a mechanical reduction at the joint/actuator transmission. This tag may not be needed for all transmissions.

**<hardwareInterface>** :Specifies a supported joint-space hardware interface.

* ***Joint:*** Describes the kinematic and dynamic properties of a joint.

**<origin>** : This is the transform from the parent link to the child link. The joint is located at the origin of the child link.

**xyz** : Represents the x,y,z offset.

**rpy** : Represents the rotation around fixed axis: first roll around x, then pitch around y and finally yaw around z. All angles are specified in radians.

**<parent>** :Parent link name with mandatory attribute:

**Link:** The name of the link that is the parent of this link in the robot tree structure.

**<child>** :Child link name with mandatory attribute:

**Link:** The name of the link that is the child link.

**<axis>** :The joint axis specified in the joint frame. This is the axis of rotation for revolute joints, the axis of translation for prismatic joints, and the surface normal for planar joints. The axis is specified in the joint frame of reference. Fixed and floating joints do not use the axis field.

**xyz** :Represents the x,y,z components of a vector. The vector should be normalized.

**<dynamics>** :An element specifying physical properties of the joint. These values are used to specify modeling properties of the joint, particularly useful for simulation.

**damping** :The physical damping value of the joint.

**friction** :The physical static friction value of the joint

**<limit>** :An element can contain the following attributes:

**lower** :An attribute specifying the lower joint limit. Omit if joint is continuous.

**upper** :An attribute specifying the upper joint limit. Omit if joint is continuous.

**effort** :An attribute for enforcing the maximum joint effort.

**velocity** :An attribute for enforcing the maximum joint velocity.

**2.2. Construcción del paquete de ROS**

El primer paso es construir el paquete de ROS. Si te has descargado el paquete entero de github tal y como se especifica en el capitulo 1 apartado 1.4 instalacion, no necesitas hacer esto. Para construirlo, usaremos las siguientes instrucciones:

cd ~/ws\_rx90/src

catkin\_create\_pkg rx90\_description std\_msgs rospy roscpp

Dentro del subpaquete llamado rx90\_description meteremos el urdf del robot, el launcher y las texturas usadas en el propio urdf. Std\_msgs rospy y roscpp son paquetes propios de ROS que permiten por ejemplo la compilación en lenguaje C++ y Python.

Para usar nuestro robot que hemos escrito en lenguaje XML, usaremos un nuevo tipo de fichero llamado “launcher”. Roslaunch is a tool for easily launching multiple ROS nodes locally and remotely via SSH, as well as setting parameters on the Parameter Server. It includes options to automatically respawn processes that have already died. Roslaunch takes in one or more XML configuration files (with the .launch extension) that specify the parameters to set and nodes to launch, as well as the machines that they should be run on.

[<**node**>](http://wiki.ros.org/roslaunch/XML/node): Launch a node.

[<**param**>](http://wiki.ros.org/roslaunch/XML/param): Set a parameter on the Parameter Server

[<**remap**>](http://wiki.ros.org/roslaunch/XML/remap): Declare a name remapping.

[<**machine**>](http://wiki.ros.org/roslaunch/XML/machine): Declare a machine to use for launching.

[<**rosparam**>](http://wiki.ros.org/roslaunch/XML/rosparam): Set ROS parameters for the launch using a rosparam file.

[<**include**>](http://wiki.ros.org/roslaunch/XML/include): Include other roslaunch files.

[<**env**>](http://wiki.ros.org/roslaunch/XML/env): Specify an environment variable for launched nodes.

[<**arg**>](http://wiki.ros.org/roslaunch/XML/arg): Declare an argument.

[<**group**>](http://wiki.ros.org/roslaunch/XML/group): Group enclosed elements sharing a namespace or remap.

Para visualizer nuestro robot y comrobar que tiene el comportamiento deseado, usaremos los siguientes nodos en nuestro launcher:

*Tf:* tf is a package that lets the user keep track of multiple coordinate frames over time. tf maintains the relationship between coordinate frames in a tree structure buffered in time, and lets the user transform points, vectors, etc between any two coordinate frames at any desired point in time.

*Robot\_state\_publisher*: This package allows you to publish the state of a robot to [tf](http://ros.org/wiki/tf). Once the state gets published, it is available to all components in the system that also use tf. The package takes the joint angles of the robot as input and publishes the 3D poses of the robot links, using a kinematic tree model of the robot. The package can both be used as a library and as a ROS node.

*Joint\_state\_publisher:* This package publishes [sensor\_msgs/JointState](http://docs.ros.org/api/sensor_msgs/html/msg/JointState.html) messages for a robot. The package reads the robot\_descriptionparameter, finds all of the non-fixed joints and publishes a JointState message with all those joints defined.

Can be used in conjunction with the [robot\_state\_publisher](http://wiki.ros.org/robot_state_publisher) node to also publish transforms for all joint states

*Rviz:* rviz is a 3D visualizer for the Robot Operating System (ROS) framework.



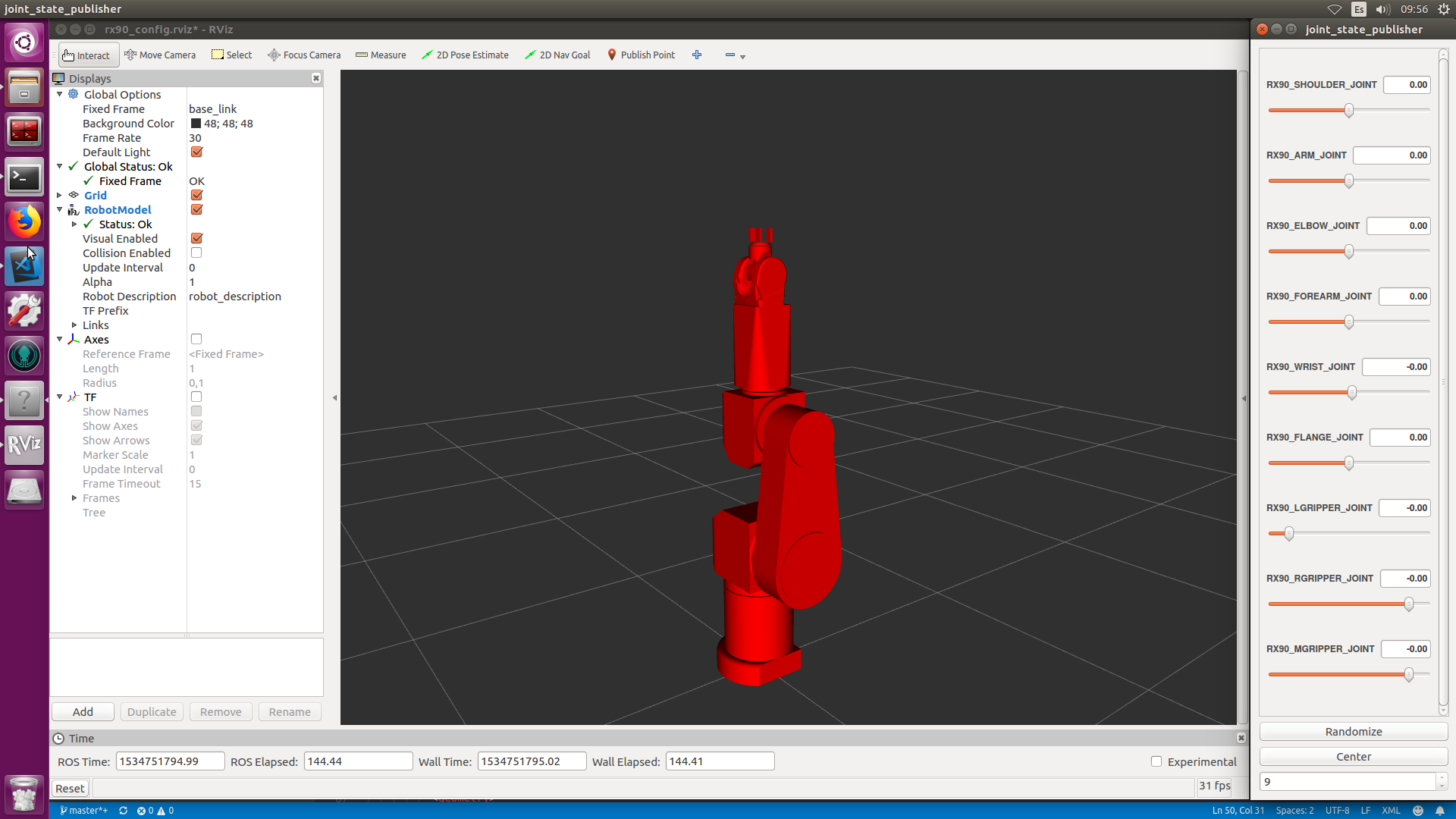
*Figura 2. Launcher file.*

**2.3. Visualización en Rviz**

Para lanzar nuestro paquete usaremos la siguiente orden:

roslaunch rx90\_description rx90.launch

Ahora podremos visualizar nuestro archivo URDF del robot y podremos mover sus articulaciones.



*Figura 3. Rviz visualitation.*

Para ver las dependencias de los nodos y los mensajes que mandan, podemos usar un nodo llamado rqt\_graph. rqt\_graph provides a GUI plugin for visualizing the ROS computation graph.  
Its components are made generic so that other packages where you want to achieve graph representation can depend upon this pkg

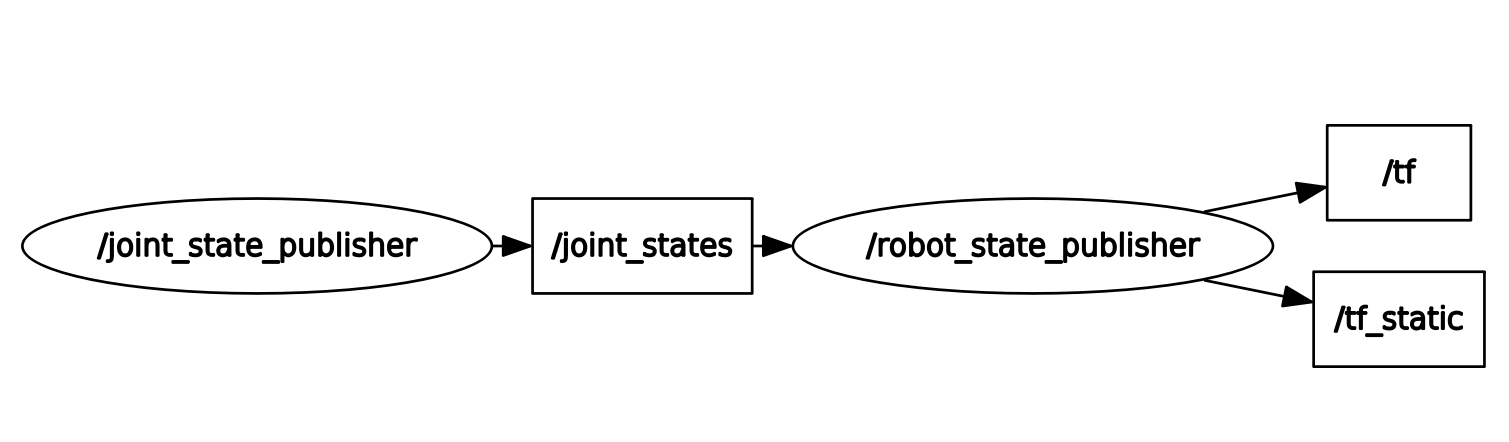


Figura 4. Rqt\_graph

Observamos que el nodo Joint\_state\_publisher manda un mensaje con las posiciones de cada articulación. Este mensaje puede ser leído desde terminal usnado rostopic echo.

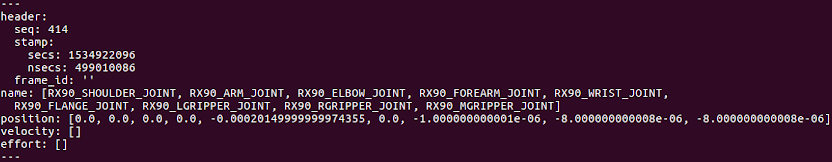


Figura 5. Joint\_state\_publisher message

Cambiando las posiciones de las articulaciones observamos el movimiento completo del robot.

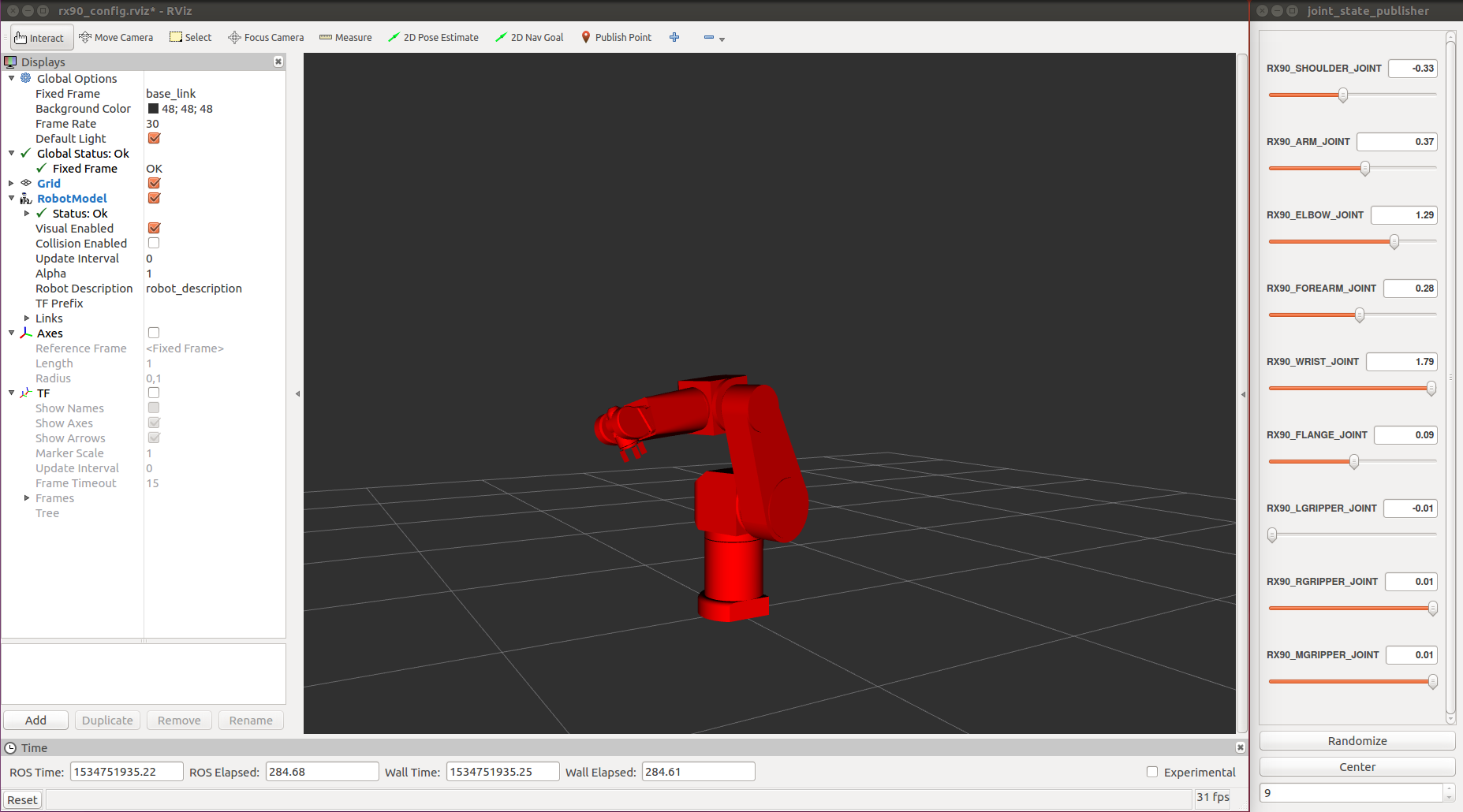


Figura 6. Rviz visualization

**2.4. Implementación en código**

CHAPTER 3

**Simulación del RX90 en Gazebo**

Index

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

3.1 Energy demand model . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 51

3.2 Control system model . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 52

3.3 Complete model . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 54

3.4 User interface . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 55

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**3.1. Creación del mundo**

La creación del archivo “.world” donde usaremos los robots en las simulaciones de gazebo usan también lenguaje XML, pero este tipo de archivo tiene diferentes tags: (solo mostraremos unos pocos, los más comunes y los que vamos a usar).

<**world**> : The world element encapsulates an entire world description including: model, scene, physics, joints and plugins

<**name**>: Unique name of the world.

<**include**> Include resources from an URI

<**uri**> URI to a resource, such a model

<**static**>: Override the static value of the included model

<**pose**>: A position (x,y,z) and orientation(roll, pitch, yaw) with respect to the specified frame.

<**gravity**> The gravity vector in m/s^2

<**atmosphere**> The atmosphere tag specifies the type and properties of the atmosphere model.

<**plugin**>: A plugin is a dynamically loaded chunk of code. It can exist as a child of world, model, and sensor.

<**scene**> Specifies the look of the environment.

<**light**> The light elemte describes a light source.

Para visualizarlo, simplemente añadimos en el archive launcher el nodo de gazebo. Sin embargo, ahora nuestros robots no podrán moverse a través del nodo joint\_state\_publisher, sino que debemos de añadirle unas controladores a todas las articulaciones de los brazos.



Figura 7. Gazebo simulation

**3.2. Controladores**

Para el movimiento de los brazos en gazebo we will setup simulated controllers to actuate the joints of our robot:

1. Add transmission elements to the URDF

We need to add some additional elements to our URDF to use controller with the robot. The <transmission> element is used to link actuators to joints.

For the purposes of gazebo\_ros\_control in its current implementation, the only important information in these transmission tags are:

* <joint name=""> the name must correspond to a joint else where in your URDF
* <type> : the type of transmission
* <hardwareInterface> - within the <actuator> and <joint> tags, this tells the gazebo\_ros\_control plugin what hardware interface to load (position, velocity or effort interfaces).

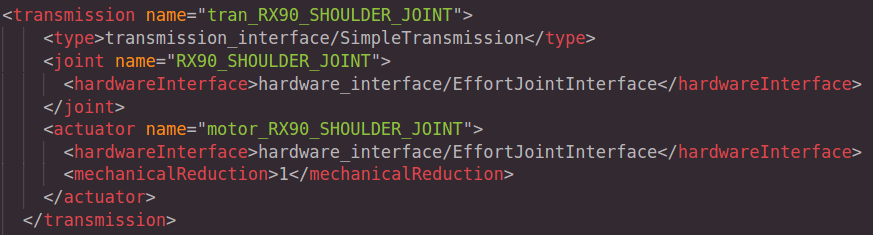


Figura 8. Transmission code.

1. Add the gazebo\_ros\_control plugin

In addition to the transmission tags, a Gazebo plugin needs to be added to our URDF that actually parses the transmission tags and loads the appropriate hardware interfaces and controller manager. By default the gazebo\_ros\_control plugin is very simple, though it is also extensible via an additional plugin architecture to allow power users to create their own custom robot hardware interfaces between ros\_control and Gazebo.

The gazebo\_ros\_control <plugin> tag also has the following optional child elements:

* <robotNamespace>: The ROS namespace to be used for this instance of the plugin, defaults to robot name in URDF/SDF
* <controlPeriod>: The period of the controller update (in seconds), defaults to Gazebo's period
* <robotParam>: The location of the robot\_description (URDF) on the parameter server, defaults to '/robot\_description'
* <robotSimType>: The pluginlib name of a custom robot sim interface to be used.



1. Crear un archivo .yalm con los datos de los controladores(pid)

The PID gains and controller settings must be saved in a yaml file that gets loaded to the param server via the roslaunch file.

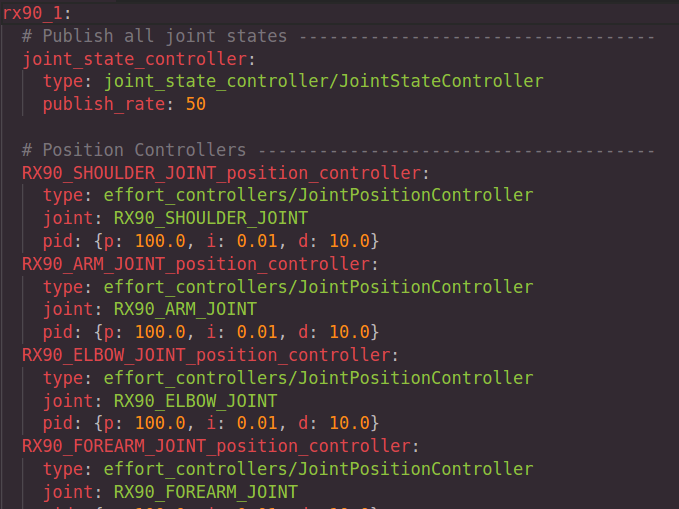


Figura 9. Yalm file.

1. Crear un archivo .launch nuevo y llamarlo desde el principal.

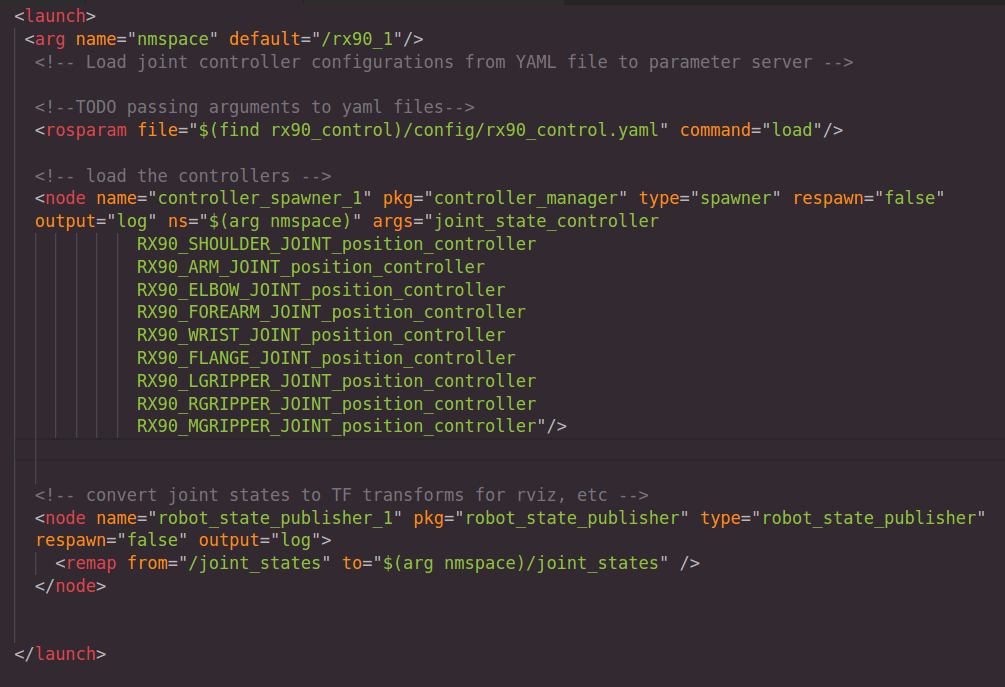


Figura 10. Launch file.

**3.3. Publishers and subscribers**

Cuando todas las controladoras están en funcionamiento y lanzamos el nodo de ROS controller\_spawner, podemos mandarle posiciones a cada articulación en radianes. Esta acción podemos ejecutarla desde cualquier programa que publique con ese topic o desde una terminal con el comando rostopic pub.

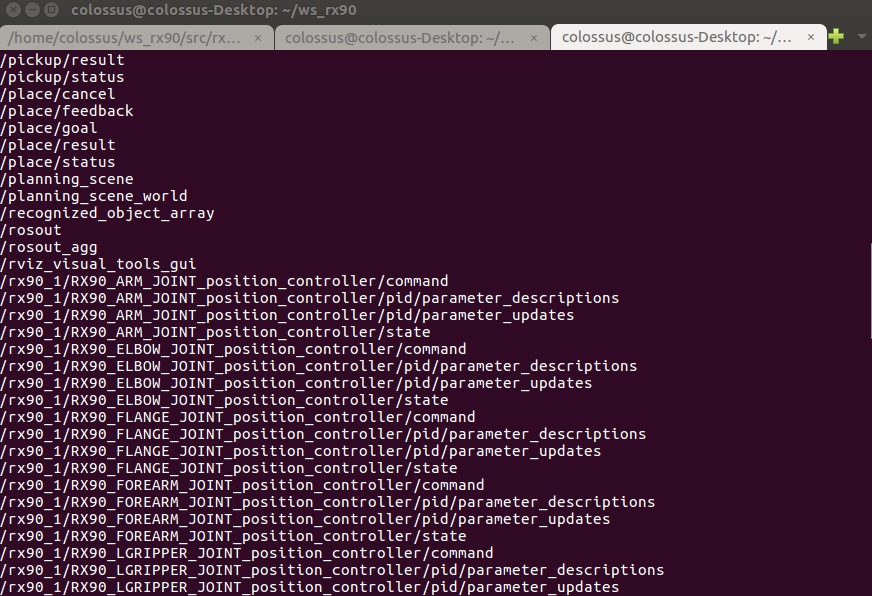
****

Figura 11. List of the nodes used in the simulation

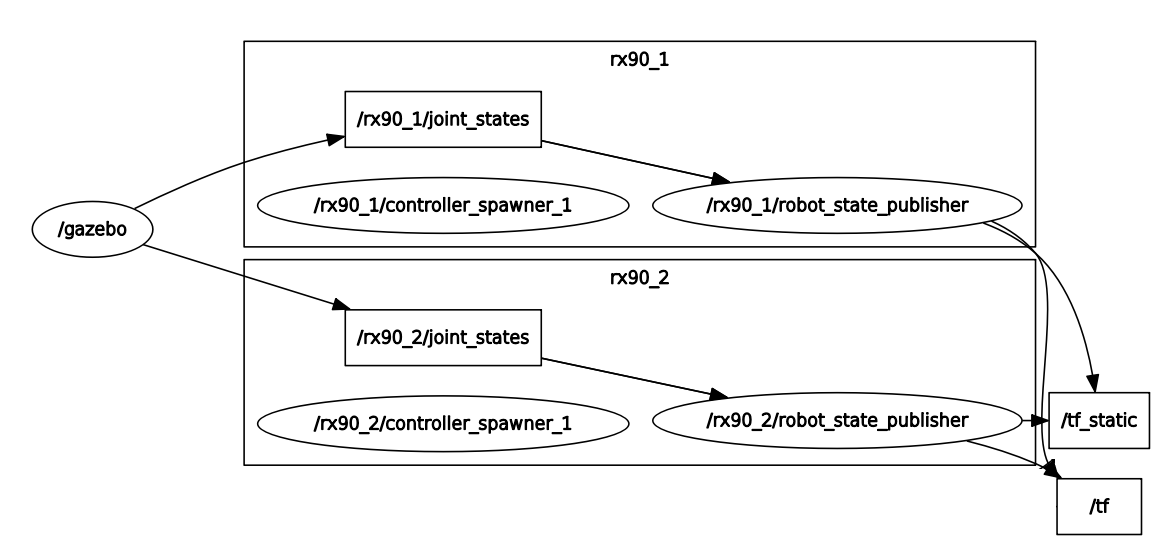
****

Figura 12. Rqt\_graph scheme

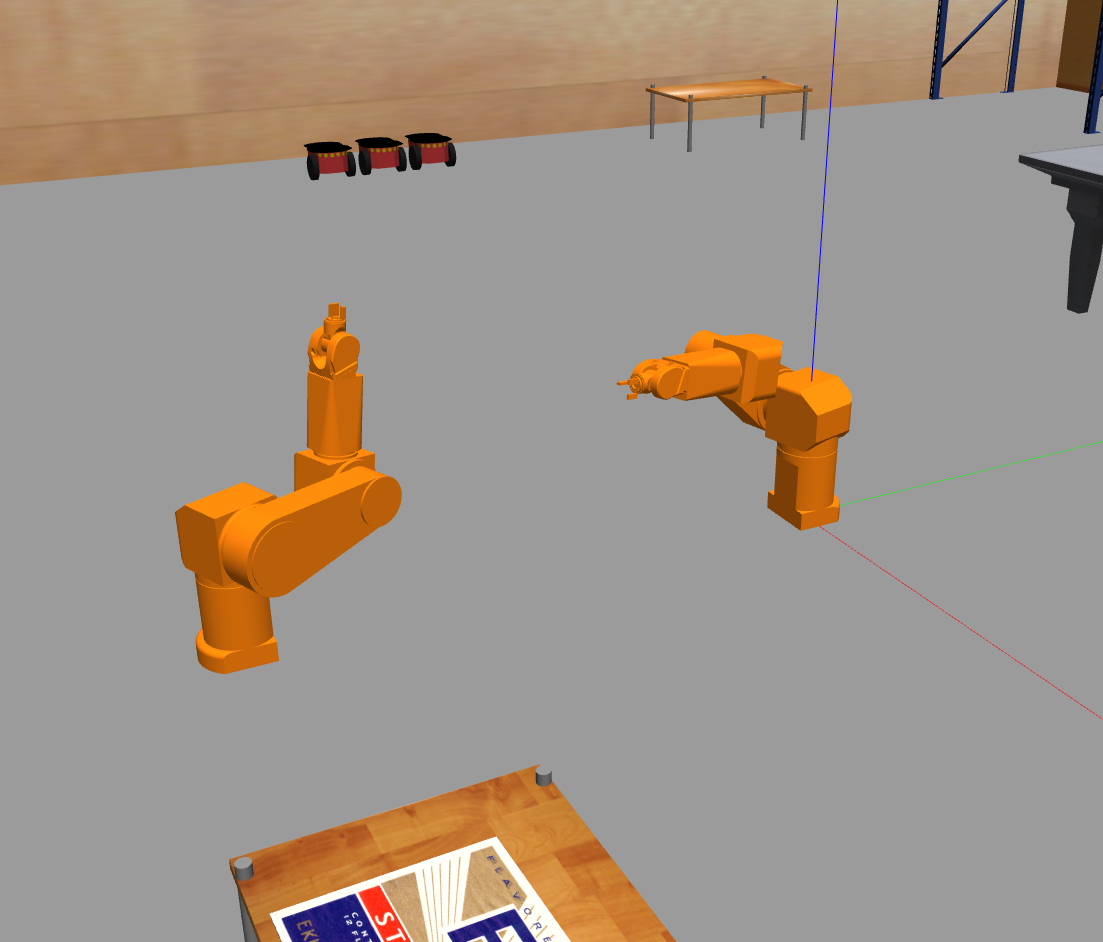
****

Figura 13. Gazebo simulation with movement.

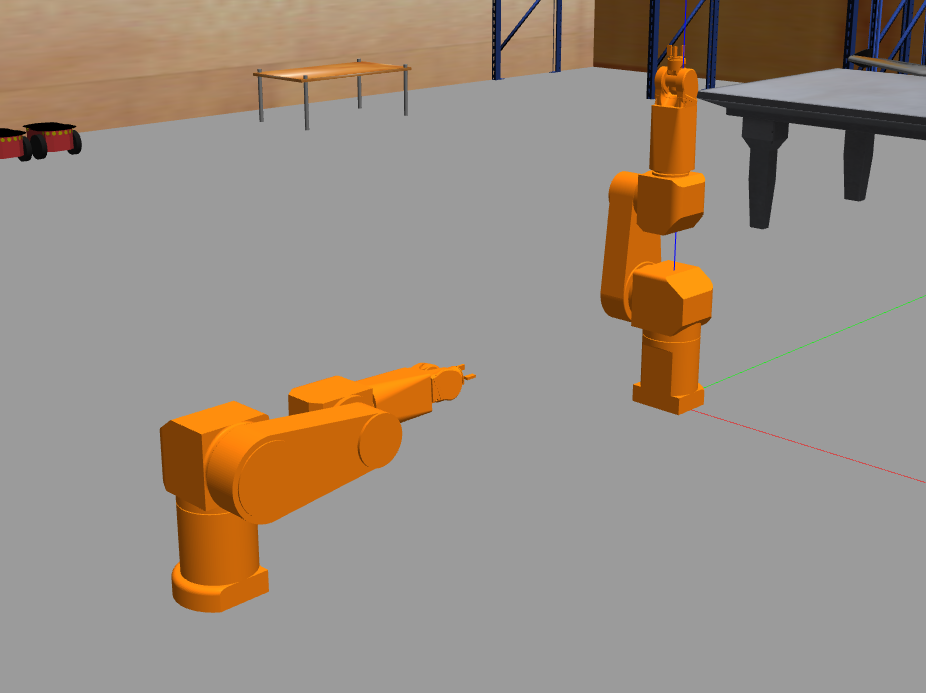


Figura 14. Movement of the rx90 before publishing to the topic.

CHAPTER 4

**Comunicación entre procesos**

Index

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

4.1 Comunicación serie con el Rx90. . . . . . . . . . . . . . . . . . . . . . . . . . . 57

4.2 Comunicación con el simulador . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 62

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

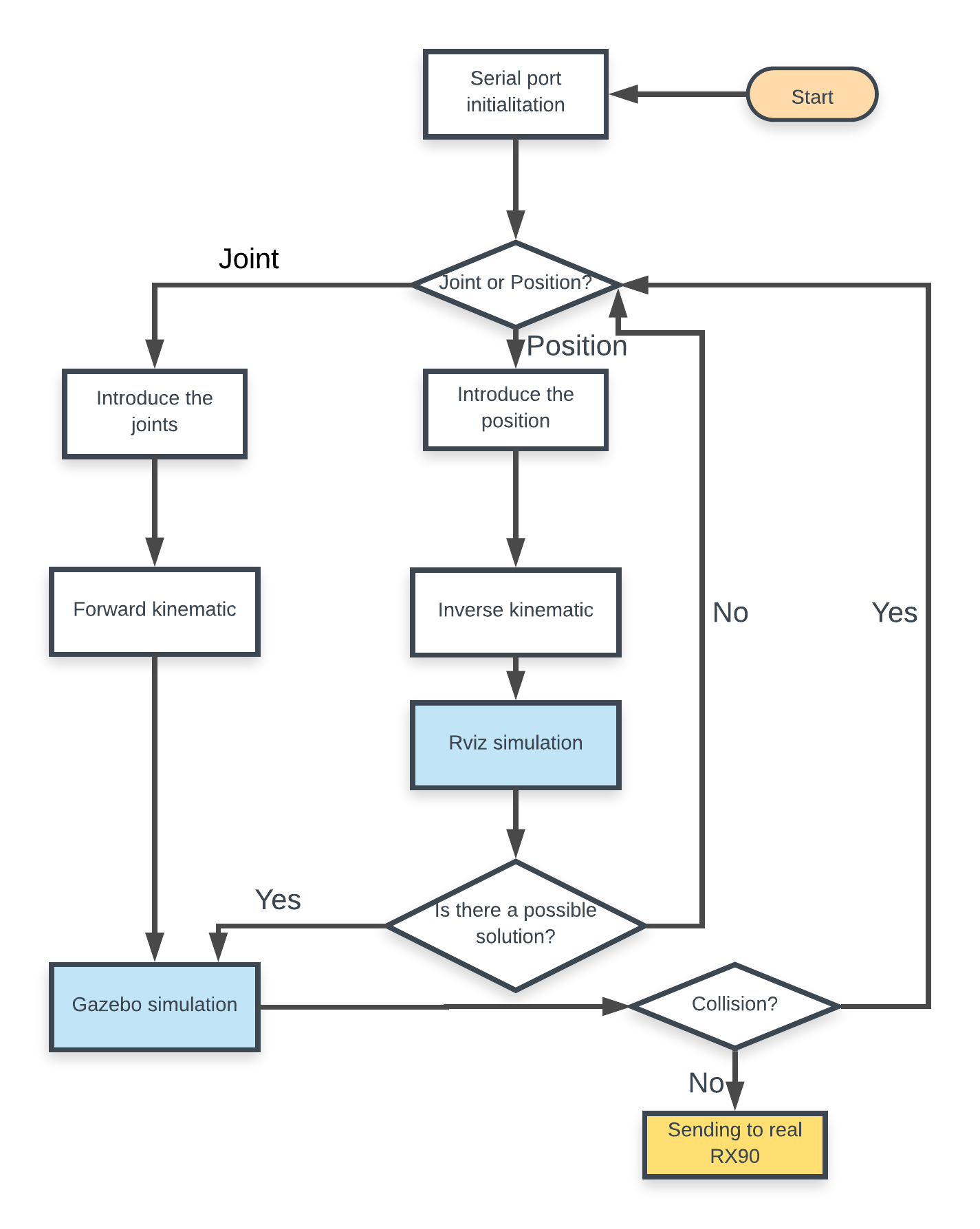
**4.1. Comunicación serie con el Rx90**

**Habla del puerto serie y del lenguaje v+, vaya copia lo de esta mujer**

**4.2. Planteamiento del problema**

Ya que tenemos en funcionamiento el simulador de gazebo, nuestro objetivo es mandar órdenes por puesto serie al robot real siempre que comprobemos anteriormente en simulación que no hay ningún problema de colisión y las posiciones son correctas.

Para ello, seguiremos el siguiente diagrama de flujo:



*Figura 15. Flowchart.*

La cinemática del robot estudia el movimiento del mismo con respecto a un sistema de referencia. La cinemática se interesa por la descripción analítica del movimiento espacial del robot como una función del tiempo, y en particular por las relaciones entre la posición y la orientación del extremo final del robot con los valores que toman sus coordenadas articulares. Existen dos problemas fundamentales a resolver en la cinemática del robot; el primero de ellos se conoce como el problema cinemático directo y el segundo como problema cinemático inverso.

Observando el diagrama, estos problemas están presentes y debemos solucionarlos. El problema cinemático directo consiste en determinar cuál es la posición y orientación del extremo final del robot, con respecto a un sistema de coordenadas que se toma como referencia, conocidos los valores de las articulaciones y los parámetros geométricos de los elementos del robot. El problema cinemático inverso, resuelve la configuración que debe adoptar el robot para una posición y orientación del extremo conocidas.

El problema de la cinemática directa se resuelve fácilmente publicando en Gazebo a cada una de las articulaciones en radianes. Es lo que hacíamos en el apartado 3.3 del presente documento.

Sin embargo, la cinemática inversa en bastante más complicada de resolver. Para ello, usaremos un paquete de ros llamado MoveIt! el cual nos proporcionara si existe una solución a este problema proporcionándole unas coordenadas *x, y, z, roll, pitch, yaw*.

**4.3. Paquete moveit**

MoveIt! 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 and other domains.

En concreto, usaremos The MoveIt! Setup Assistant is a graphical user interface for configuring any robot for use with MoveIt!. Its primary function is generating a Semantic Robot Description Format (SRDF) file for your robot. Additionally, it generates other necessary configuration files for use with the MoveIt! pipeline.

To launch the program, type the following command on the terminal:

roslaunch moveit\_setup\_assistant setup\_assistant.launch

STEP 1: Loading your URDF

Specify the location of an existing URDF for your robot. The robot model will be loaded on the parameter server.

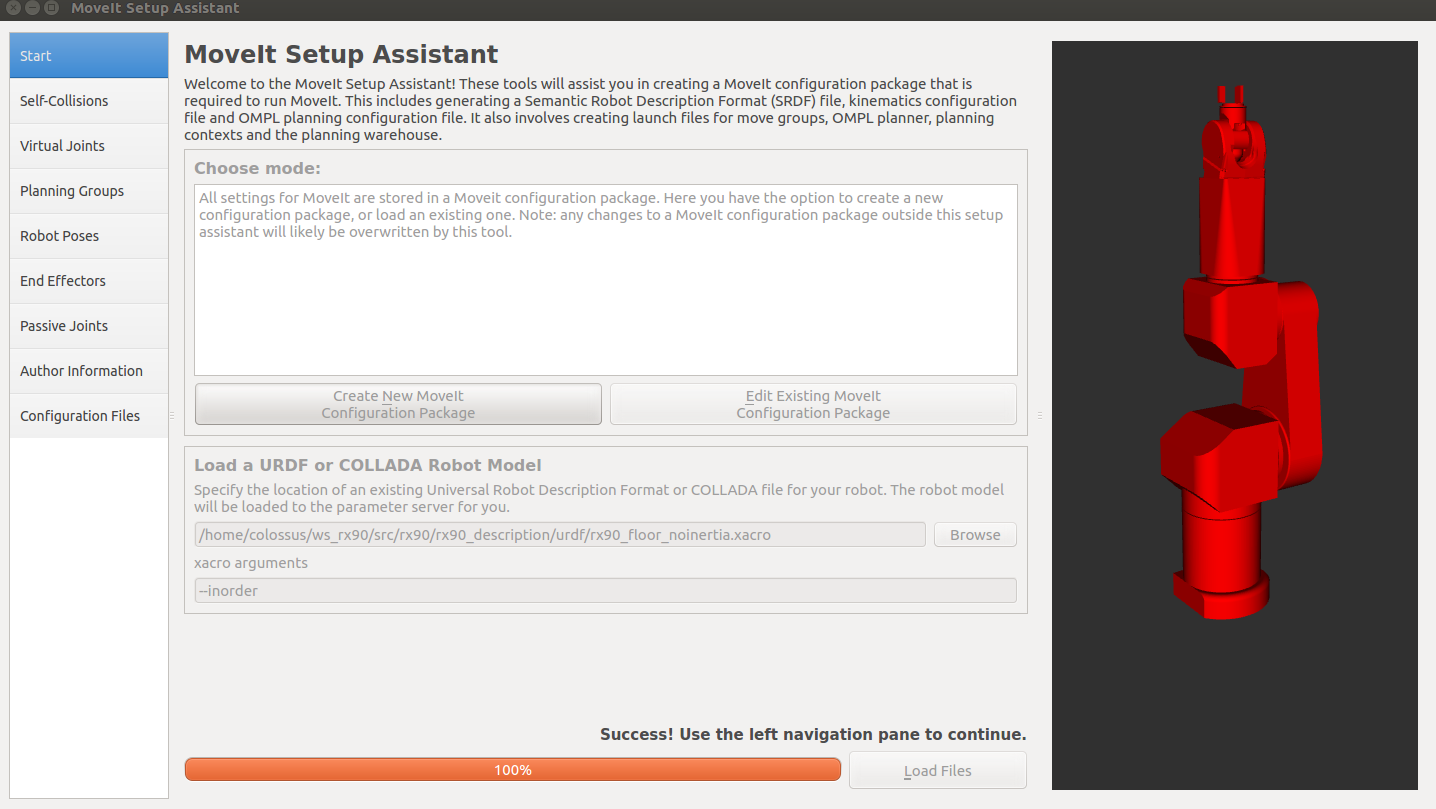


Figura 16. MoveIt! Setup Assistant

STEP 2: Generate Self-Collision Matrix

The Self-Collision Matrix Generator searches for pairs of links on the robot that can safely be disabled from collision checking. These pairs of links are disabled when they are always in collision, never in collision, in collision in the robot’s default position or when the links are adjacent to each other on the kinematic chain. The sampling density specifies how many random robot positions to check for self collision. Higher densities require more computation time while lower densities have a higher possibility of disabling pairs that should not be disabled. Collision checking is done in parallel to decrease processing time.

STEP 3: Add virtual joints

Virtual joints are used primarily to attach the robot to the world. For the RX90 we will define only one virtual joint attaching the “*RX90\_BASE”* link of the RX90 URDF to the world  frame. This virtual joint represents the motion of the base of the robot in a plane.

STEP 4: Add planning groups

Planning groups are used for semantically describing different parts of the robot, such as defining what an arm is, or an end effector. We will define two groups, the arm and the gripper.

STEP 5: Add robot poses

The Setup Assistant allows you to add certain fixed poses into the configuration. This helps if, for example, you want to define a certain position of the robot as a “Home” or “Random” position.

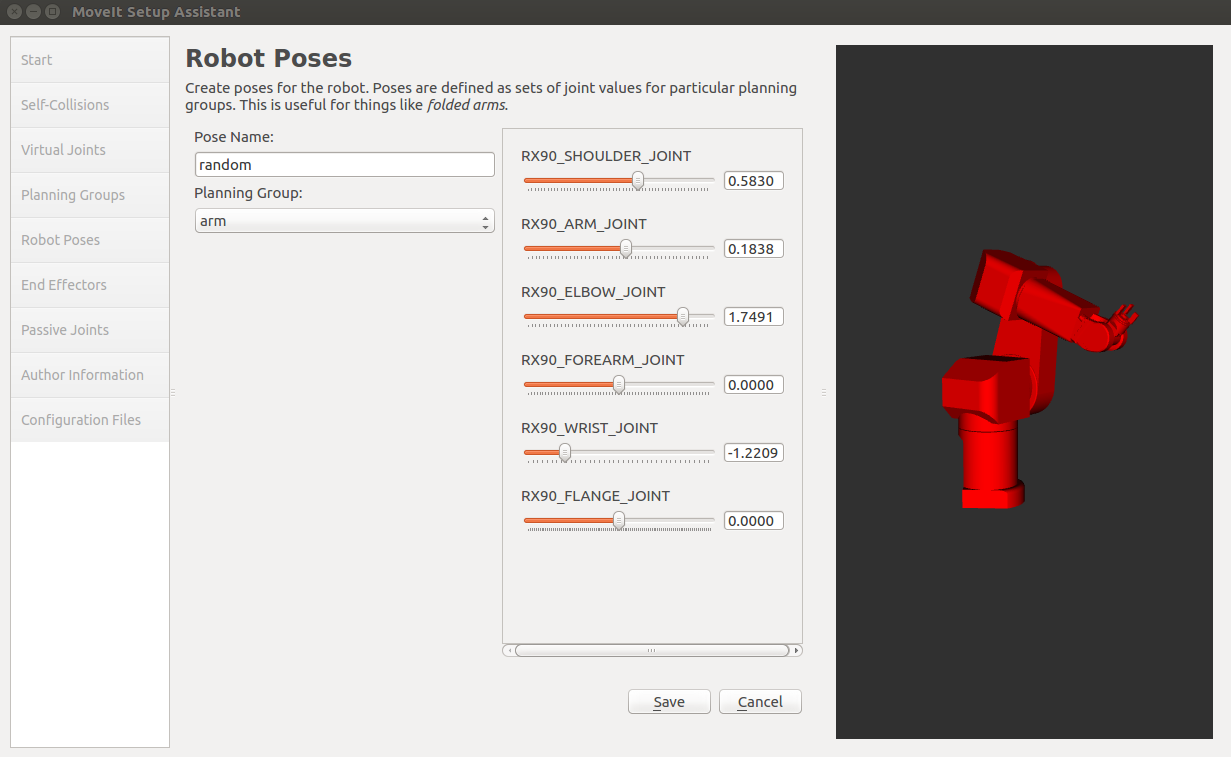


Figura 17. Random position.

STEP 6: Add Passive joints

The passive joints tab is meant to allow specification of any passive joints that might exist in a robot. These are joints that are unactuated on a robot (e.g. passive casters.) This tells the planners that they cannot (kinematically) plan for these joints because they can’t be directly controlled. The RX90 does not have any passive joints so we will skip this step, but it is important to keep it in mind.

STEP 7. Generate configuration files

The Setup Assistant will now generate and write a set of launch and config files into the directory of your choosing.

* 1. **Problema colisiones**

Para resolver el problema de colisiones, hay varias posibles soluciones. La primera es crear un plugin a partir del cual nos informaría de las colisiones. Esta opción quedo descartada dado que solo funciona este plugin al activar desde la gui de gazebo la opción de visualizar contactos. La opción adoptada es la subscripción a un topic de gazebo (los cuales son diferentes a los topics de ROS) el cual nos informaría de alguna posible colisión en el modelo.

Si se produce una colisión, el programa no te dará la opción de poder enviar la posición o articulaciones al robot real.

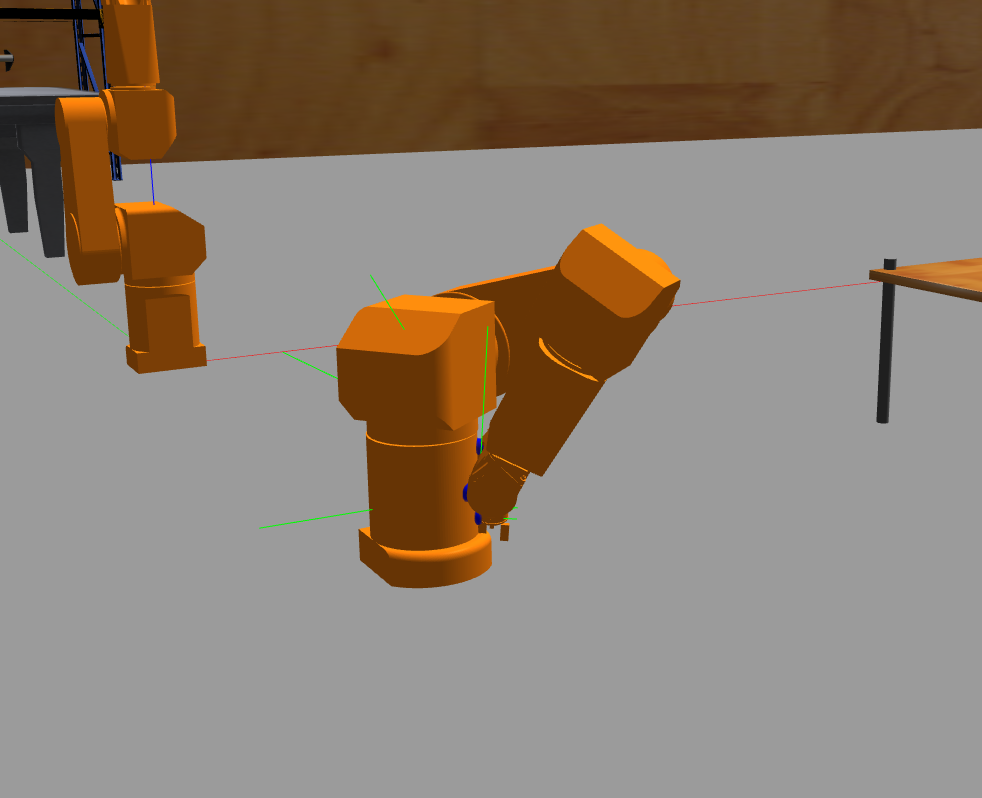


Figura 18. Auto-collision in gazebo

* 1. **Comunicación entre funciones**

CHAPTER 5

**Conclusiones**

Index

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

4.1 Analysis . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 57

4.2 Conclusions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 62

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**5.1. Objetivos conseguidos**

El principal objetivo es la creación de un simulador para el control de los brazos robóticos Staubli RX90. Este simulador se ha llevado a cabo exitosamente pudiendo controlar los brazos y

Por último, todo el conocimiento que he adquirido durante estos meses es de un valor incalculable,

**5.2. Futuras líneas de trabajo**

Al ser un simulador, siempre puede ser mejorado y usado para diferentes aplicaciones. Este simulador puede usarse para distintos proyectos de nivel europeo como AEROARMS o AEROBI, simulando uavs con brazos de seis grados de libertad (parecidos a los Staubli de este trabajo).

# Bibliography

|  |  |
| --- | --- |
| [1] | **Porras Tejada, Azahara Mª.** Control remoto del brazo robótico RX90. |
| [2] | http://gazebosim.org/ |
| [3] | https://www.staubli.com |
| [4] | http://moveit.ros.org/ |
| [5] | http://www.ros.org/ |
| [6] |  |
| [7] |  |
| [9] |  |
| [10] |  |
| [11] |  |
| [12] |  |
| [13] |  |
| [14] |  |