Universidad de Oviedo

ESCUELA POLITÉCNICA DE INGENIERÍA DE GIJÓN.

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

ÁREA DE ROBÓTICA

MANUAL DE USUARIO: CREACIÓN DE PAQUETE MOVEIT Y CONTROL REMOTO DEL ROBOT UR3E

D. Menéndez García, Alejandro
TUTOR: D. Álvarez Prieto Diego

FECHA: (Junio de 2022)

ÍNDICE

1. INTRODUCCIÓN

En este manual se va a explicar el cómo crear un paquete Moveit y como empezar a controlar el robot colaborativo UR3e, instalado en el laboratorio de robótica, teniendo en cuenta el lugar de trabajo. Antes de ello, se explicarán los sistemas y programas que se van a emplear y los pasos previos que hay que realizar para conseguir estos objetivos.

Un punto importante y que cabe recalcar, es que este proyecto no es recomendable hacerlo desde una máquina virtual. Concretamente, esto es debido a que cuando se quiera establecer conexión con el robot, van a existir problemas de conexión entre la máquina virtual, nuestro PC y el robot. Empezar a hacerlo desde una máquina virtual solo permitiría realizar la parte de simulación del robot. Si después se quisiese establecer el control remoto con el robot, se tendría que repetir este último apartado de nuevo en el Pc.

1.1. ROS

ROS (Robotic Operating System) es un middleware de código abierto y gratuito. ROS está divido en dos partes: una parte del sistema operativo ‘ros’ y ‘ros-pkg’ que incluye un conjunto de paquetes, herramientas y librerías que ofrecen la posibilidad de controlar un robot o realizar un conjunto de tareas. De esos conjuntos de paquetes que incluye ROS, están incluidos los entornos de programación de Python y C++ siendo capaz de compilarlos, aunque estén escritos en distintos lenguajes.

Normalmente ROS se usa con el apoyo del sistema operativo de Ubuntu (en este caso la versión Ubuntu 20.04) porque, aunque como se dijo anteriormente ROS tiene una parte de sistema operativo, este no es capaz de hacer las funciones de gestionar los recursos del hardware, organizar y priorizar los servicios de las aplicaciones del software. También

se ha adaptado a otros sistemas operativos como Mac OS X, Windows y Debian. La versión de ROS utilizada es de ROS Noetic.

1.2. QUE ES GAZEBO

Gazebo es un software que efectúa simulación realista tanto de la física de los robots e interacciones de los objetos, así como de la dinámica y sensores de diversos robots. Gazebo funciona utilizando el core de ROS, por lo que permite simular el comportamiento de los robots al utilizar los paquetes desarrollados para ROS. Emplearemos Gazebo para comprobar cómo se comportará el robot en el entorno y si serán seguras las trayectorias que le mandemos hacer a el robot.

1.3. QUE ES MOVEIT

MoveIt! es un software de código abierto para ROS. MoveIt está diseñado para la programación de la manipulación y movimientos de los robots. MoveIt permite crear y simular el entorno de trabajo del robot utilizando objetos de malla 3D diseñados en cualquier programa Gazebo, permitiendo además la interacción entre el robot y el entorno de trabajo.

Con MoveIt podemos planificar el movimiento del robot a cualquier posición teniendo en cuenta la situación actual de los objetos y el robot presentes en el entorno, evitando así colisiones. Pero lo más interesante es que no sólo planifica la trayectoria evitando los obstáculos, sino que además permite interactuar con los objetos del entorno, pudiendo coger cualquier objeto del entorno e incluirlo como si fuera parte del robot a la hora de planificar la trayectoria a la posición deseada.

2. DESCARGA DEL PROYECTO

Como ya se ha comentado anteriormente, se empezará este proyecto casi desde cero, suponiendo que ya se ha trabajado con el robot físicamente, teniendo conocimientos básicos de cómo funciona y con la versión Noetic de ROS ya instalada. Una vez que se tenga esto último instalado, el siguiente paso será bajarse de Github un proyecto en el que exista una descripción del robot (URDF) y los drivers de conexión con el robot. Dicho proyecto será la base de apoyo a la hora de crear el paquete Moveit, el mundo en Gazebo, el control externo del robot y la programación en Python de programas que permitan controlar el robot.

El proyecto de Github empleado ha sido creado por el usuario fmauch y accesible en la dirección:

https://github.com/UniversalRobots/Universal_Robots_ROS_Driver

Lo siguiente será bajarse esos archivos mediante la creación de una configuración del entorno de trabajo que permita realizar todas las operaciones en el entorno de ROS.

2.1. ENTORNO DE TRABAJO

El entorno de trabajo en el que se va a trabajar va a ser la línea de comandos de linux o el intérprete de comandos bash. Con él se va a poder acceder al sistema sin utilizar la interfaz gráfica, es decir, realizar todo tipo de tareas en formato texto mediante órdenes.

Figura 1: Terminal Ubuntu

2.2. INSTALACIÓN DE PROYECTO DE GITHUB

Pulsando el enlace anterior, se abrirá la página web de Github con el proyecto inicial. Para descargarlo y poder trabajar con él, hay que seguir los pasos que se indican más abajo.

```
# source global ros
$ source /opt/ros/<your_ros_version>/setup.bash

# create a catkin workspace
$ mkdir -p catkin_ws/src && cd catkin_ws

# clone the driver
$ git clone https://github.com/UniversalRobots/Universal_Robots_ROS_Driver.git src/Universal_Robots_ROS_Dri

# clone fork of the description. This is currently necessary, until the changes are merged upstream.
$ git clone -b calibration_devel https://github.com/fmauch/universal_robot.git src/fmauch_universal_robot

# install dependencies
$ sudo apt update -qq
$ rosdep update
$ rosdep install --from-paths src --ignore-src -y

# build the workspace
$ catkin_make

# activate the workspace (ie: source it)
$ source devel/setup.bash
```

En resumen, se tiene que abrir el terminal y copiar línea por línea las órdenes que aparecen en la imagen anterior.

```
~$ source /opt/ros/noetic/setup.bash
```

Se usa el comando source en la carpeta de Noetic para que se pueda leer y ejecutar el archivo del fichero.

```
$ mkdir -p catkin_ws/src && cd catkin_ws
```

La instrucción *'mkdir'* crea en la carpeta personal un directorio '*catkin_ws'* y dentro de ese directorio una carpeta *'src'*. También, mediante la instrucción *'cd'* nos situaremos en nuestro terminal en la carpeta *'catkin_ws'* creada.

```
alejandro@alejandro-VirtualBox:~/catkin_ws$ git clone https://github.com/Univers
alRobots/Universal_Robots_ROS_Driver.git src/Universal_Robots_ROS_Driver
```

La instrucción *'git clone'* clona en la carpeta *'src'*, los archivos correspondientes a los Drivers que se necesitarán para poder establecer las comunicaciones del PC con el robot UR3e.

```
alejandro@alejandro-VirtualBox:~/catkin_ws$ git clone -b calibration_devel https
://github.com/fmauch/universal_robot.git src/fmauch_universal_robot
```

Se vuelve a clonar en la carpeta *'src'*, unos archivos de fmauch, el creador del proyecto descargado, que contiene las descripciones del robot, así como archivos que permite lanzar el Gazebo del robot y unos ejemplos de paquetes Moveit, aunque estos no van a ser empleados al final.

```
~/catkin_ws$ sudo apt update -qq
```

La instrucción '*sudo apt-get update -qq'* sirve para actualizar los archivos de índice de paquetes, para así obtener la lista más reciente de paquetes disponibles en los repositorios.

```
~/catkin_ws$ rosdep update
```

La instrucción *'rosdep update'* actualiza los archivos del repositorio.

```
alejandro@alejandro-VirtualBox:~/catkin_ws$ rosdep install --from-paths src --ig
nore-src -y
```

Este comando instala todos los paquetes necesarios para poder usar los archivos descargados anteriormente, por ejemplo Gazebo, los kinematics del robot, paquetes de calibraciones del robot, Moveit (aunque lo que se bajará de Moveit será un paquete no completo), … .

```
~/catkin_ws$ catkin_make
```

Catkin_make es una herramienta de la línea de comandos que añade algunas cosas al flujo de trabajo de catkin.

Una vez realizado estos pasos, ya se puede acceder y ejecutar los archivos descargados, por ejemplo, ejecutar el comando que abre el Gazebo del robot UR3e.

```
~/catkin_ws$ roslaunch ur_gazebo ur3e_bringup.launch
```

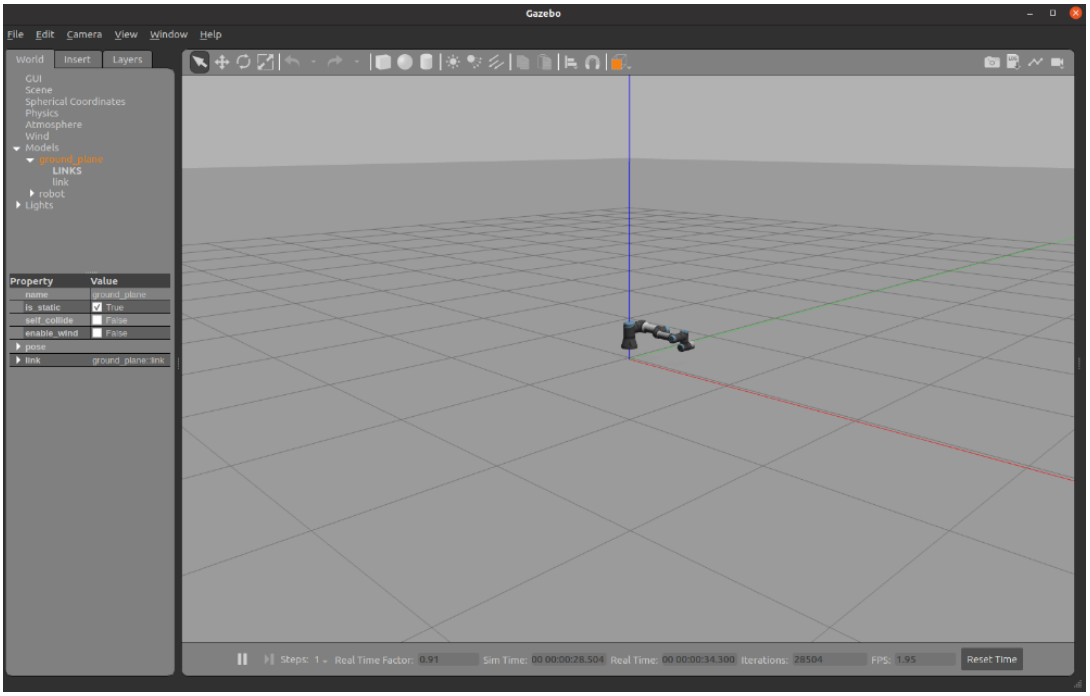


Figura 2: Gazebo inicial UR3e

3. DISEÑO DEL LUGAR DE TRABAJO

El siguiente paso va a ser realizar el diseño del entorno en el que va a trabajar el robot. En el laboratorio de la universidad, el robot está situado sobre una Base, atornillada encima de una mesa. También existe una regleta en la esquina derecha del fondo.

La creación de estos elementos del entorno no se podría realizar directamente utilizando la interfaz de gazebo, debido a que da problemas el crear este entorno y luego lanzar el archivo de gazebo generando el robot encima de la base que se ha creado. Esto se debe a que a la hora de generar el robot encima de la base no se establece ningún *'joint'* entre sus *'links'*, por ejemplo, el de la Base (*Base*) y la base del robot (*base_link*).

Una solución a este problema va a ser crear estos elementos utilizando un sistema propio de ROS formado por marcos denominado XACRO, que utilizará un lenguaje como los archivos URDF (*Unified Robot Description Format*) de XML.

Para ello se creará una carpeta dentro de la carpeta *'src'* donde se guardará los distintos *'.xacro'* correspondientes a los elementos de existentes en el lugar de trabajo. Todas las capturas del código que se ven abajo en las Figuras 3, 4 y 5 son las correspondientes a los objetos reales, con sus correspondientes medidas, instalados en el laboratorio (en el Apartado 7 está el código por si se quiere usar).

```
~/catkin_ws/src$ catkin_create_pkg myur3e_description
```

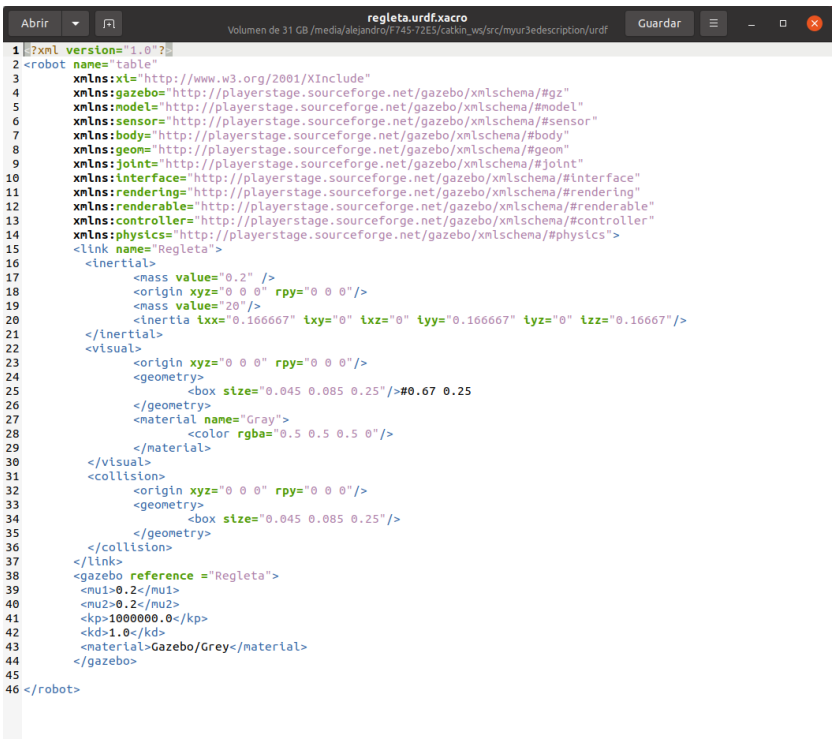
```
~/catkin_ws/src/myur3e_description$ mkdir urdf
~/catkin_ws/src/myur3e_description/urdf$ gedit
```

```xml
<?xml version="1.0"?>
<robot name="table"
       xmlns:xi="http://www.w3.org/2001/XInclude"
       xmlns:gazebo="http://playerstage.sourceforge.net/gazebo/xmlschema/#gz"
       xmlns:model="http://playerstage.sourceforge.net/gazebo/xmlschema/#model"
       xmlns:sensor="http://playerstage.sourceforge.net/gazebo/xmlschema/#sensor"
       xmlns:body="http://playerstage.sourceforge.net/gazebo/xmlschema/#body"
       xmlns:geom="http://playerstage.sourceforge.net/gazebo/xmlschema/#geom"
       xmlns:joint="http://playerstage.sourceforge.net/gazebo/xmlschema/#joint"
       xmlns:interface="http://playerstage.sourceforge.net/gazebo/xmlschema/#interface"
       xmlns:rendering="http://playerstage.sourceforge.net/gazebo/xmlschema/#rendering"
       xmlns:renderable="http://playerstage.sourceforge.net/gazebo/xmlschema/#renderable"
       xmlns:controller="http://playerstage.sourceforge.net/gazebo/xmlschema/#controller"
       xmlns:physics="http://playerstage.sourceforge.net/gazebo/xmlschema/#physics">

       <link name="Base">
         <inertial>
                <mass value="0.5" />
                <origin xyz="0 0 0.5" rpy="0 0 0"/>
                <mass value="20"/>
                <inertia ixx="0.166667" ixy="0" ixz="0" iyy="0.166667" iyz="0" izz="0.16667"/>
        </inertial>
         <visual>
                <origin xyz="0 0 0.0125" rpy="0 0 0"/>
                <geometry>
                        <box size="0.7 0.3 0.025"/>#0.67 0.25
                </geometry>
                <material name="Gray">
                        <color rgba="0.5 0.5 0.5 0"/>
                </material>
         </visual>
         <collision>
                <origin xyz="0 0 0.0125" rpy="0 0 0"/>
                <geometry>
                        <box size="0.7 0.3 0.025"/>
                </geometry>
         </collision>
       </link>
       <gazebo reference ="Base">
         <mu1>0.2</mu1>
         <mu2>0.2</mu2>
         <kp>1000000.0</kp>
         <kd>1.0</kd>
         <material>Gazebo/Orange</material>
       </gazebo>

</robot>
```

Figura 3: Archivo '.xacro' de la Base




```xml
<?xml version="1.0"?>
<robot name="table"
       xmlns:xi="http://www.w3.org/2001/XInclude"
       xmlns:gazebo="http://playerstage.sourceforge.net/gazebo/xmlschema/#gz"
       xmlns:model="http://playerstage.sourceforge.net/gazebo/xmlschema/#model"
       xmlns:sensor="http://playerstage.sourceforge.net/gazebo/xmlschema/#sensor"
       xmlns:body="http://playerstage.sourceforge.net/gazebo/xmlschema/#body"
       xmlns:geom="http://playerstage.sourceforge.net/gazebo/xmlschema/#geom"
       xmlns:joint="http://playerstage.sourceforge.net/gazebo/xmlschema/#joint"
       xmlns:interface="http://playerstage.sourceforge.net/gazebo/xmlschema/#interface"
       xmlns:rendering="http://playerstage.sourceforge.net/gazebo/xmlschema/#rendering"
       xmlns:renderable="http://playerstage.sourceforge.net/gazebo/xmlschema/#renderable"
       xmlns:controller="http://playerstage.sourceforge.net/gazebo/xmlschema/#controller"
       xmlns:physics="http://playerstage.sourceforge.net/gazebo/xmlschema/#physics">
       <link name="Regleta">
         <inertial>
                <mass value="0.2" />
                <origin xyz="0 0 0" rpy="0 0 0"/>
                <mass value="20"/>
                <inertia ixx="0.166667" ixy="0" ixz="0" iyy="0.166667" iyz="0" izz="0.16667"/>
         </inertial>
         <visual>
                <origin xyz="0 0 0" rpy="0 0 0"/>
                <geometry>
                        <box size="0.045 0.085 0.25"/>#0.67 0.25
                </geometry>
                <material name="Gray">
                        <color rgba="0.5 0.5 0.5 0"/>
                </material>
         </visual>
         <collision>
                <origin xyz="0 0 0" rpy="0 0 0"/>
                <geometry>
                        <box size="0.045 0.085 0.25"/>
                </geometry>
         </collision>
       </link>
       <gazebo reference ="Regleta">
        <mu1>0.2</mu1>
        <mu2>0.2</mu2>
        <kp>1000000.0</kp>
        <kd>1.0</kd>
        <material>Gazebo/Grey</material>
       </gazebo>

</robot>
```

Figura 4: Archivo '.xacro' de la regleta

table.urdf.xacro
Volumen de 31 GB /media/alejandro/F745-72E5/catkin_ws/src/myur3edescription/urdf

```
<?xml version="1.0"?>
<robot name="table"
       xmlns:xi="http://www.w3.org/2001/XInclude"
       xmlns:gazebo="http://playerstage.sourceforge.net/gazebo/xmlschema/#gz"
       xmlns:model="http://playerstage.sourceforge.net/gazebo/xmlschema/#model"
       xmlns:sensor="http://playerstage.sourceforge.net/gazebo/xmlschema/#sensor"
       xmlns:body="http://playerstage.sourceforge.net/gazebo/xmlschema/#body"
       xmlns:geom="http://playerstage.sourceforge.net/gazebo/xmlschema/#geom"
       xmlns:joint="http://playerstage.sourceforge.net/gazebo/xmlschema/#joint"
       xmlns:interface="http://playerstage.sourceforge.net/gazebo/xmlschema/#interface"
       xmlns:rendering="http://playerstage.sourceforge.net/gazebo/xmlschema/#rendering"
       xmlns:renderable="http://playerstage.sourceforge.net/gazebo/xmlschema/#renderable"
       xmlns:controller="http://playerstage.sourceforge.net/gazebo/xmlschema/#controller"
       xmlns:physics="http://playerstage.sourceforge.net/gazebo/xmlschema/#physics">

  <property name="table_height" value="0.75" />
  <property name="table_width" value="1.5" />
  <property name="table_depth" value="1.0" />
  <property name="leg_radius" value="0.02" />
  <property name="table_x" value="0.75" />
  <property name="table_y" value="0.5" />
  <property name="table_z" value="0.0" />

  <property name="table_top_thickness" value="0.08"/>

  <property name="M_PI" value="3.1415926535897931" />


  <!-- tabletop height is .55+.01+.025=.585 -->
  <link name="table_top_link">
    <inertial>
      <mass value="1.0" />
      <!--origin xyz="0.75 0.5 0.71" /-->
      <origin xyz="0 0 0" />
      <inertia ixx="1" ixy="0"  ixz="0"
               iyy="1" iyz="0"
               izz="1" />
    </inertial>
    <visual>
      <origin xyz="0.75 0.5 0.73" />
      <geometry>
        <box size="1.5 1 0.08" />
      </geometry>
    </visual>
    <collision>
      <origin xyz="0.75 0.5 0.73" />
      <geometry>
        <box size="1.5 1 0.08" />
      </geometry>
    </collision>
```

Figura 5: Archivo '.xacro' de la mesa

Una vez se han creado los *'.xacro'* de los elementos del entorno, lo siguiente será establecer los *'joint'* entre ellos y el robot. Esto se hará modificando dos archivos situados en distintos directorios.

El primero archivo que se modificará es *'ur3e_macro.xacro'* situado en el directorio 'catkin_ws\src\fmauch_universal_robot\ur_description\urdf\inc'. Se añadirá el código de la siguiente imagen justo antes del </robot>, abajo del todo del archivo.

```
<link name="world"/>

<xacro:include filename="$(find myur3edescription)/urdf/Base.urdf.xacro"/>
<joint name="world_joint" type="fixed">
  <parent link="world"/>      <!--world -->
  <child link="Base"/>       <!--Base -->
  <origin xyz="0 0 0" rpy="0 0 0"/>
</joint>

<joint name="base_joint" type="fixed">
              <parent link="Base"/>        <!--Base -->
              <child link="base_link"/>    <!--base_link-->
              <origin xyz="0 0 0.025" rpy="0.0 0.0 0.0"/>
</joint>
<xacro:include filename="$(find myur3edescription)/urdf/table.urdf.xacro"/>
<joint name="table_joint" type="fixed">
              <parent link="Base"/>               <!--Base -->
              <child link="table_top_link"/>      <!--table_top_link -->
              <origin xyz="-0.43 -0.2 -0.7725" rpy="0 0 0" />
</joint>

<xacro:include filename="$(find myur3edescription)/urdf/regleta.urdf.xacro"/>
<joint name="Regleta_joint" type="fixed">
              <parent link="Base"/>               <!--Base -->
              <child link="Regleta"/>             <!--Regleta -->
              <origin xyz="-0.41 -0.175 0.12" rpy="0.0 0.0 0.0"/>
</joint>
```

Figura 6: Joints de los distintos objetos del lugar de trabajo

Lo que se está haciendo en este código es importar los *'.xacro'* creados anteriormente. Por ejemplo, con el *'.xacro'* de la Base donde va apoyado nuestro robot, se crea un *'joint'* entre el link de nuestro mundo vacío (*'world'*) y el link de la Base creada, estableciéndolo como el origen de nuestro entorno. Después se establecerá un *'joint'* entre el link de la base del robot (*'base_link'*) con el link de la Base ('Base'), donde el origen de este *'joint'* respecto al que se estableció antes como origen, está a 0.025m en el eje z. Esto se hará con el resto midiendo aproximadamente donde podrían estar estos objetos situados respecto a nuestra Base.

El segundo archivo, que se tendrá que modificar, se llama *'ur.xacro'* y está situado en el directorio 'catkin_ws\src\fmauch_universal_robot\ur_gazebo\urdf'. El código que se añadirá será el mismo que el anterior, sustituyéndolo por el código de un *'joint'* que se encuentra abajo del archivo.

3.1. SPAWN Y POSICIÓN INICIAL DEL ROBOT

Con lo que se hizo anteriormente, se ha creado los objetos existentes en nuestro lugar de trabajo. El siguiente paso que queda por hacer es generar (spawnear) correctamente el robot encima de la Base y cambiar la posición inicial en la que generar el robot, porque si no se generaría en la posición inicial de la Figura2 y en el laboratorio el robot parte de una posición inicial distinta. Para esto, se modificará el archivo llamado *'ur_control.launch.xml'* situado en el directorio 'catkin_ws\src \fmauch_universal_robot\ur_gazebo\launch\inc'. Los cambios que se harán en el apartado de 'spawn' del archivo serán:

Abrir *ur_control.launch.xml ~/catkin_ws/src/fmauch_universal_robot/ur_gazebo/launch/inc Guardar

ur_control.launch.xml *ur_control.launch.xml

```
21  <arg name="stopped_controllers" default="joint_group_pos_controller"/>
22
23  <!-- Gazebo parameters -->
24  <arg name="gazebo_model_name" default="robot" doc="The name to give to the model in Gazebo (after spawning
    it)." />
25  <arg name="gazebo_world" default="worlds/empty.world" doc="The '.world' file to load in Gazebo." />
26  <arg name="gui" default="true" doc="If true, Gazebo UI is started. If false, only start Gazebo server." />
27  <arg name="paused" default="false" doc="If true, start Gazebo in paused mode. If false, start simulation as
    soon as Gazebo has loaded." />
28  <arg name="robot_description_param_name" default="robot_description" doc="Name of the parameter which
    contains the robot description (ie: URDF) which should be spawned into Gazebo." />
29  <arg name="spawn_z" default="0.1" doc="At which height the model should be spawned. NOTE: lower values will
    cause the robot to collide with the ground plane." />
30  <arg name="start_gazebo" default="true" doc="If true, Gazebo will be started. If false, Gazebo will be
    assumed to have been started elsewhere." />
31
32  <!-- Load controller settings -->
33  <rosparam file="$(arg controller_config_file)" command="load"/>
34
35  <!-- Start Gazebo and load the empty world if requested to do so -->
36  <include file="$(find gazebo_ros)/launch/empty_world.launch" if="$(arg start_gazebo)">
37    <arg name="world_name" value="$(arg gazebo_world)"/>
38    <arg name="paused" value="$(arg paused)"/>
39    <arg name="gui" value="$(arg gui)"/>
40  </include>
41
42  <!-- Spawn the model loaded earlier in the simulation just started -->
43  <node name="spawn_gazebo_model" pkg="gazebo_ros" type="spawn_model"
44    args="
45      -urdf
46      -param $(arg robot_description_param_name)
47      -model $(arg gazebo_model_name)
48      -x 0.43
49      -y 0.2
50      -z $(arg spawn_z)
51      -J shoulder_lift_joint -1.57
52      -J wrist_1_joint -1.57"
53    output="screen" respawn="false" />
54
55  <!-- Load and start the controllers listed in the 'controllers' arg. -->
56  <node name="ros_control_controller_spawner" pkg="controller_manager" type="spawner"
```

```
<arg name="spawn_z" default="0.775" doc="At which height the model should be spawned. NOTE: lower values will
cause the robot to collide with the ground plane." />
```

Figura 7: Spawn y posición inicial del robot

Básicamente lo que se hace es cargar el URDF, parámetros y modelo del robot. El lugar donde se generará, que será encima de la Base, corresponde con las coordenadas -x 0.43 -y 0.3 -z 0.775 (la coordenada z se cambia en la línea 29 del código del mismo archivo). En cuanto a la posición inicial, queremos que el Hombro (shoulder_lift) y la Muñeca1 (wrist_1) estén ambas a -90º, con lo que se tendrá que poner sus juntas a $-90º \cong -1.57 rad/s$.

Como último paso, se tiene que iniciar el Gazebo pausado. Este paso es importante hacerlo porque hay veces que, dependiendo de la capacidad de lectura y ejecución de tu ordenador, al ejecutar el *'.launch'* del gazebo, este se ejecuta demasiado rápido y no le da tiempo a cambiar la posición inicial del robot. Para ello cambiaremos el *false* por *true* del siguiente código del *'ur3e_bringup.launch'*:

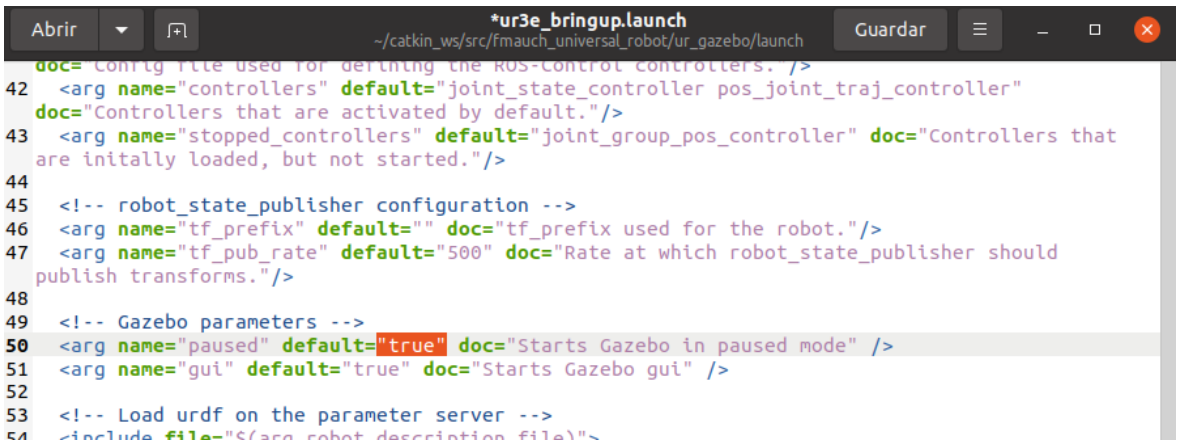


Figura 8: Cambio Gazebo inicialmente pausado

Una vez hecho, ya se podrá ejecutar el Gazebo. El resultado será:

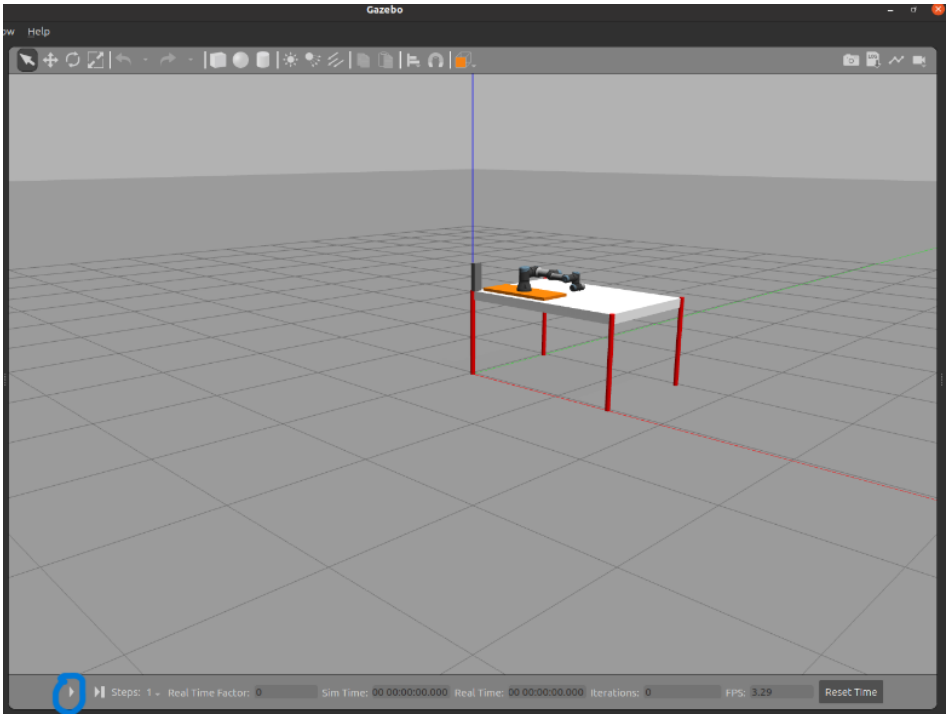


Figura 9: Gazebo Ur3e inicialmente pausado

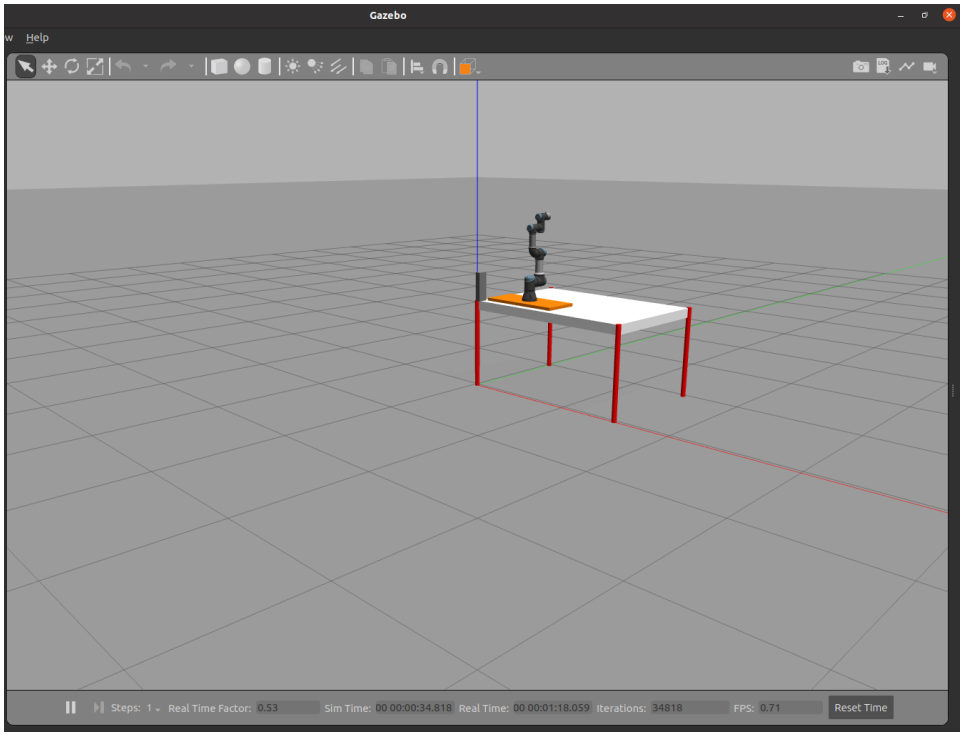


Figura 10: Gazebo del Robot UR3e

4. CREACIÓN DEL PAQUETE MOVEIT

A partir de aquí, se va a tratar el cómo crear un paquete Moveit propio correspondiente a nuestro robot. Un paso previo que se va a hacer es instalar el paquete Moveit completo, ya que cuando al principio se instaló los archivos de Github, el paquete de Moveit que instaló no era completo y se necesita completo para, por ejemplo, ejecutar el Moveit Setup Assintant que es necesario para crear nuestro Moveit.

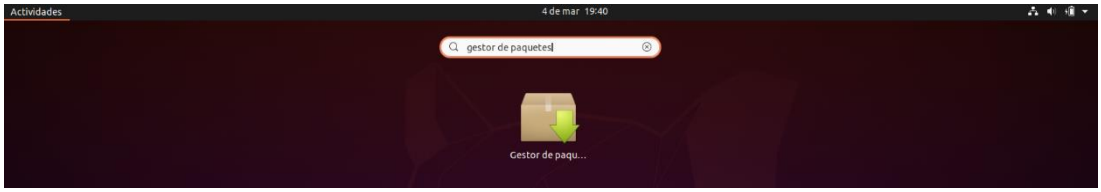
4.1. SYNAPTIC

Para instalar el paquete Moveit completo, es recomendable que se haga a través del gestor de paquetes Synaptic. Si no se tiene el gestor de paquetes Synaptic, es recomendable que se instale, porque si en un futuro se necesita descargar más paquetes, resulta más cómodo mediante Synaptic que bajarlos escribiendo comandos en el terminal.

Para bajarse Synaptic, abre el terminal y ejecuta el siguiente comando:

```
~$ sudo apt-get install synaptic
```

Una vez instalado Synaptic, pon en el buscador ‘gestor de paquetes’ y haz clic en el icono que te aparece a continuación:



Es posible que pida cada vez que lo abras la contraseña de inicio de tu PC. Ponla y después se abrirá el siguiente menú:

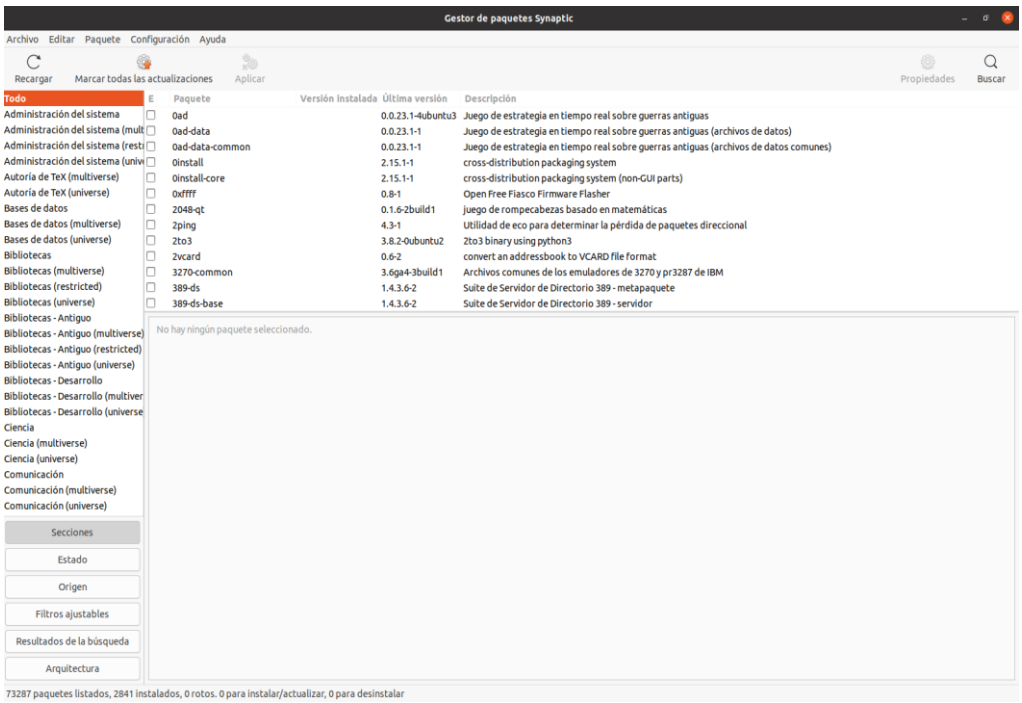


Figura 11: Gestor de paquetes Synaptic

Se puede buscar cualquier paquete haciendo click en el icono de arriba a la derecha. Se abrirá el siguiente menú de buscar:

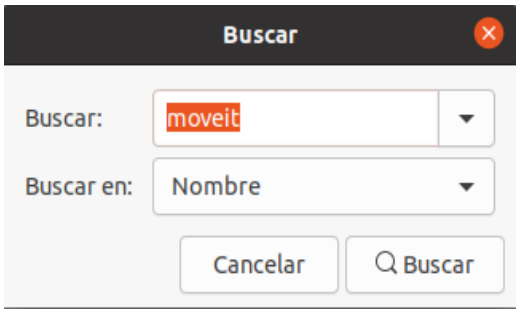


Figura 12: Buscador de Synaptic

La búsqueda, como se puede ver en la imagen anterior, se hace mediante palabras clave que se encuentren escritas en el nombre del paquete. Aunque, también se pueden buscar esas palabras, por ejemplo, en el nombre y en la descripción. Se da a buscar, y de los paquetes que aparecen, hay que darle a descargar al siguiente:

```
ros-noetic-moveit          1.1.8-1focal.20220: 1.1.8-1focal.20220: Meta package that contains all essential package of MoveIt.
```

Hay que descargar todos los paquetes secundarios que dice que se van a descargar e instalar al aplicar la descarga de *'ros-noetic-moveit'*.

4.2. MOVEIT SETUP ASSISTANT

Una vez aplicados todos los pasos previos ya se puede empezar a crear el propio Moveit de nuestro robot. Lo primero que se hará, será abrir el Moveit Setup Assistant, que ayudará a crear el paquete Moveit. Para ello, hay que ejecutar en el terminal el comando:

```
~/catkin_ws$ roslaunch moveit_setup_assistant setup_assistant.launch
```

4.3. START

Al ejecutar el comando anterior se abrirá la siguiente pestaña:

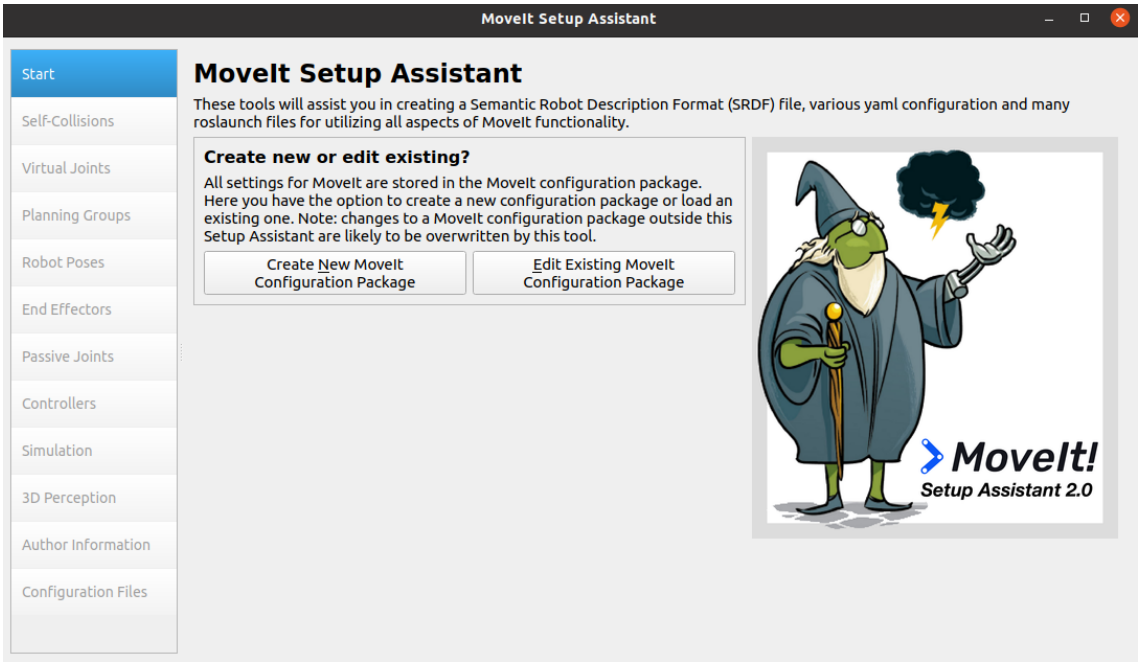


Figura 13: Pantalla de inicio del Moveit Setup Assistant

Como se puede observar, se puede tanto crear un paquete Moveit nuevo como editar uno ya existente. En este caso, se va a crear uno nuevo, así que hay que hacer click en *'Create New Moveit Configuration Package'.* Se abrirá:

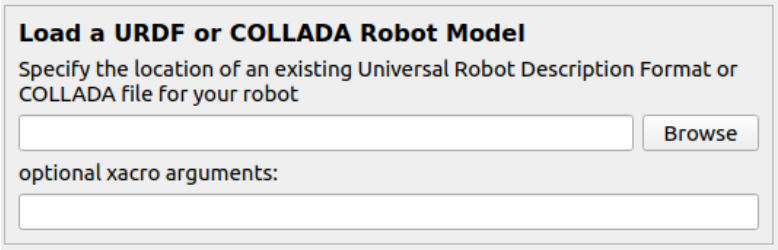


Figura 14: Interfaz de cargar archivos

Esta última imagen dice que hay que pasarle el URDF del robot. Le damos a browse y buscamos el *'.xacro'* del robot UR3e que está situado en el directorio 'catkin_ws/src/fmauch_universal_robot/ur_description/urdf':

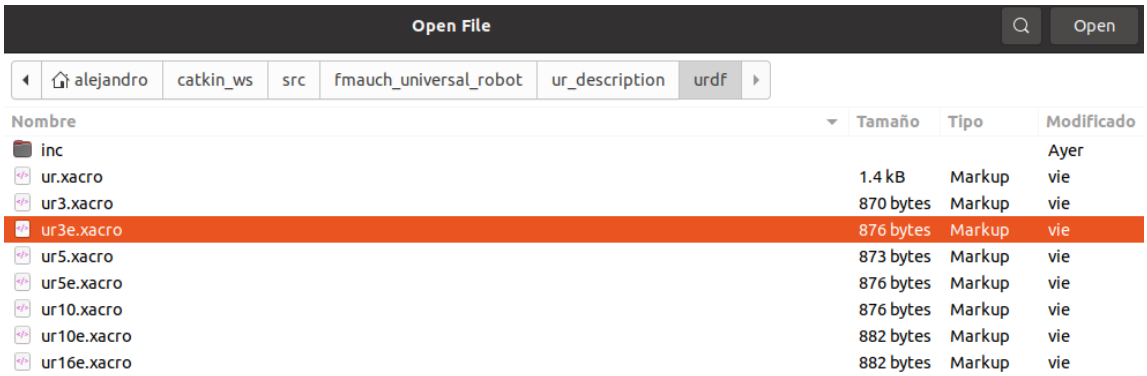


Figura 15: Directorio y archivo ur3e.xacro

Este *'.xacro'* está vinculado a archivos, los cuales se han modificado anteriormente, para que el robot se adaptase a el lugar de trabajo, con lo que no habrá que pasarle ningún *'.xacro'* opcional. Se hace click en Load Files y se espera a que cargue todos los archivos.

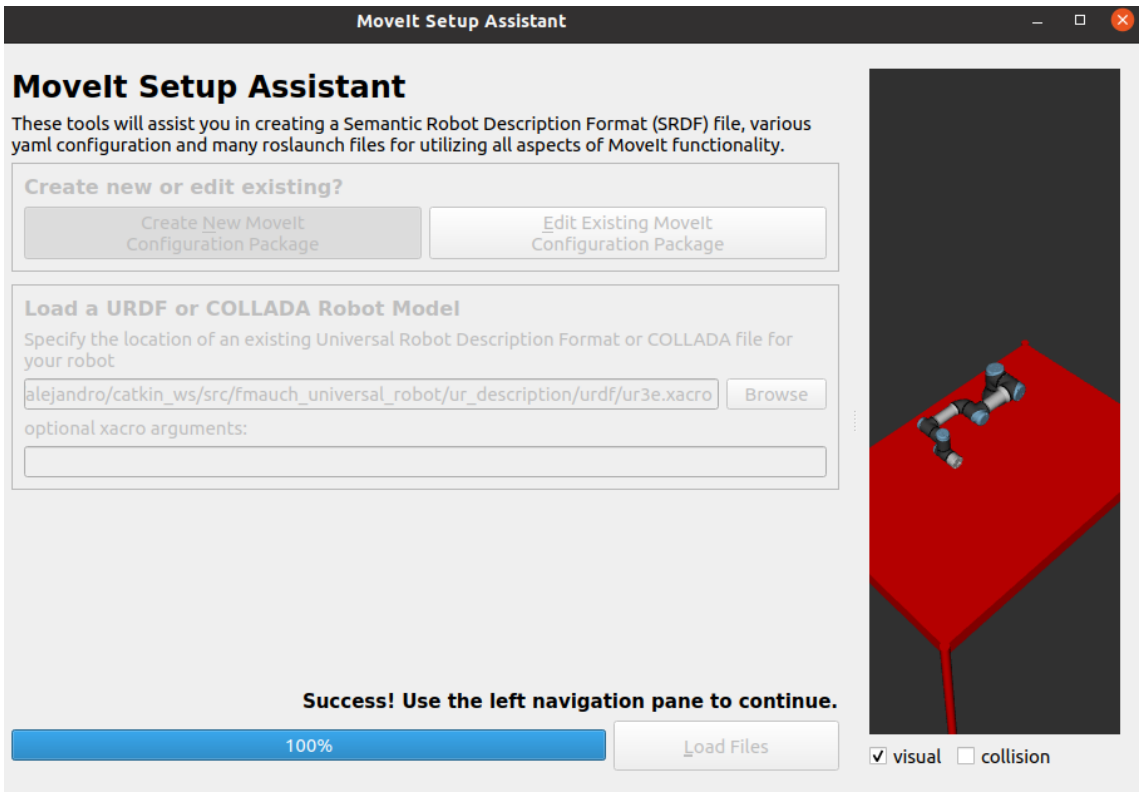


Figura 16: Archivo cargado

Una vez cargados, si nos fijamos en la última imagen, a la derecha, aparece visible el robot y la mesa. La Base y la regleta también están ahí, pero por alguna razón solo se mostrarán después en momentos de planificar trayectorias con colisión, con lo que en un principio no supone un problema que cause fallo en la realidad.

4.4. SELF-COLLISIONS

El siguiente paso es generar las Self-Collisions (Auto-Colisiones). Se puede pinchar ya por defecto, en Generate Collision Matrix y te las generará automáticamente.

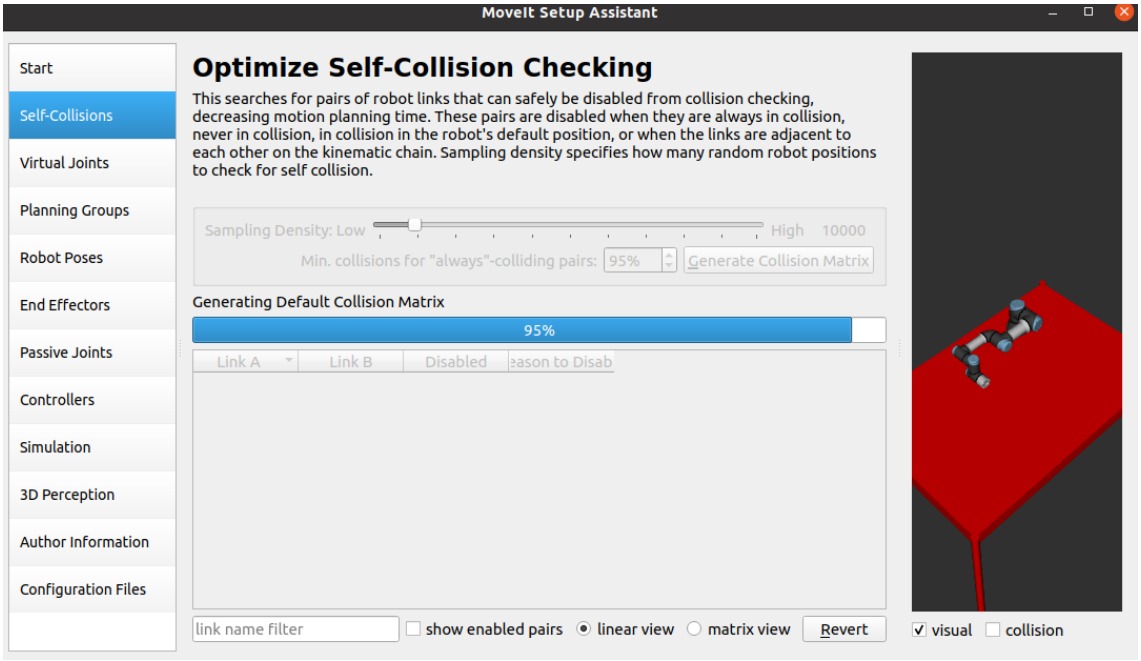


Figura 17: Generar las Self-collisions

4.5. VIRTUAL JOINTS

El siguiente paso es definir las juntas virtuales (virtual joints). En este caso, simplemente se va a hacer un joint que junte el link de la base del robot (base_link) con el mundo en el que vaya a estar situado. Para añadir cualquier junta virtual hay que darle a 'Add Virtual Joint'.

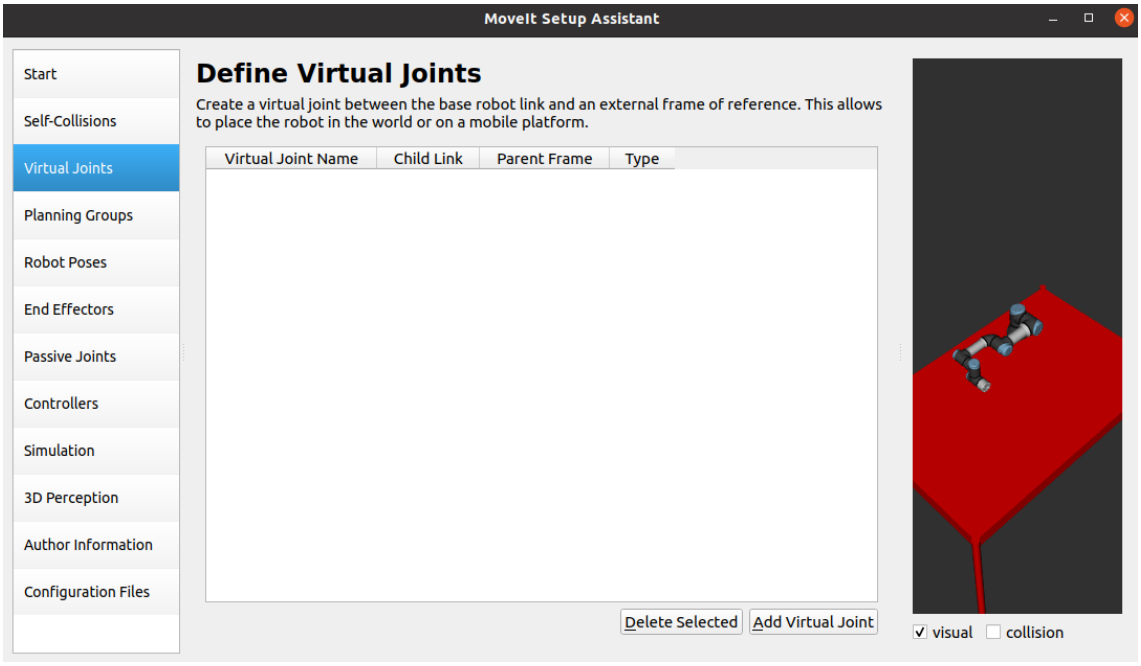


Figura 18: Interfaz Virtual Joints

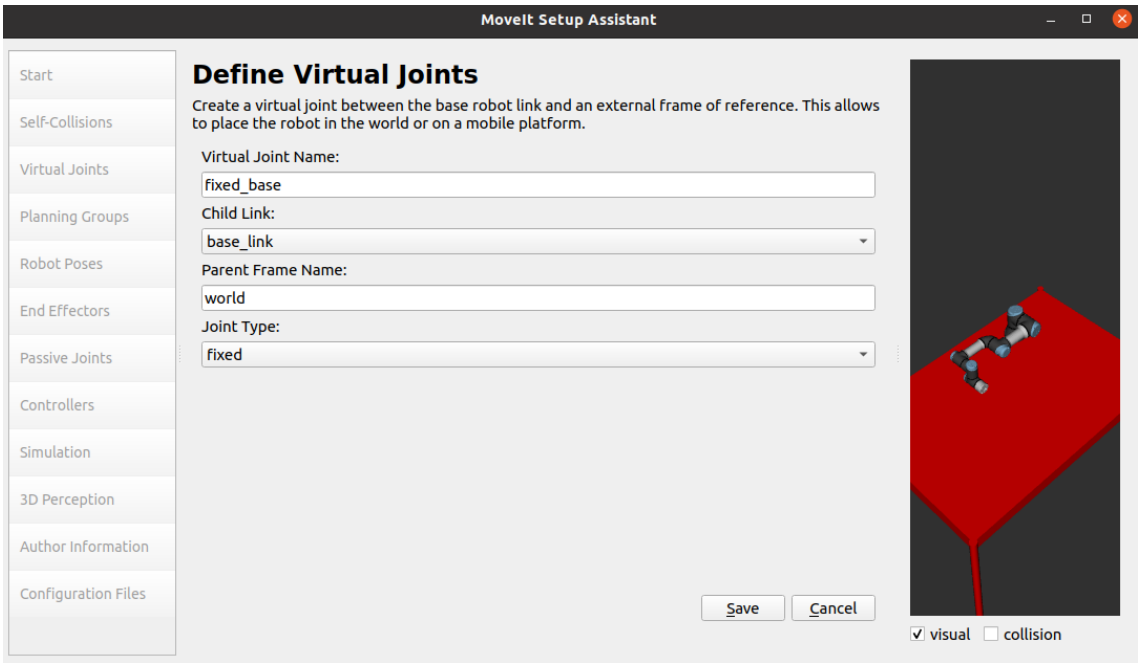


Figura 19: Configuración virtual joints

(Nota: En todos los pasos acordarse de darle a Save)

4.6. PLANNING GROUPS

El siguiente paso es definir cuáles van a ser los Planning Groups. Por lo general los Planning Groups de un robot, de los cuales se va a poder ver su comportamiento, son el robot en sí y la herramienta que va atornillada a la Muñeca3.

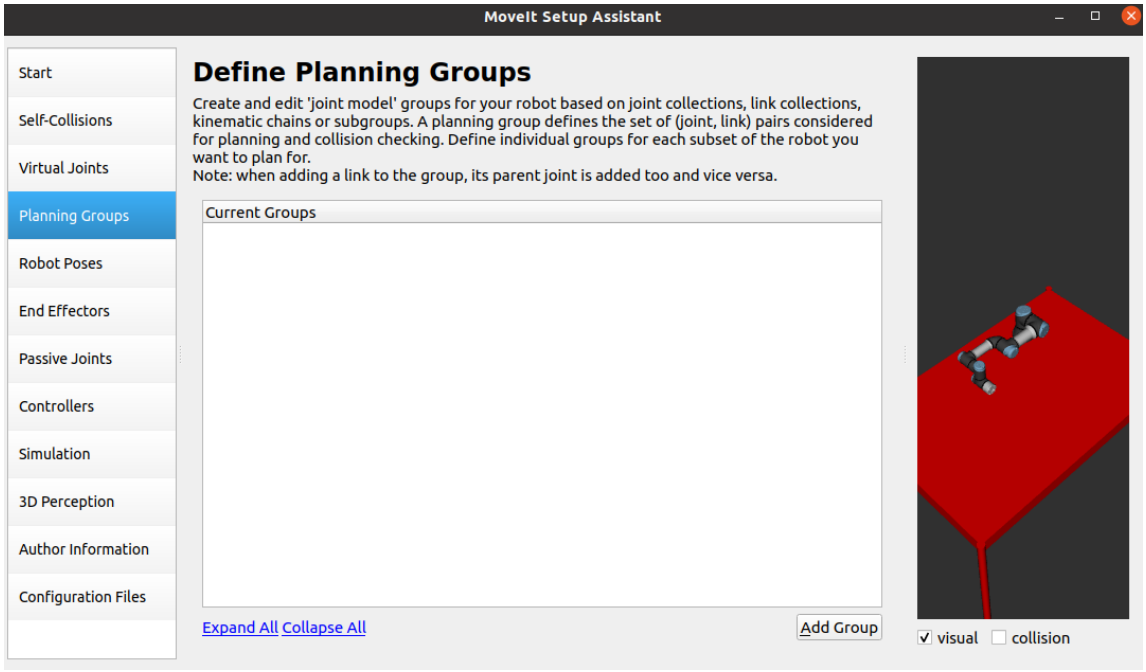


Figura 20: Planning Groups

Dándole a Add Group, se procederá a añadir primero el grupo correspondiente al robot, el cual se llamará *'manipulator'*. Se usará el KDLKinematicsPlugin para resolver los aspectos del movimiento aparte de las consideraciones de masa y fuerza.

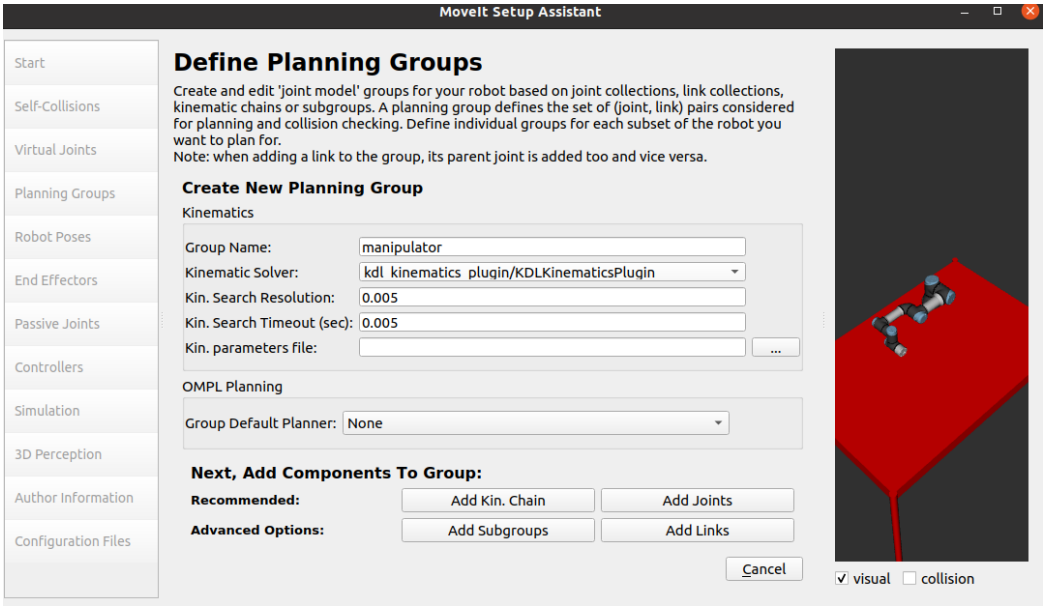


Figura 21: Configuración del manipulator

Como dijimos, este grupo *'manipulator'* trabaja con todo el robot, con lo que tendremos que pinchar en 'Add Kin. Chain' para decirle desde que link hasta que link se moverá el robot cuando se maneje el Planning Group *'manipulator'*. Los movimientos irán desde el link de la base del robot (base_link) hasta el link de la herramienta (tool0).

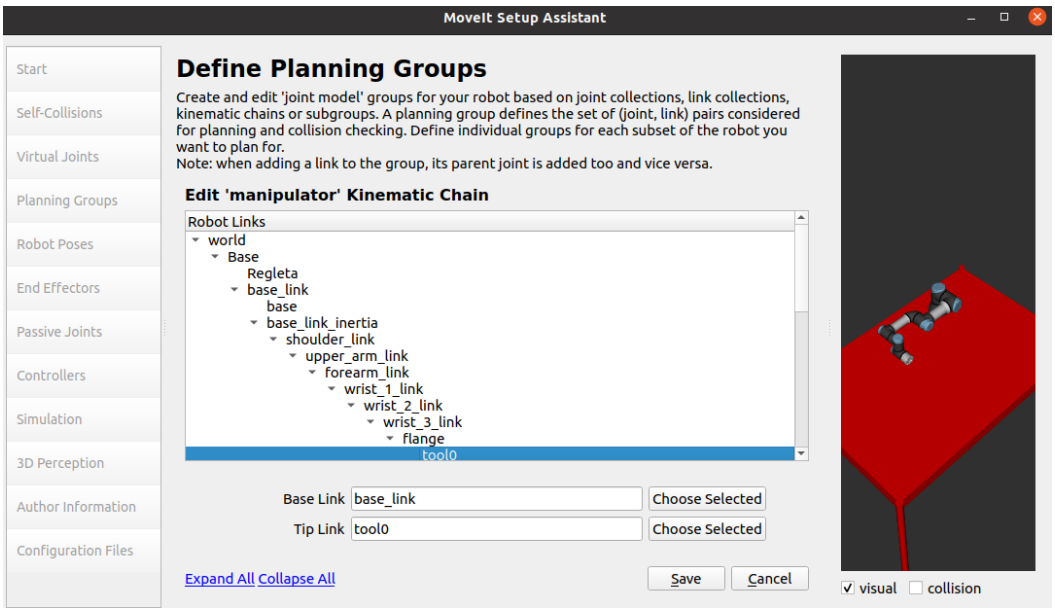


Figura 22: Configuración manipulator

Se guarda (Save) y con esto se tendrá el grupo correspondiente al manipulador del robot. Ahora queda el grupo que permitiría muestrear el comportamiento de la herramienta. Para ello volvemos a dar a Add Group y a este nuevo grupo lo llamaremos *'endeffector'*.

Como a la hora de hacer este manual no se había instalado una herramienta en nuestro robot no vamos a poner ningún Kinematic Solver. Antes de dar a Save, hay que añadirle el link, esta vez correspondiente a la herramienta (tool0).

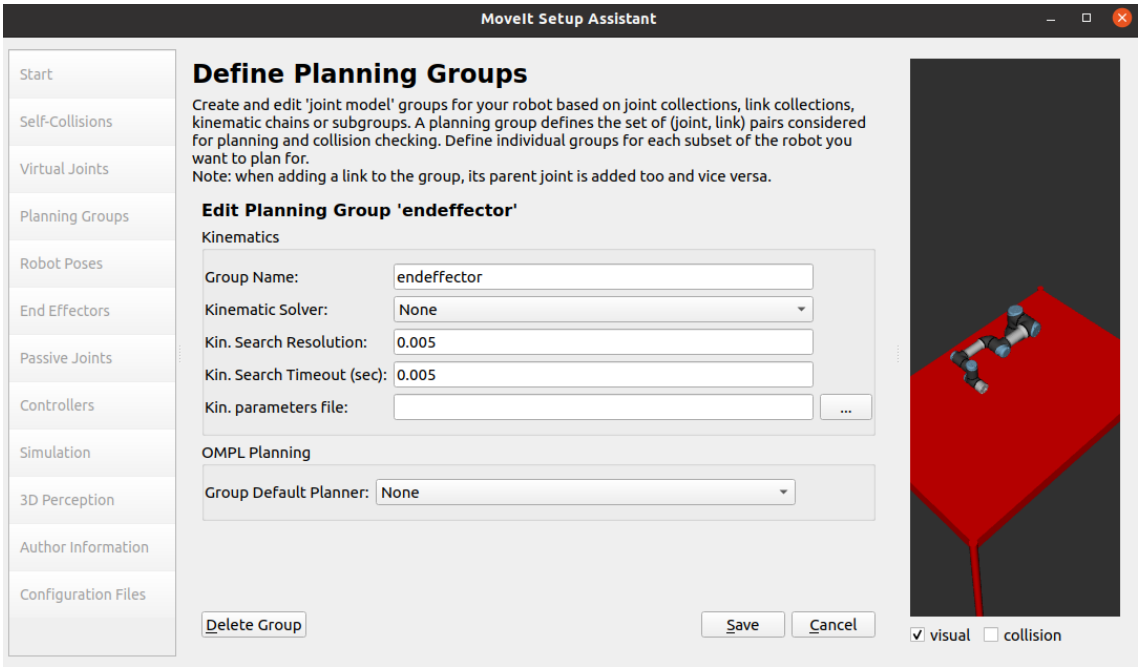


Figura 23: Configuración endeffector

El resultado tendría que ser:

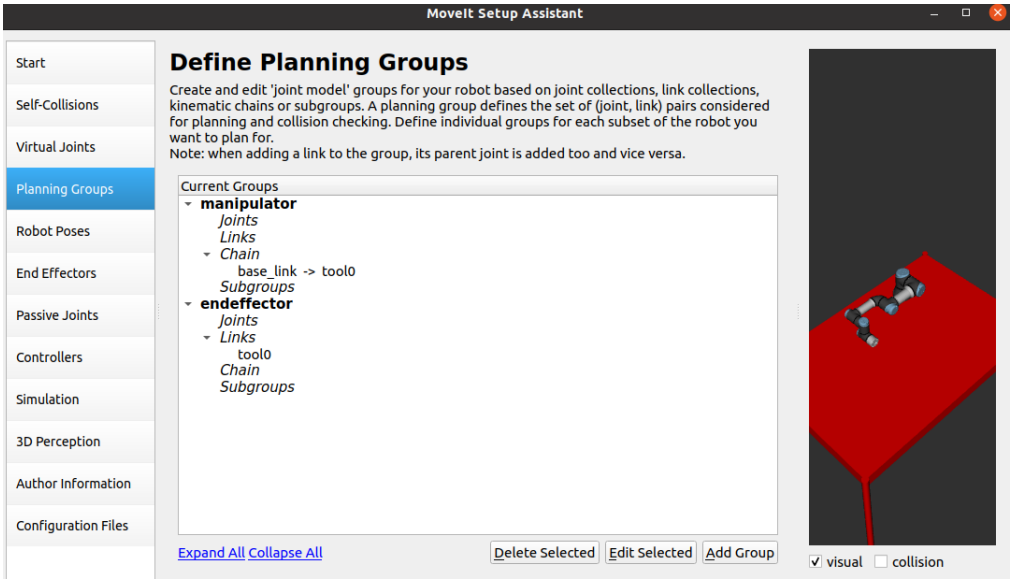


Figura 24: Resultado Planning Groups

4.7. ROBOT POSES

Después, la siguiente opción que da, es programarle unas posiciones predeterminadas a el robot. En este caso se ha guardado una posición que se llamará *'up'*, la cual corresponde con la posición del robot en el estado inicial.

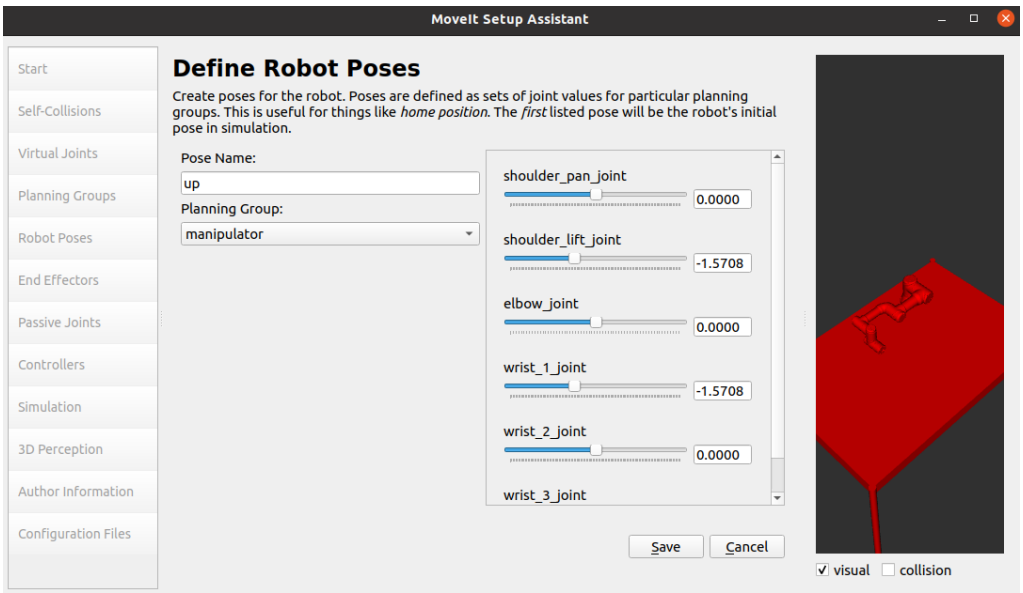


Figura 25: Robot poses

4.8. END EFFECTOR

El siguiente paso es programar el End Effector de nuestro robot. Como se ha comentado anteriormente, a la hora de desarrollar este manual, el robot no disponía de ninguna herramienta acoplada en él, con lo que el End Effector corresponderá al link tool0.

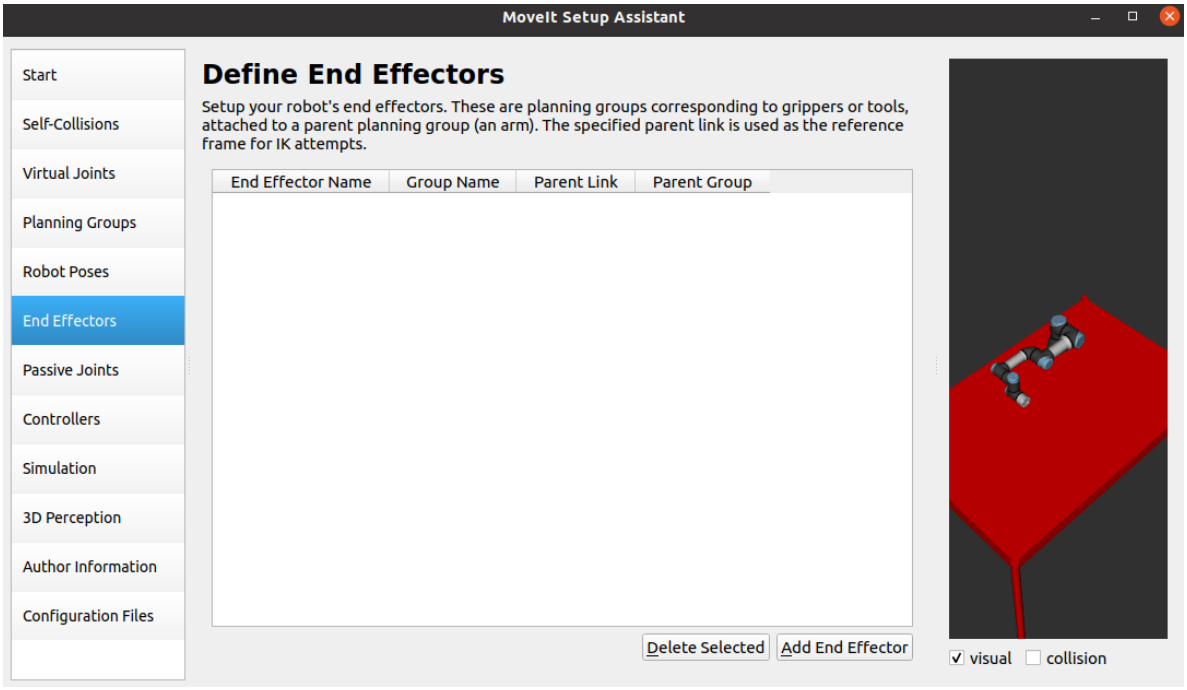


Figura 26: Interfaz Endeffectors

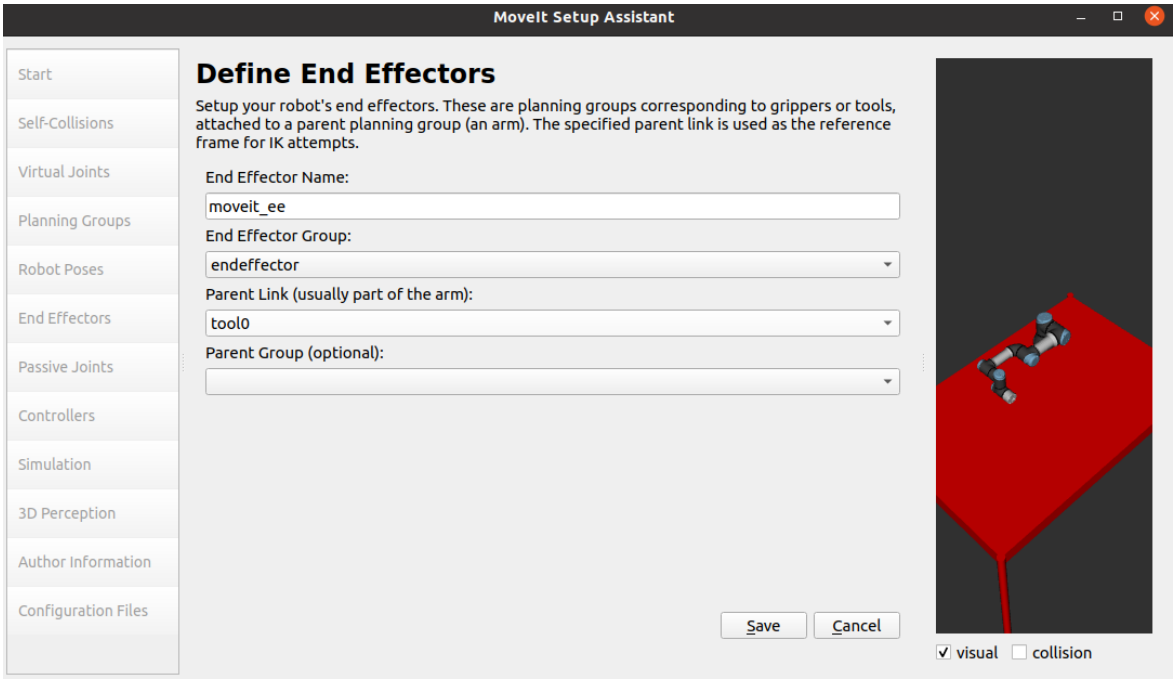


Figura 27: Configuración EndEffector

Dejo aquí el enlace a la página de moveit_tutorial para que veas como se tendría que programar esta parte anterior si se tuviera herramienta. También un enlace de un video a Youtube donde se muestra un posible ejemplo de cómo implementar este apartado.

https://ros-planning.github.io/moveit_tutorials/doc/setup_assistant/setup_assistant_tutorial.html

https://www.youtube.com/watch?v=BxCik8OI1Fw (Última visita: 07/06/2022)

4.9. PASSIVE JOINTS

En el siguiente paso de Passive joints no hace falta hacer nada. Se puede pasar al siguiente apartado.

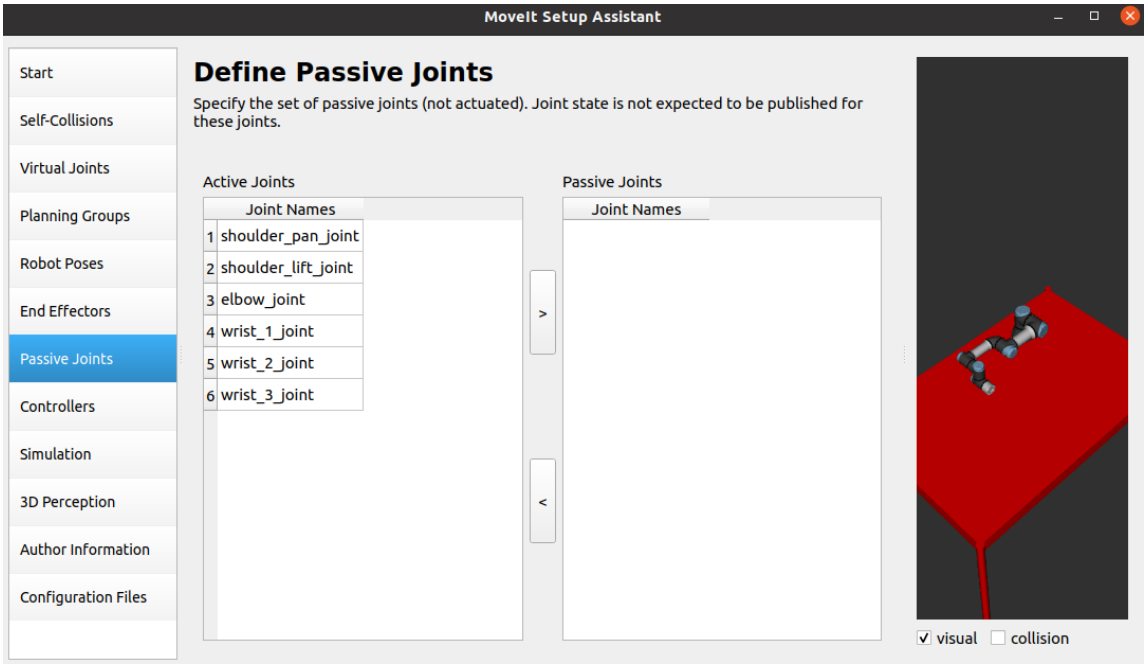


Figura 28: Passive Joints

4.10. CONTROLLERS

El siguiente apartado consiste en programar el/los controladores con los que el *'Controller manager'* va a operar.

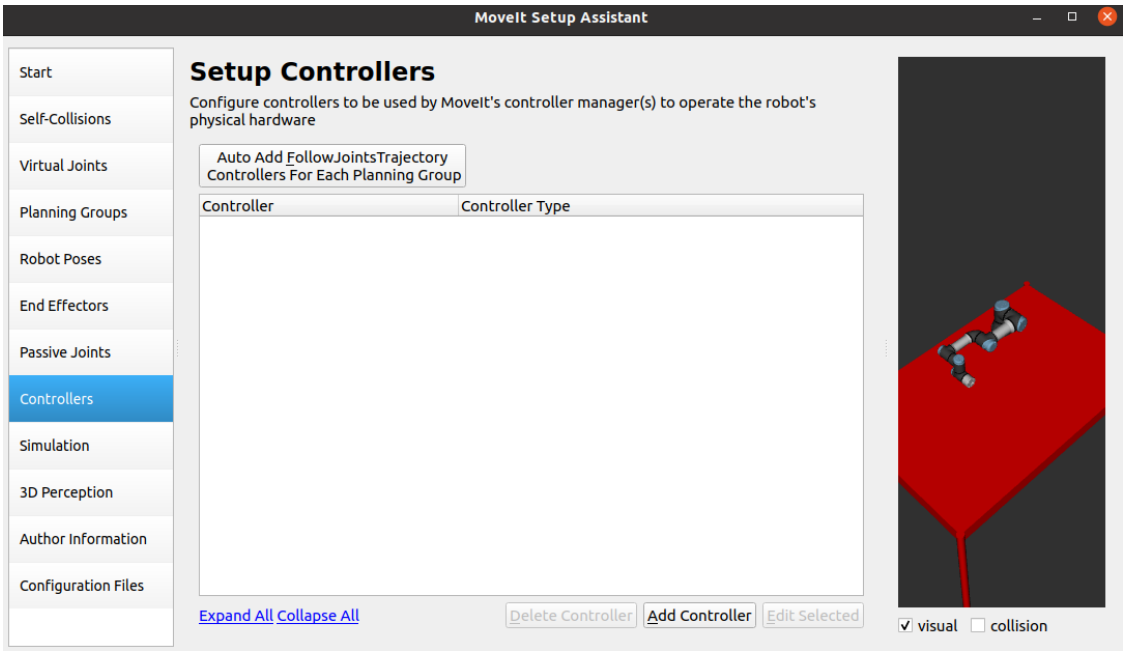


Figura 29: Interfaz Controladores

Dándole a Add Controller situado debajo de la pantalla, se podrá añadir el controlador.

El nombre del controlador es muy importante que se llame *'pos_joint_traj_controller'*. Esto es porque este controlador es el que se dispara cuando arrancamos el Gazebo del robot, con lo que de esta forma se tendría un controlador común cuando se esté en modo simulación y cuando se esté controlando nuestro robot. El tipo de controlador tendrá que ser el FollowJointTrayectory. Antes de darle a Save, tendremos que darle los *'joint'* que actúan con ese controlador, con lo que pinchamos en Add Individual Joint y se añadirán las seis juntas del robot.

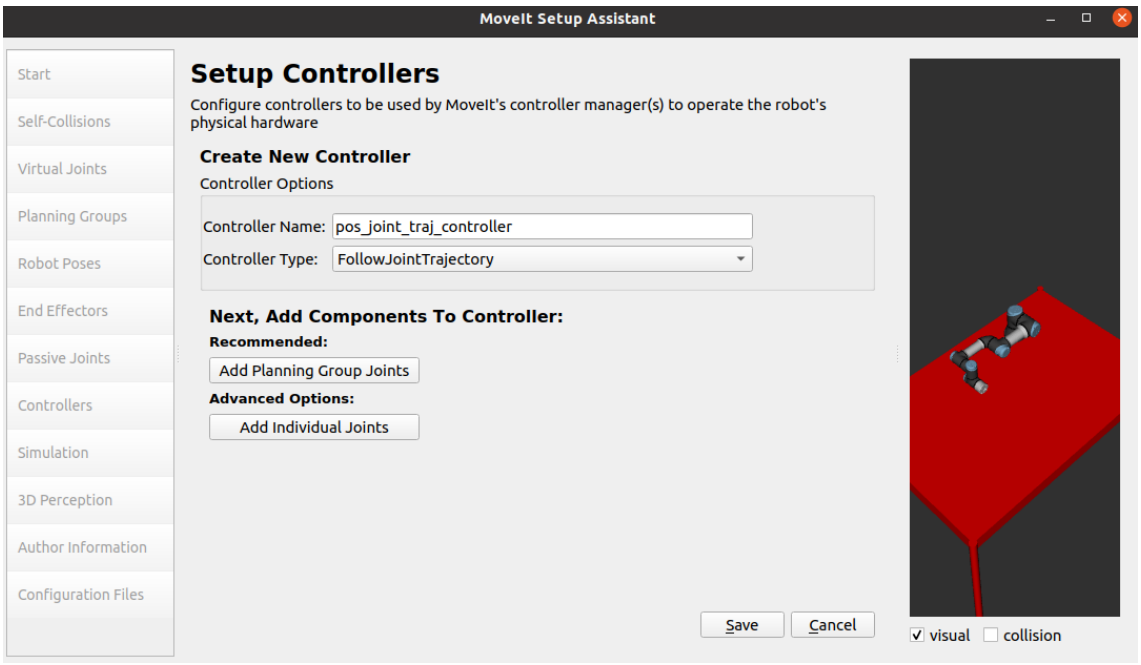


Figura 30: Controlador pos_joint_traj_controller

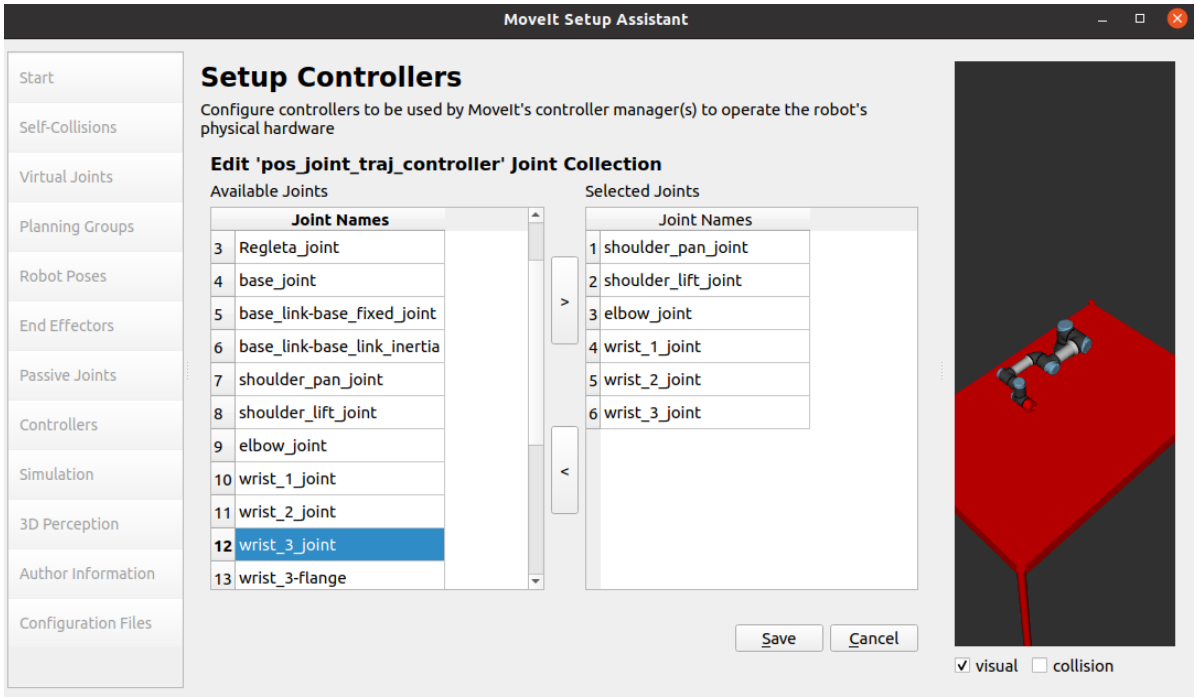


Figura 31: Joints del controlador

Tendría que quedar la configuración del controlador de la siguiente forma:

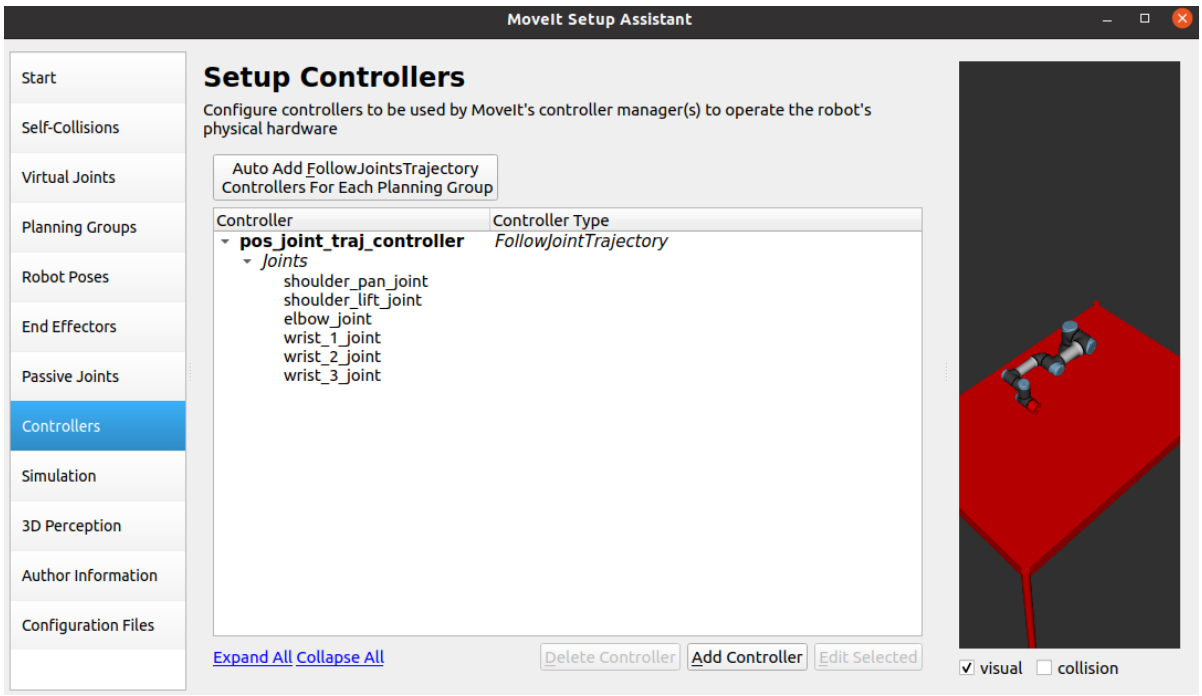


Figura 32: Resultado Controladores

4.11. SIMULATION, 3D Y AUTOR

Los siguientes dos apartados de Simulation y 3D Perception se pueden saltar y en el apartado de Author Information, el cual es un apartado obligatorio, invéntate el nombre y mail (no influye en nada).

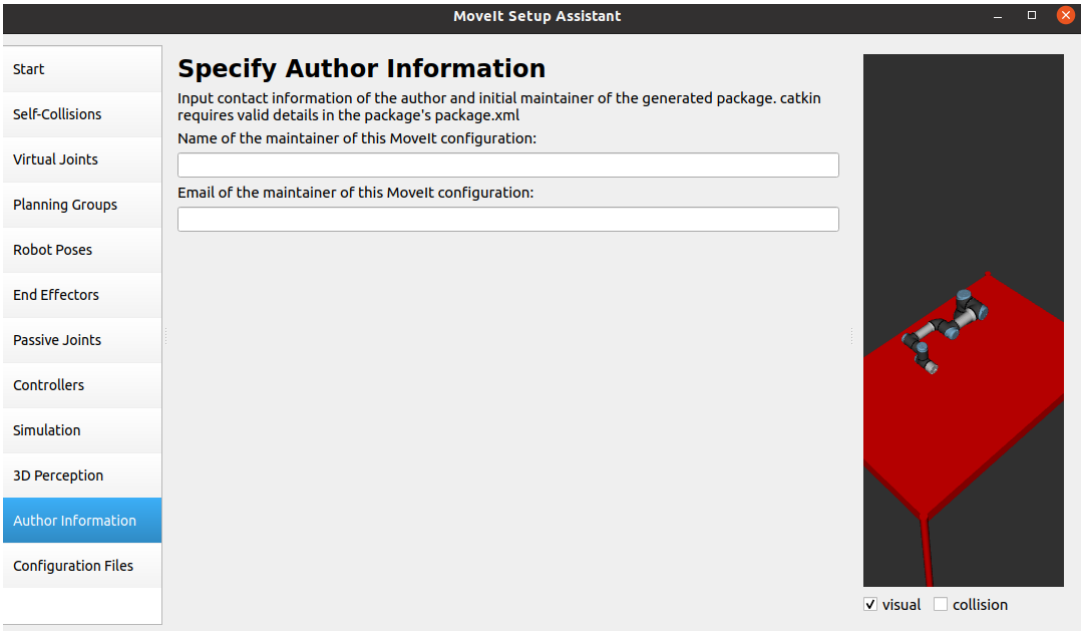


Figura 33: Author Information

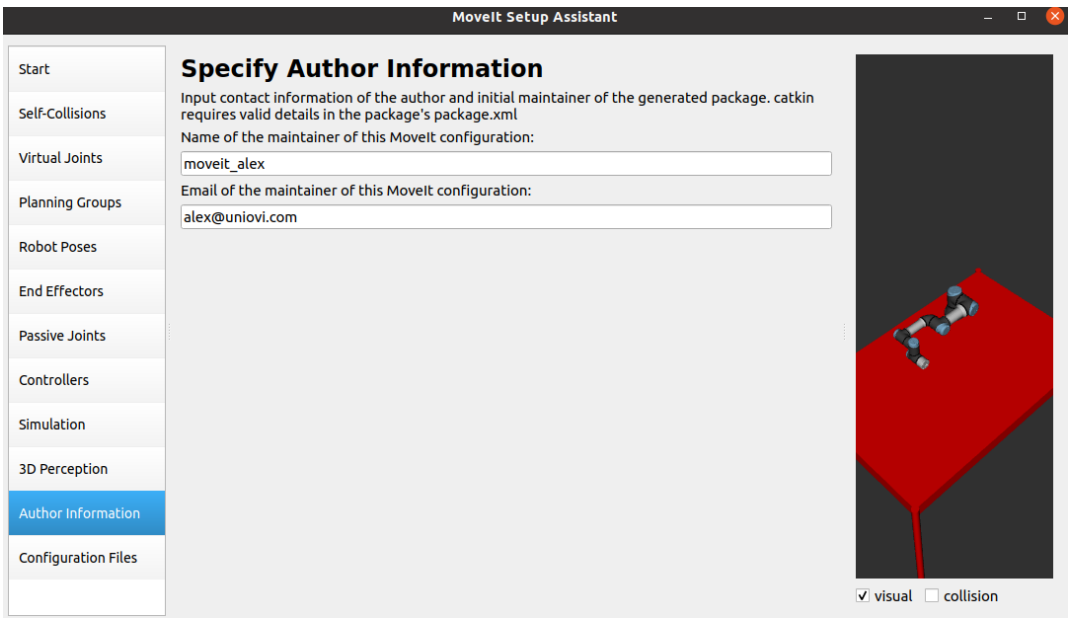


Figura 34: Autor y email

4.12. CONFIGURATION FILES

Una vez hecho ya solo queda elegir el directorio en donde se quiere guardar el paquete (en este caso en /home/Alejandro/catkin_ws/src), el nombre con el que lo queremos guardar (en este caso my_moveit_config) y por último darle a generar el paquete Moveit!

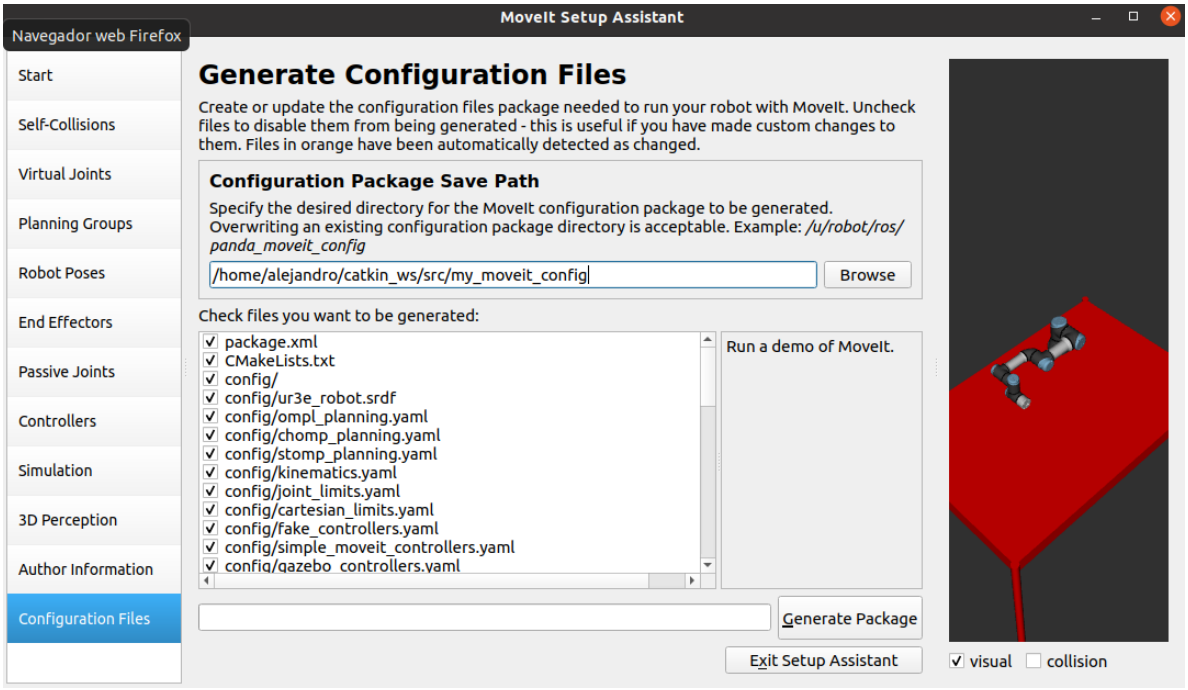


Figura 35: Generar paquete moveit

Debería salir que se ha completado la generación del paquete Moveit correctamente.

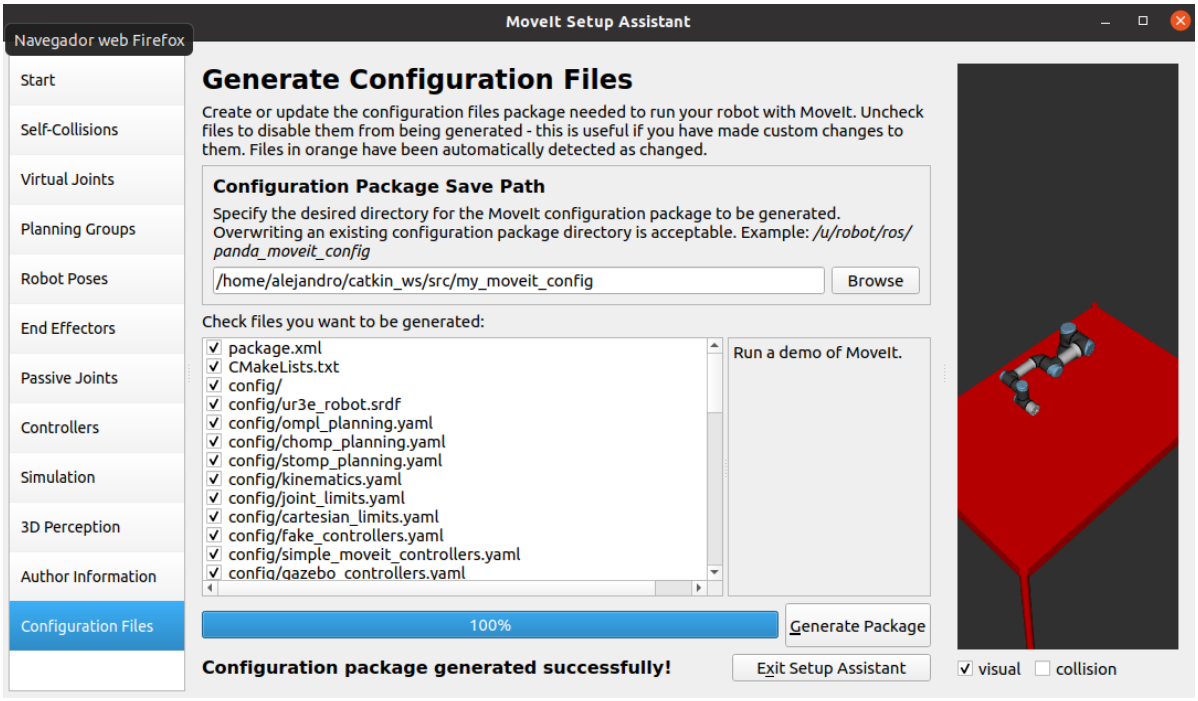


Figura 36: Paquete generado

Si no se produjo error hacemos click en Exit Setup Assistant y ya se habría creado nuestro paquete Moveit.

5. FUNCIONAMIENTO EN MODO SIMULACIÓN

Con el paquete Moveit creado, el siguiente paso será realizar una demostración del funcionamiento del robot en nuestro lugar de trabajo creado en Gazebo, usando el archivo de RVIZ creado en nuestro paquete Moveit. RVIZ permitirá configurar escenas en las que el robot trabajará, generar planos, visualizar la trayectoria planeada e interactuar directamente con el robot visualizado.

5.1. MODIFICACIONES AL PAQUETE MOVEIT CREADO

Pero primero, se tienen que hacer unas pequeñas modificaciones a el paquete Moveit creado para declarar si nos encontramos en el modo de simulación o en el modo de control remoto.

Para declarar como va a trabajar el robot, se va a crear un *'.launch'* dentro del paquete Moveit creado. Para ello, como en ocasiones anteriores, hay que ejecutar el comando *'gedit'* en el terminal y escribir el siguiente código:

Abrir — ur3e_moveit_planning_execution.launch — ~/catkin_ws/src/my_moveit_config/launch — Guardar

```
<launch>
  <arg name="sim" default="false" />
  <arg name="debug" default="false" />

  <!-- Remap follow_joint_trajectory -->
  <remap if="$(arg sim)" from="/pos_joint_traj_controller/follow_joint_trajectory" to="/-
pos_joint_traj_controller/follow_joint_trajectory"/>

  <!-- Launch moveit -->
  <include file="$(find my_moveit_config)/launch/move_group.launch">
    <arg name="debug" default="$(arg debug)" />
  </include>
</launch>
```

Figura 37: ur3e_planning_execution.launch

El programa anterior se ha guardado como *'ur3e_moveit_planning_execution.launch'*, el cual, por defecto, si se hace un *'roslaunch'* de este archivo se lanza en modo de control remoto (modo simulación por defecto = *false).* También se lanza el archivo move_group, el cual proporciona una funcionalidad fácil de usar para la mayoría de las operaciones que el robot puede querer llevar a cabo.

Otra modificación que hay que hacer, es en el archivo *'moveit_rviz.launch'*. La modificación que hay que hacerle, es hacer que Rviz lance por defecto el '*moveit_rviz*'. El cambio es el siguiente:

Abrir — moveit_rviz.launch — ~/catkin_ws/src/my_moveit_config/launch — Guardar

```
<launch>

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

  <arg name="rviz_config" default="$(find my_moveit_config)/launch/moveit.rviz" />
  <arg     if="$(eval rviz_config=='')" name="command_args" value="" />
  <arg unless="$(eval rviz_config=='')" name="command_args" value="-d $(arg rviz_config)" />

  <node name="$(anon rviz)" launch-prefix="$(arg launch_prefix)" pkg="rviz" type="rviz" respawn="false"
        args="$(arg command_args)" output="screen">
  </node>

</launch>
```

Figura 38: Modificación en moveit_rviz.launch

Una vez guardado este cambio ya se puede observar cómo va a trabajar el robot UR3e en modo simulación.

5.2. TRABAJAR EN MODO SIMULACIÓN

Para empezar a trabajar en modo simulación, se empezará por ejecutar el Gazebo creado anteriormente ya que es donde se verá el comportamiento simulado del robot cuando le planifiquemos una trayectoria. Para ello se ejecutará el comando:

```
alejandro@alejandro-VirtualBox:~/catkin_ws$ roslaunch ur_gazebo ur3e_bringup.lau
nch
```

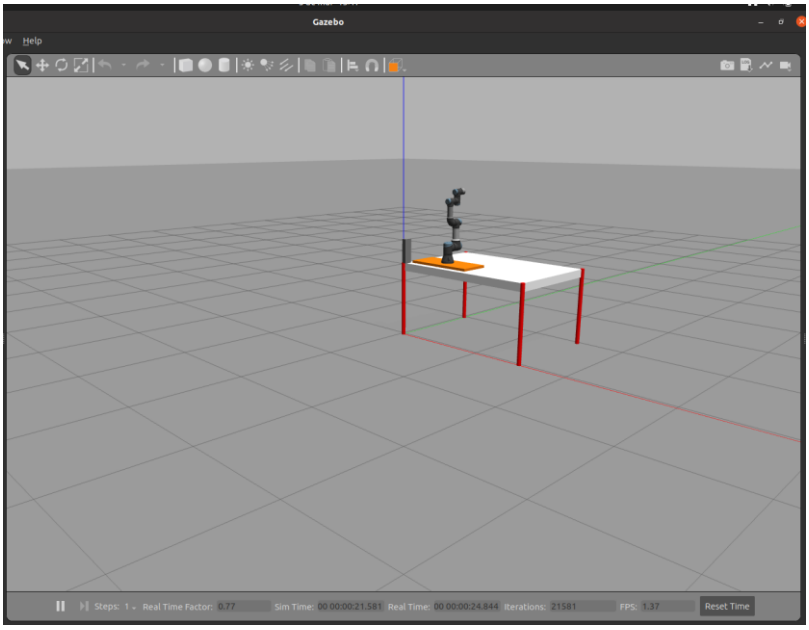


Figura 39: Gazebo Ur3e

(NOTA: Acordarse de que el Gazebo se lanza pausado, con lo que hay que darle al Play)

El siguiente paso es lanzar el archivo *'ur3e_planning_execution.launch'* (Apartado5.1) en modo simulación. Para ello, se ejecutará el siguiente comando en una pestaña nueva del terminal:

```
alejandro@alejandro-VirtualBox:~/catkin_ws$ roslaunch my_moveit_config ur3e_move
it_planning_execution.launch sim:=true
```

Como se puede comprobar, escribiendo el código *'sim:=true'* se estará trabajando en modo simulación.

A partir de aquí ya solo quedaría decidir qué programa o archivo se va a utilizar para crear las trayectorias que queremos que recorra el robot. En este manual se van a presentar dos opciones: usando la interfaz RVIZ o mediante un archivo de Python.

5.2.1. USANDO UN ARCHIVO DE PYTHON

Una de las formas de controlar el robot es mediante un archivo de Python. El archivo de Python creado para este manual es:

```
1 #!/usr/bin/env python
2 import sys
3 import rospy
4 import actionlib
5 from control_msgs.msg import FollowJointTrajectoryAction, FollowJointTrajectoryGoal
6 from trajectory_msgs.msg import JointTrajectoryPoint
7 from controller_manager_msgs.srv import SwitchControllerRequest, SwitchController
8 from controller_manager_msgs.srv import LoadControllerRequest, LoadController
9 import geometry_msgs.msg as geometry_msgs
10 from cartesian_control_msgs.msg import (
11     FollowCartesianTrajectoryAction,
12     FollowCartesianTrajectoryGoal,
13     CartesianTrajectoryPoint,
14 )
```

En estas primeras 12 líneas de código importo todas las librerías y funciones necesarias para el control del robot.

```
15 if sys.version_info[0] < 3:
16     input = raw_input
```

Estas líneas de código son para hacer compatible el archivo tanto con Python1 como con Python3.

```
17 JOINT_NAMES = ["shoulder_pan_joint","shoulder_lift_joint","elbow_joint","wrist_1_joint",
18 "wrist_2_joint","wrist_3_joint",]
19 JOINT_TRAJECTORY_CONTROLLERS = ["scaled_pos_joint_traj_controller",
20     "scaled_vel_joint_traj_controller",
21     "pos_joint_traj_controller",
22     "vel_joint_traj_controller",
23     "forward_joint_traj_controller",]
24 CONFLICTING_CONTROLLERS = ["joint_group_vel_controller", "twist_controller"]
```

En estas líneas de código (17-24) declaramos una lista con los nombres de las juntas del robot, una lista con los controladores disponibles y otra lista con los controladores conflictivos que pueden generar movimientos indeseados en el robot.

```
25 class TrajectoryClient:
27
28     def __init__(self):
29         rospy.init_node("test_move")
30
31         timeout = rospy.Duration(8)
32         self.switch_srv = rospy.ServiceProxy(
33             "controller_manager/switch_controller", SwitchController)
34
35
36         self.load_srv = rospy.ServiceProxy("controller_manager/load_controller", LoadController)
37
38
39         try:
40             self.switch_srv.wait_for_service(timeout.to_sec())
41
42         except rospy.exceptions.ROSException as err:
43             rospy.logerr("Could not reach controller switch service. Msg: {}".format(err))
44             sys.exit(-1)
45
46         self.joint_trajectory_controller = JOINT_TRAJECTORY_CONTROLLERS[0]
```

En estas líneas de código (25-46) se crea una clase Trajectory Client y una función donde inicializamos el nodo ‘test_move’ y se le da una duración donde se espera que se inicie. También en la línea 32 se activa el cliente que permite cambiar de controlador y se asegura que el tráfico de información esté bien conectado. En la línea 36 se activa el cliente que permite cargar el controlador mánager. Finalmente se lanza el servicio y por defecto seleccionamos el primer controlador declarado en la lista de controladores.

```
47     def send_joint_trajectory(self):
48
49
50
51         self.switch_controller(self.joint_trajectory_controller)
52         trajectory_client = actionlib.SimpleActionClient(
53             "{}/follow_joint_trajectory".format(self.joint_trajectory_controller),
54             FollowJointTrajectoryAction,
55         )
56
57
58         goal = FollowJointTrajectoryGoal()
59         goal.trajectory.joint_names = JOINT_NAMES
60
```

```
61
62
63          position_list = [[0, -1.57, -1.57, 0, 0, 0]]
64          position_list.append([0, -1.57, -1.57, 0, 0, 0])
65          position_list.append([-1.57, -2.36, -1.71, -0.61, 1.59, 0])
66          position_list.append([-1.57, -2.36, -1.71, -0.61, 1.59, 0])
67          position_list.append([-0.5, -1.57, -1.2, 0, 0, 0])
68          position_list.append([-0.5, -1.57, -1.2, 0, 0, 0])
69          position_list.append([0, -1.57, 0, -1.57, 0, 0])
70          duration_list = [5.0, 6.0, 10.0, 11.0, 14.0,15.0,18.0]
71          for i, position in enumerate(position_list):
72              point = JointTrajectoryPoint()
73              point.positions = position
74              point.time_from_start = rospy.Duration(duration_list[i])
75              goal.trajectory.points.append(point)
76
77          self.ask_confirmation(position_list)
78          rospy.loginfo("Executing trajectory using the {}".format(self.joint_trajectory_controller))
79
80          trajectory_client.send_goal(goal)
81          trajectory_client.wait_for_result()
82
83          result = trajectory_client.get_result()
84          rospy.loginfo("Trajectory execution finished in state {}".format(result.error_code))
```

En estas líneas de código (47-84) se crea una función donde se elabora una trayectoria y se envía a través del controlador que se está utilizando. Primero se cambia al controlador que seleccionaste al inicio del programa. Después se crea una lista vacía donde se añadirán las posiciones a las que se quieran llegar. Una posición es una lista formada por el valor en radianes que debería tener cada junta para llegar a esa posición.

Una vez confirmamos la lista con las posiciones que se quieren, la recorremos y enviamos cada posición a la función trayectory_client.send_goal().

```
85 def ask_confirmation(self, waypoint_list):
86
87
88          rospy.logwarn("The robot will move to the following waypoints: \n{}".format(waypoint_list))
89          confirmed = False
90          valid = False
91          while not valid:
92              input_str = input(
93                  "Please confirm that the robot path is clear of obstacles.\n"
94                  "Keep the EM-Stop available at all times. You are executing\n"
```

```
95                "the motion at your own risk. Please type 'y' to proceed or 'n' to abort: "
96            )
97            valid = input_str in ["y", "n"]
98            if not valid:
99                rospy.loginfo("Please confirm by entering 'y' or abort by entering 'n'")
100           else:
101               confirmed = input_str == "y"
102       if not confirmed:
103           rospy.loginfo("Exiting as requested by user.")
104           sys.exit(0)
```

Estas líneas de código (85-104) corresponden a una función donde simplemente se está pidiendo una confirmación de que la trayectoria creada en la función anterior es la que se desea realizar.

```
105   def choose_controller(self):
106
107       rospy.loginfo("Available trajectory controllers:")
108       for (index, name) in enumerate(JOINT_TRAJECTORY_CONTROLLERS):
109           rospy.loginfo("{} (joint-based): {}".format(index, name))
110       choice = -1
111       while choice < 0:
112           input_str = input(
113               "Please choose a controller by entering its number (Enter '0' if "
114               "you are unsure / don't care): "
115           )
116           try:
117               choice = int(input_str)
118               if choice < 0 or choice >= len(JOINT_TRAJECTORY_CONTROLLERS):
119                   rospy.loginfo(
120                       "{} not inside the list of options. "
121                       "Please enter a valid index from the list above.".format(choice)
122                   )
123                   choice = -1
124           except ValueError:
125               rospy.loginfo("Input is not a valid number. Please try again.")
126       if choice < len(JOINT_TRAJECTORY_CONTROLLERS):
127           self.joint_trajectory_controller = JOINT_TRAJECTORY_CONTROLLERS[choice]
128           return "joint_based"
129
130       return "Outside the limits"
```

Estas líneas de código (105-130) corresponden a una función donde se va a escoger el controlador que se quiere utilizar para realizar la trayectoria. Hay que recordar que cuando se esté en modo Simulación el único controlador que se debe utilizar es el

pos_joint_traj_controller, mientras que en el modo de Control Remoto se puede emplear cualquier controlador.

```
131 def switch_controller(self, target_controller):
132 
133         other_controllers = (
134             JOINT_TRAJECTORY_CONTROLLERS
135             + CONFLICTING_CONTROLLERS
136         )
137 
138         other_controllers.remove(target_controller)
140         srv = LoadControllerRequest()
141         srv.name = target_controller
142         self.load_srv(srv)
143 
144         srv = SwitchControllerRequest()
145         srv.stop_controllers = other_controllers
146         srv.start_controllers = [target_controller]
147         srv.strictness = SwitchControllerRequest.BEST_EFFORT
148         self.switch_srv(srv)
```

En estas líneas de código (121-148) se implementa la función que cambia de controlador. A la función se le pasa el controlador que se quiere emplear y se cambia a ese controlador. Esta es la última función dentro de la clase TrayectoryClient.

```
149 if __name__ == "__main__":
150     client = TrajectoryClient()
151 
152 
153 
154     trajectory_type = client.choose_controller()
155     if trajectory_type == "joint_based":
156         client.send_joint_trajectory()
157         client.switch_controller(JOINT_TRAJECTORY_CONTROLLERS[2])
158     else:
159         raise ValueError(
160             "I only understand types 'joint_based' and 'cartesian', but got '{}'".format(
161                 trajectory_type
162             )
163         )
```

Estas últimas líneas de código (149-163), corresponden con el main del programa, donde lo que se hace es llamar a la clase TrayectoryClient anteriormente programada, escoger el controlador que se quiere utilizar y si el controlador es ‘joint_based’ hacer la trayectoria que le hayamos programado. Al final se vuelve a cambiar al controlador

'pos_joint_traj_controller' que es el controlador común con Gazebo.

(Nota: este código se va a emplear también para el modo de Control Remoto)

Acordarse de usar el comando '*chmod u+x nombre_archivo.py*' para darle al archivo permisos de lectura y ejecución. Para ejecutar el archivo Python recordar usar el comando rosrun.

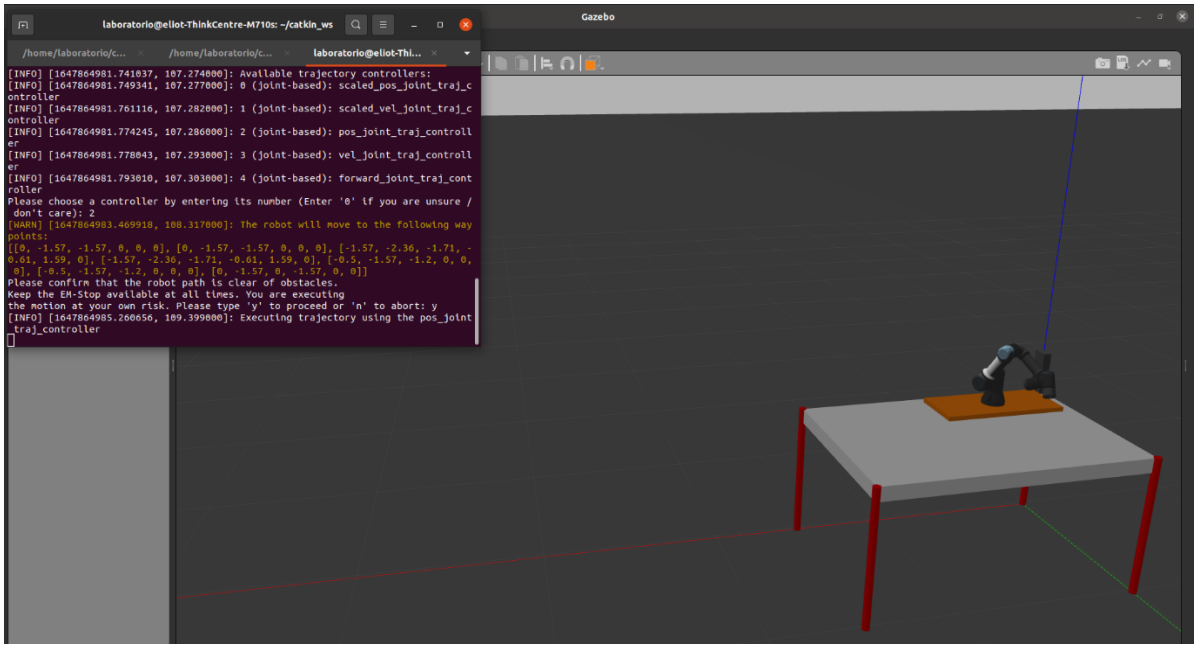


Figura 40: Simulación en Gazebo mediante archivo Python

5.2.2. USANDO RVIZ

RVIZ es una herramienta de visualización 2D y 3D para aplicaciones de ROS, proporciona una vista del modelo de robot, planifica trayectorias, captura la información de los sensores en el sistema interno del robot y reproduce los datos capturados (Gazebo por ejemplo). Puede mostrar datos de cámara, láseres y dispositivos 3D y 2D, como imágenes y nubes de puntos.

Para lanzar RVIZ, en una pestaña nueva del terminal se ejecutará el comando:

```
alejandro@alejandro-VirtualBox:~/catkin_ws$ roslaunch my_moveit_config moveit_rviz.launch config:=true
```

Una vez ejecutado se debería abrir la siguiente interfaz:

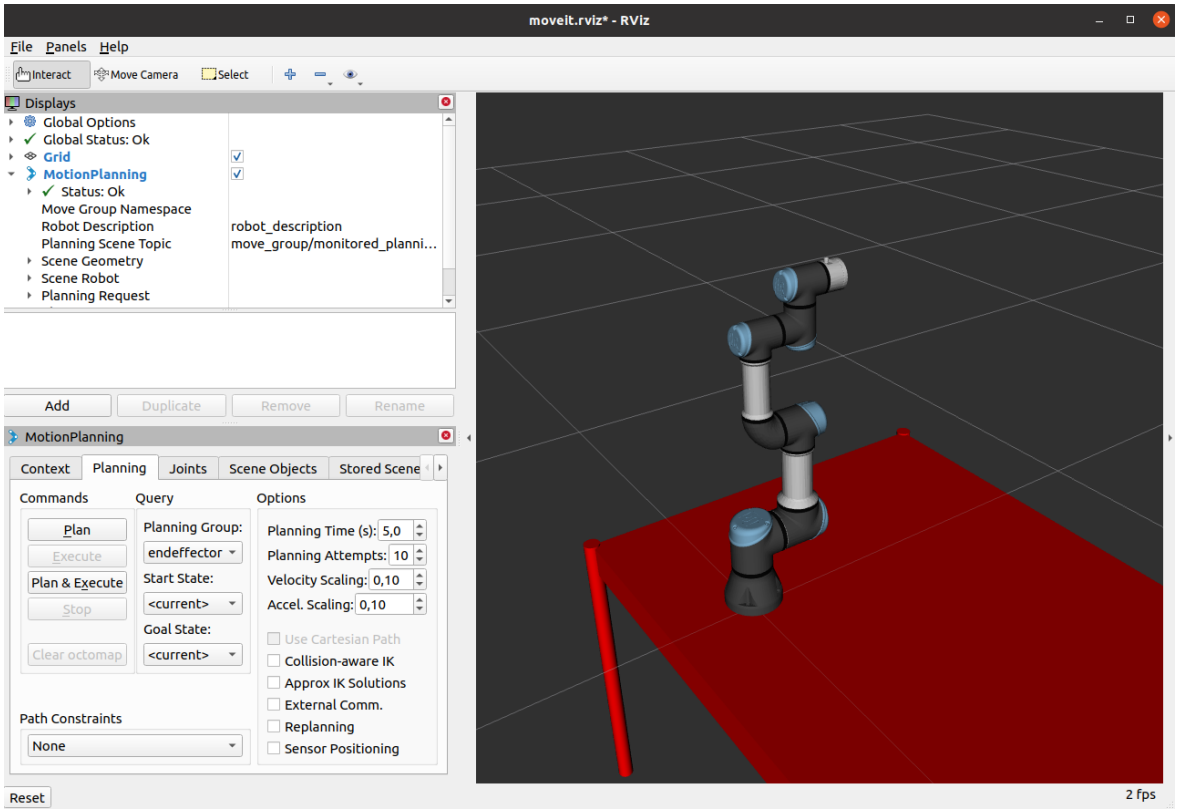


Figura 41: RVIZ Ur3e

A partir de aquí, ya se puede empezar a planificar trayectorias, cambiando en la pestaña de debajo de Planning, el Planning Group de endeffector (que es el Planning Group que realizaría los movimientos de la herramienta) por manipulator (que es el Planning Group que realizaría los movimientos del robot).

Una manera, la cual es más eficiente a la hora de planificar trayectorias y ver que la trayectoria que se ha planificado no interfiere con los objetos del entorno, es manipulando el robot en la pestaña de Joints. En la siguiente figura se muestra una posible trayectoria:

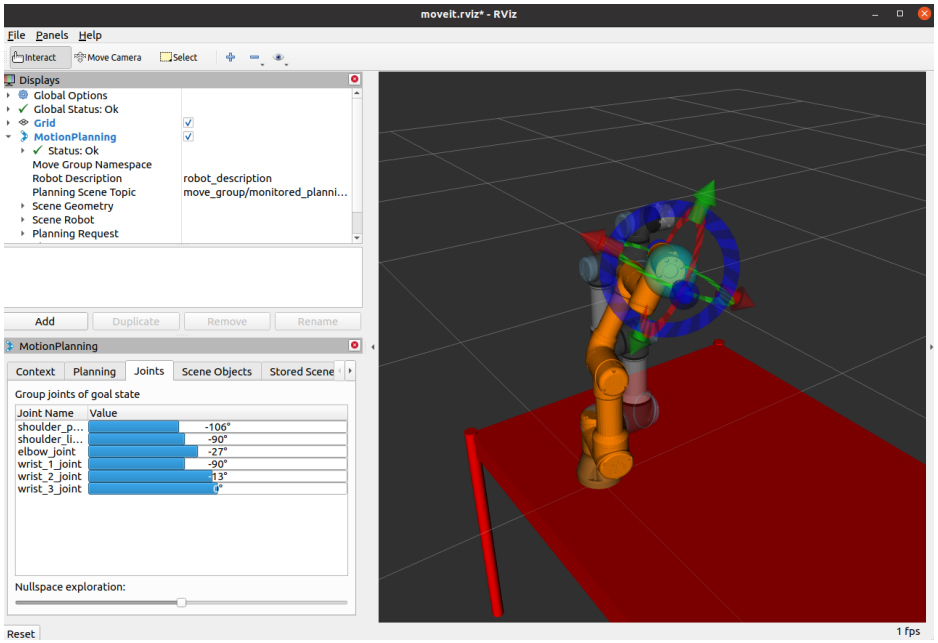
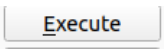


Figura 42: Ur3e controlado por los Joints

Una vez se haya decidido el punto al que queremos que se mueva nuestro robot, volvemos a la pestaña de Planning y se le da a *'Plan'*. Si hay algún problema en el hueco debajo del botón de Stop deberá aparecer un *'Failed'*. Sino es así, significará que la trayectoria planificada es viable, con lo que se podrá darle al botón de *'Execute'*.



Una vez dado en ejecutar, se puede comprobar que el robot cambia a esa posición tanto en RVIZ como en Gazebo.

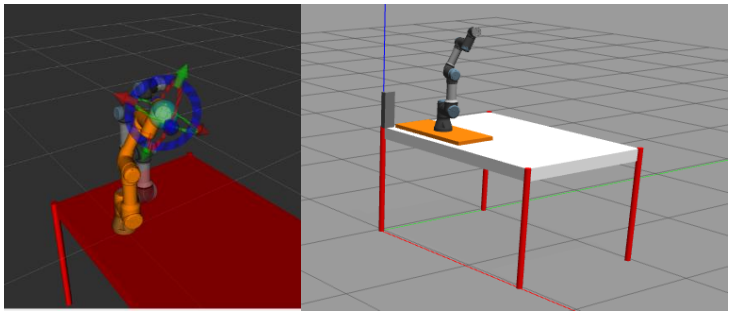


Figura 43: Ur3e controlado mediante Rviz y representado en Gazebo

6. FUNCIONAMIENTO EN MODO DE CONTROL REMOTO

En este apartado se va a tratar el cómo establecer conexión con nuestro robot y el cómo controlarlo de manera remota usando nuestro PC. Para este apartado es importante que la conexión sea mediante cable Ethernet y no mediante una conexión inalámbrica de Wifi para así tener mejor conexión y evitar caídas que podrían hacer dejar de funcionar a el robot. Otro punto importante, como se comentó al principio del manual, es que no se puede hacer este apartado del proyecto desde una máquina virtual, debido a problemas con la conexión entre la máquina virtual, el PC y el robot.

6.1. ESTABLECER CONEXIÓN CON EL ROBOT

6.1.1. PAQUETES INSTALADOS EN EL PANEL DE CONTROL

Primero de todo, se hay que asegurar que en el panel de control del robot se hayan instalado y comprobado que estén activos los paquetes de External Control, Remote TCP & Toolpath y rs485. Puedes comprobarlo en Ajustes/Sistema/URCaps.

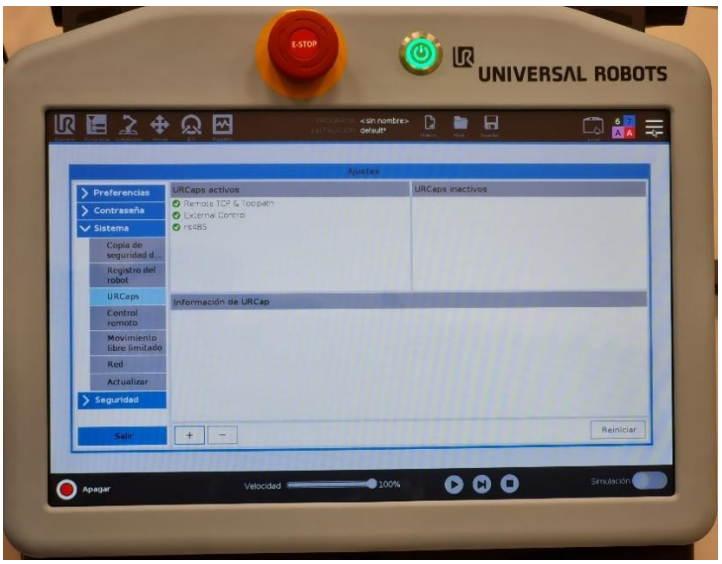


Figura 43: Paquetes instalados en el panel de control

6.1.2. DIRECCIONES IP'S NECESARIAS

Para establecer conexión con el robot, se va a necesitar saber tanto la dirección IP del robot como la dirección IP del ordenador que se está utilizando para controlar el robot. La dirección del robot la podemos encontrar en Ajustes/Sistema/Red.

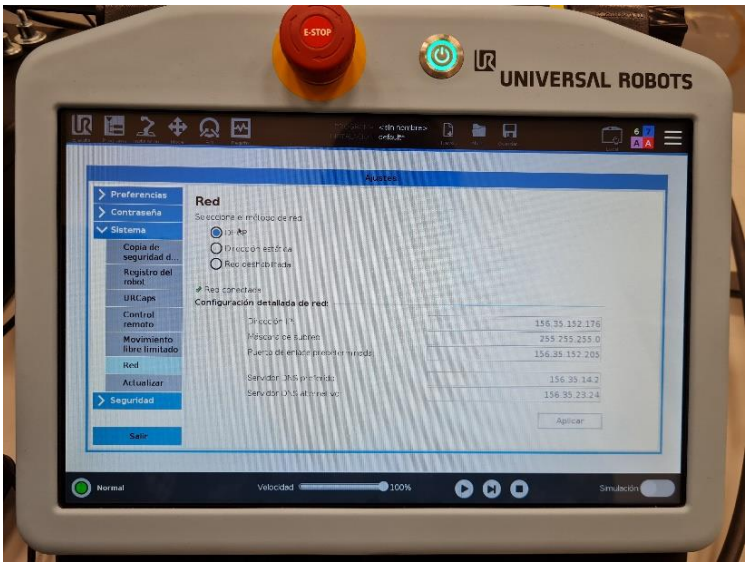


Figura 44: Dirección IP del robot (robot_ip)

Como se puede comprobar en la Figura la IP del robot es 156.35.152.176 .

Dato: En la IP anterior, '156.35.' es la secuencia numérica perteneciente a la subred de la Universidad de Oviedo. '152.' es la secuencia numérica perteneciente a la dirección del área de Ingeniería de Sistemas y Automática. '176' es la secuencia numérica perteneciente al dispositivo desde el que te conectas.

Para conocer la IP de tu PC, simplemente se puede buscar por internet una página que te la muestre. En este caso por ejemplo, la IP del PC desde donde se desarrolla el proyecto es 156.35.152.183 .

6.1.3. CALIBRACIÓN DEL ROBOT

El siguiente paso es generar un archivo *'.yaml'* que contenga el cómo está calibrado inicialmente el robot. Cada robot UR viene calibrado de la fábrica. Para hacer uso de esto también en ROS, primero debe extraer la información de calibración del robot. Para extraer como está calibrado el robot hay que ejecutar el siguiente comando:

```
$ roslaunch ur_calibration calibration_correction.launch \
robot_ip:=<robot_ip> target_filename:="${HOME}/my_robot_calibration.yaml"
```

```
alejandro@alejandro-VirtualBox:~/catkin_ws$ roslaunch ur_calibration calibration_correction.launch
 robot_ip:=156.35.152.176 target_filename:="${HOME}/my_robot_calibration.yaml"
```

Este comando lo que hace es lanzar un programa *'calibration_correction.launch'*, el cual, pasándole la dirección IP del robot, crea un archivo *'my_robot_calibration.yaml'* en el escritorio {HOME} que contiene el cómo está calibrado el robot.

6.1.4. ARCHIVO LAUNCH PARA CONECTARSE AL ROBOT

En este paso se va a crear una carpeta donde se guardará un archivo *'.launch'*, similar a uno ya creado, que permitirá conectarnos al robot, cambiándole el cómo está calibrado el robot y cuál es su dirección IP.

Para crear esa carpeta se va a ejecutar los siguientes comandos:

```
# Replace your actual catkin_ws folder
$ cd <catkin_ws>/src
$ catkin_create_pkg example_organization_ur_launch ur_client_library \
-D "Package containing calibrations and launch files for our UR robots."
# Create a skeleton package
$ mkdir -p example_organization_ur_launch/etc
$ mkdir -p example_organization_ur_launch/launch
```

El primer comando nos sitúa en la carpeta *'src'*.

```
alejandro@alejandro-VirtualBox:~/catkin_ws$ catkin_create_pkg example_organization_ur_launch
 ur_client_library -D "Package containing calibrations and launch files for our UR robots."
```

Este comando crea una carpeta *'example_organization_ur_launch'* donde se guardará el archivo que lanza la comunicación con el robot y el archivo *'.yalm'* de cómo está calibrado nuestro robot que se creó anteriormente.

```
~/catkin_ws/src$ mkdir -p example_organization_ur_launch/etc
~/catkin_ws/src$ mkdir -p example_organization_ur_launch/launch
```

Estos dos comandos crean dos directorios. Uno el directorio etc, donde se guardará el archivo *'my_robot_calibration.yaml'* creado anteriormente, el cual contiene la calibración del robot. El otro será un directorio launch, donde se guardará el archivo *'.launch'* que lanzará la comunicación con el robot.

```
# Replace your actual catkin_ws folder
$ cd <catkin_ws>/src/example_organization_ur_launch/launch
$ roscp ur_robot_driver ur10_bringup.launch ex-ur10-1.launch
```

```
~/catkin_ws/src$ cd example_organization_ur_launch/launch/
```

```
alejandro@alejandro-VirtualBox:~/catkin_ws/src/example_organization_ur_launch/la
unch$ roscp ur_robot_driver ur3e_bringup.launch ex-ur3e-1.launch
```

Estos dos últimos comandos nos situará dentro de la carpeta launch donde se copiará el archivo *'ur3e_bringup.launch'*, situado en la carpeta *'ur_robot_driver'*, con el nombre de *'ex-ur3e-1.launch'*.

```
4   <arg name="robot_ip" default="156.35.152.176" doc="IP address by which the robot can be
  reached."/>
14   <arg name="kinematics_config" default="$(find example_organization_ur_launch)/etc/-
  my_robot_calibration.yaml" doc="Kinematics config file used for calibration correction. This
  will be used to verify the robot's calibration is matching the robot_description."/>
```

Dentro de ese archivo copiado, se tiene que poner en la línea 4, en la variable *'robot_ip'*, el valor por defecto de la IP del robot (Figura 44). También en la línea 14, hay que cambiar el valor por defecto de la variable *'kinematics_config'* por la dirección donde

está situado el archivo *'my_robot_calibration.yaml'* que contiene el cómo está calibrado el robot.

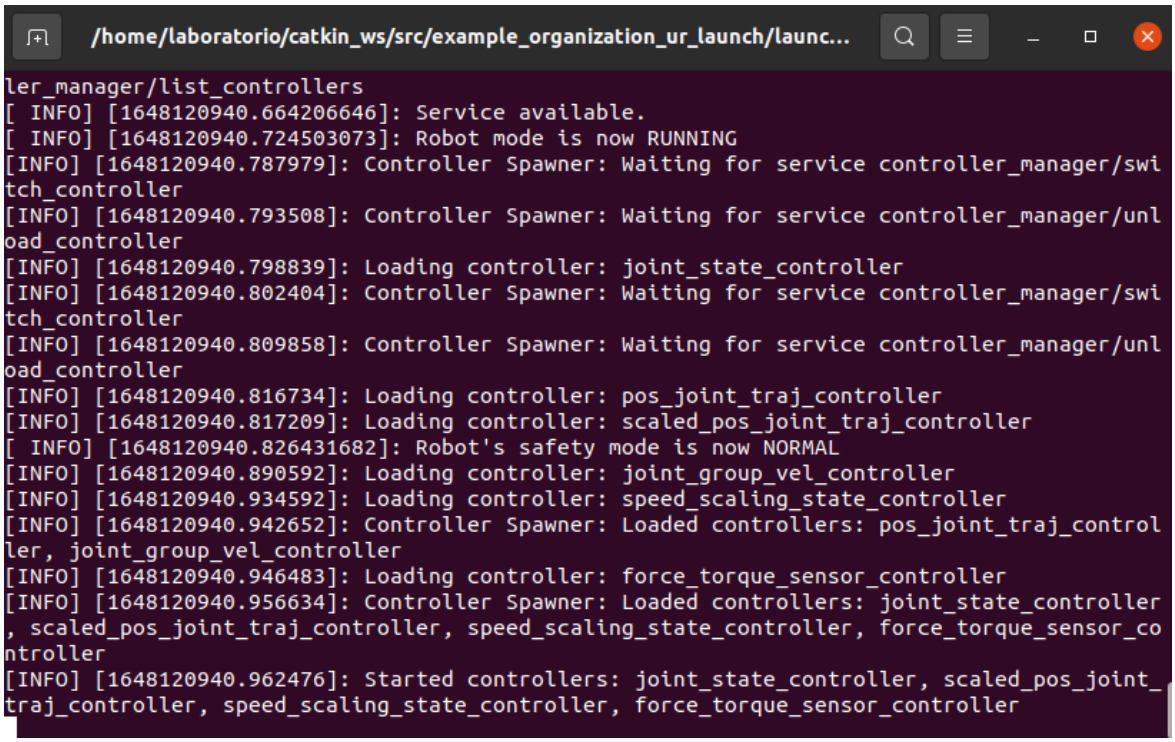
Una vez hechos estos cambios ya se puede empezar a trabajar en modo de control remoto.

6.2. TRABAJAR EN MODO CONTROL REMOTO

Para empezar a trabajar en el modo de control remoto, se empezará por ejecutar el archivo creado anteriormente *'ex-ur3e-1.launch'*, que lo que hará será conectarse a nuestro robot. Para ello se ejecutará el comando:

```
laboratorio@eliot-ThinkCentre-M710s:~/catkin_ws$ roslaunch example_organization_ur_launch
ex-ur3e-1.launch
```

Una vez ejecutado este comando, si no ha dado ningún error, solo warnings, al final tendría que aparecer los siguientes mensajes:



Como se puede apreciar en esta última imagen, aparece el mensaje 'Robot mode is now RUNNING'. Se observa que también se han cargado los controladores y que la seguridad del robot está en modo normal. Esto significa que el PC ya está conectado al robot. Ahora lo que faltaría sería hacer lo contrario, conectar el robot a el PC. Para esto primero hay que asegurarse de que en el panel de control esté instalado el paquete Remote Control (Apartado 6.1.1.). Si está ya instalado, en el panel de control, en la pestaña de Instalación, si se despliega abajo URCaps, tendría que aparecer la configuración de la variable External Control. En el hueco de Host IP habrá que poner la IP del ordenador al cual se va a conectar, el Custom port por defecto será el 50002 y el Host name será el nombre que se le quiera poner a la variable de External Control.

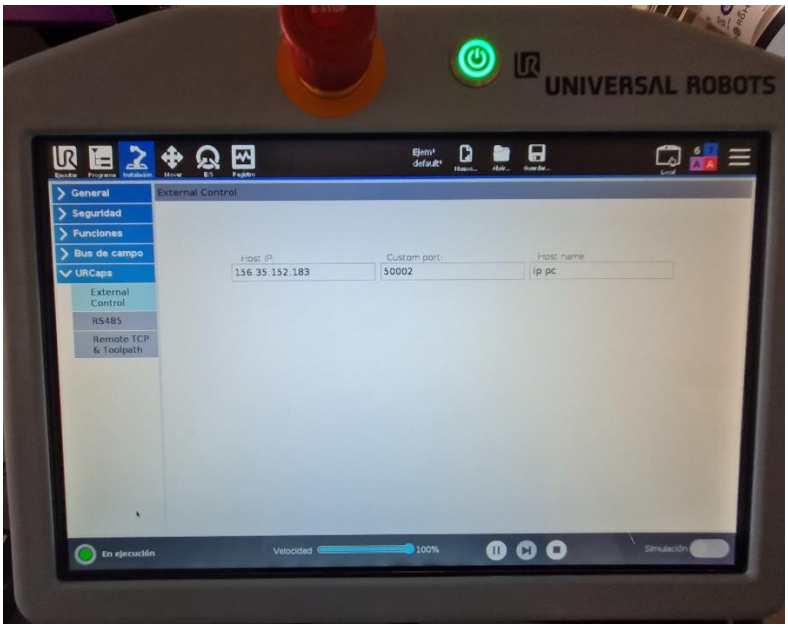


Figura 45: Configuración de la variable External Control

Una vez configurada y guardada la configuración, el siguiente paso para conectar el robot al PC es, en la pestaña de Programa, añadir en un programa nuevo la variable External Control de URCaps.

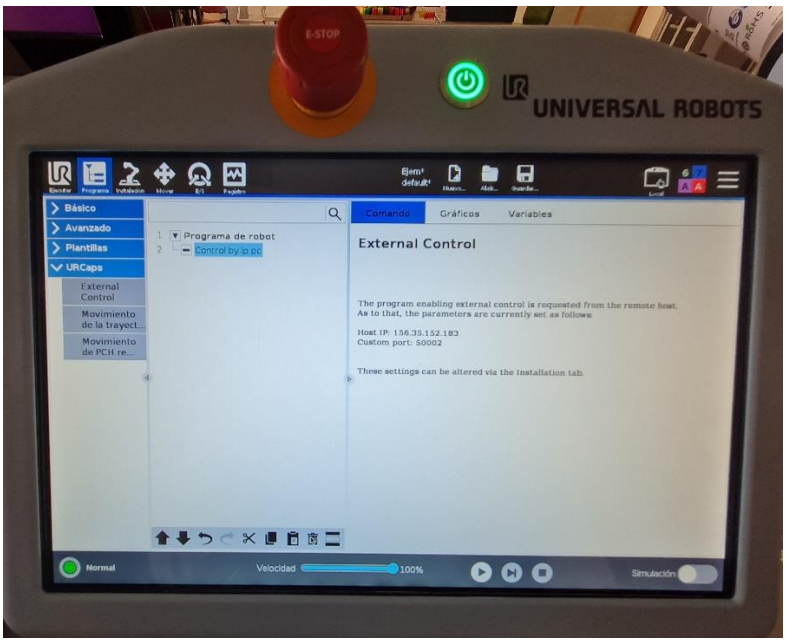


Figura 46: Programa para conectarse al PC

Guardamos y ejecutamos el programa (dándole al play). Tendría que aparecer, en el terminal del PC, siguiente a la respuesta de ejecutar el comando anterior (página 53) lo siguiente:

```
[INFO] [1648120940.962476]: Started controllers: joint_state_controller, scaled_pos_joint_
traj_controller, speed_scaling_state_controller, force_torque_sensor_controller
[ INFO] [1648121016.400398686]: Robot requested program
[ INFO] [1648121016.400532545]: Sent program to robot
[ INFO] [1648121016.530611215]: Robot connected to reverse interface. Ready to receive con
trol commands.
```

Con esto ya se ha establecido conexión del PC con el robot y del robot con el PC.

El siguiente paso es lanzar el archivo *'ur3e_planning_execution.launch'* en modo de control remoto. Para ello, se ejecutará el siguiente comando en una pestaña nueva del terminal:

```
~/catkin_ws$ roslaunch my_moveit_config ur3e_moveit_planning_execution.launch
```

Como se puede comprobar, en este caso no se escribió el código *'sim:=true'*, con lo que se estará trabajando en modo de control remoto.

A partir de aquí ya solo quedaría decidir de nuevo, que programa o archivo se va a utilizar para crear las trayectorias que se quiere que recorra el robot. Las opciones son las mismas que en el modo simulación. Usando la interfaz RVIZ o mediante un archivo de Python.

6.2.1. USANDO UN ARCHIVO DE PYTHON

El archivo Python empleado es el mismo que el explicado en el Apartado 5.2.1. Aquí hay que tener en cuenta que se puede usar cualquier controlador que se configuró en el archivo Python. En modo simulación solo se podía usar la segunda opción, el controlador *'pos_joint_traj_controller'* ya que era el único controlador que había en común entre Moveit y el Gazebo. En este caso como el Robot admite más controladores. Al ejecutar el programa entonces debería empezar a moverse el robot a los puntos que se han programado.

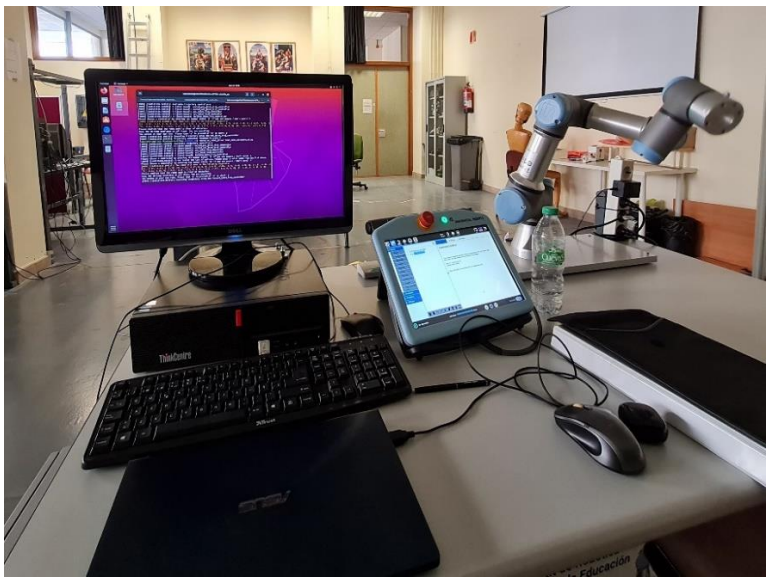


Figura 47: Control Remoto del Robot usando un archivo Pyhton

6.2.2. USANDO RVIZ

Este caso también es el mismo que en el modo simulación. Ejecutando el comando:

```
alejandro@alejandro-VirtualBox:~/catkin_ws$ roslaunch my_moveit_config moveit_rv
iz.launch config:=true
```

Ya se puede empezar a planificar trayectorias y que el robot se mueva al punto dado usando la trayectoria que él ha calculado.

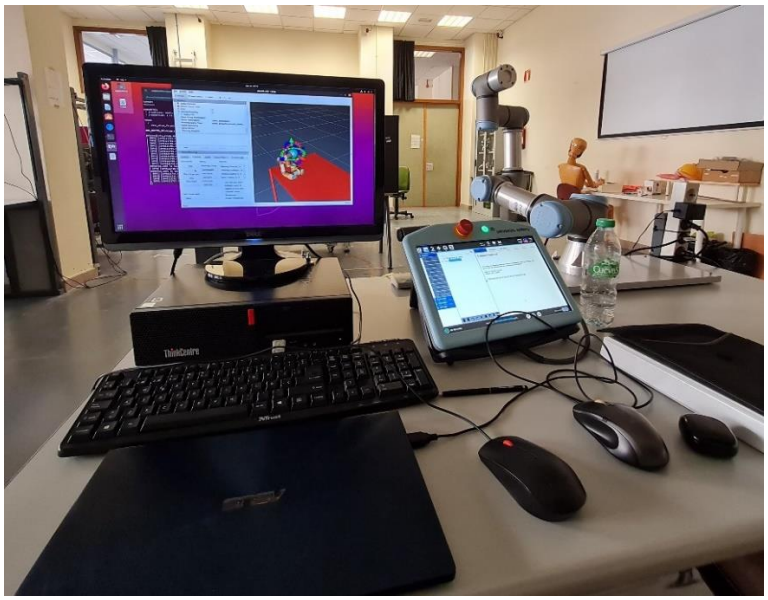


Figura 48: Control Remoto del Robot usando RVIZ

7. '.XACRO' DE LOS OBJETOS

En este apartado están los códigos completos de los objetos creado en el Apartado 3. El lenguaje de programación usado para la creación de los objetos es de XML.

7.1. BASE

```
<?xml version="1.0"?>
<robot name="table"
       xmlns:xi="http://www.w3.org/2001/XInclude"
       xmlns:gazebo="http://playerstage.sourceforge.net/gazebo/xmlschema/#gz"
       xmlns:model="http://playerstage.sourceforge.net/gazebo/xmlschema/#model"
       xmlns:sensor="http://playerstage.sourceforge.net/gazebo/xmlschema/#sensor"
       xmlns:body="http://playerstage.sourceforge.net/gazebo/xmlschema/#body"
       xmlns:geom="http://playerstage.sourceforge.net/gazebo/xmlschema/#geom"
       xmlns:joint="http://playerstage.sourceforge.net/gazebo/xmlschema/#joint"
       xmlns:interface="http://playerstage.sourceforge.net/gazebo/xmlschema/#interface"
       xmlns:rendering="http://playerstage.sourceforge.net/gazebo/xmlschema/#rendering"
       xmlns:renderable="http://playerstage.sourceforge.net/gazebo/xmlschema/#renderable"
       xmlns:controller="http://playerstage.sourceforge.net/gazebo/xmlschema/#controller"
       xmlns:physics="http://playerstage.sourceforge.net/gazebo/xmlschema/#physics">

       <link name="Base">
           <inertial>
                   <mass value="0.5" />
          <origin xyz="0 0 0.5" rpy="0 0 0"/>
          <mass value="20"/>
          <inertia ixx="0.166667" ixy="0" ixz="0" iyy="0.166667" iyz="0" izz="0.16667"/>
          </inertial>
           <visual>
                   <origin xyz="0 0 0.0125" rpy="0 0 0"/>
                   <geometry>
                           <box size="0.7 0.3 0.025"/>#0.67 0.25
          </geometry>
          <material name="Gray">
                   <color rgba="0.5 0.5 0.5 0"/>
          </material>
         </visual>
         <collision>
          <origin xyz="0 0 0.0125" rpy="0 0 0"/>
                   <geometry>
                           <box size="0.7 0.3 0.025"/>
          </geometry>
         </collision>
       </link>
       <gazebo reference ="Base">
         <mu1>0.2</mu1>
```

```
41          <mu2>0.2</mu2>
42          <kp>1000000.0</kp>
43          <kd>1.0</kd>
44          <material>Gazebo/Orange</material>
45        </gazebo>
46
47 </robot>
```

7.2. REGLETA

```
1  <?xml version="1.0"?>
2 <robot name="table"
3        xmlns:xi="http://www.w3.org/2001/XInclude"
4        xmlns:gazebo="http://playerstage.sourceforge.net/gazebo/xmlschema/#gz"
5        xmlns:model="http://playerstage.sourceforge.net/gazebo/xmlschema/#model"
6        xmlns:sensor="http://playerstage.sourceforge.net/gazebo/xmlschema/#sensor"
7        xmlns:body="http://playerstage.sourceforge.net/gazebo/xmlschema/#body"
8        xmlns:geom="http://playerstage.sourceforge.net/gazebo/xmlschema/#geom"
9        xmlns:joint="http://playerstage.sourceforge.net/gazebo/xmlschema/#joint"
10       xmlns:interface="http://playerstage.sourceforge.net/gazebo/xmlschema/#interface"
11       xmlns:rendering="http://playerstage.sourceforge.net/gazebo/xmlschema/#rendering"
12
13 xmlns:renderable="http://playerstage.sourceforge.net/gazebo/xmlschema/#renderable"
14
15 xmlns:controller="http://playerstage.sourceforge.net/gazebo/xmlschema/#controller"
16       xmlns:physics="http://playerstage.sourceforge.net/gazebo/xmlschema/#physics">
17       <link name="Regleta">
18           <inertial>
19                   <mass value="0.2" />
20          <origin xyz="0 0 0" rpy="0 0 0"/>
21          <mass value="20"/>
22          <inertia ixx="0.166667" ixy="0" ixz="0" iyy="0.166667" iyz="0" izz="0.16667"/>
23           </inertial>
24           <visual>
25                   <origin xyz="0 0 0" rpy="0 0 0"/>
26                   <geometry>
27                           <box size="0.045 0.085 0.25"/>#0.67 0.25
28          </geometry>
29          <material name="Gray">
30                   <color rgba="0.5 0.5 0.5 0"/>
31          </material>
32         </visual>
33         <collision>
34          <origin xyz="0 0 0" rpy="0 0 0"/>
35                   <geometry>
36                           <box size="0.045 0.085 0.25"/>
37          </geometry>
38         </collision>
39       </link>
```

```
40        <gazebo reference ="Regleta">
41         <mu1>0.2</mu1>
42         <mu2>0.2</mu2>
43         <kp>1000000.0</kp>
44         <kd>1.0</kd>
45         <material>Gazebo/Grey</material>
46        </gazebo>
47
48 </robot>
```

7.3. MESA

```
 1 <?xml version="1.0"?>
 2 <robot name="table"
 3        xmlns:xi="http://www.w3.org/2001/XInclude"
 4        xmlns:gazebo="http://playerstage.sourceforge.net/gazebo/xmlschema/#gz"
 5        xmlns:model="http://playerstage.sourceforge.net/gazebo/xmlschema/#model"
 6        xmlns:sensor="http://playerstage.sourceforge.net/gazebo/xmlschema/#sensor"
 7        xmlns:body="http://playerstage.sourceforge.net/gazebo/xmlschema/#body"
 8        xmlns:geom="http://playerstage.sourceforge.net/gazebo/xmlschema/#geom"
 9        xmlns:joint="http://playerstage.sourceforge.net/gazebo/xmlschema/#joint"
10        xmlns:interface="http://playerstage.sourceforge.net/gazebo/xmlschema/#interface"
11        xmlns:rendering="http://playerstage.sourceforge.net/gazebo/xmlschema/#rendering"
12
13 xmlns:renderable="http://playerstage.sourceforge.net/gazebo/xmlschema/#renderable"
14
15 xmlns:controller="http://playerstage.sourceforge.net/gazebo/xmlschema/#controller"
16        xmlns:physics="http://playerstage.sourceforge.net/gazebo/xmlschema/#physics">
17
18   <property name="table_height" value="0.75" />
19   <property name="table_width" value="1.5" />
20   <property name="table_depth" value="1.0" />
21   <property name="leg_radius" value="0.02" />
22   <property name="table_x" value="0.75" />
23   <property name="table_y" value="0.5" />
24   <property name="table_z" value="0.0" />
25
26   <property name="table_top_thickness" value="0.08"/>
27
28   <property name="M_PI" value="3.1415926535897931" />
29
30
31   <!-- tabletop height is .55+.01+.025=.585 -->
32   <link name="table_top_link">
33     <inertial>
34       <mass value="1.0" />
35       <!--origin xyz="0.75 0.5 0.71" /-->
36       <origin xyz="0 0 0" />
```

```
37      <inertia ixx="1" ixy="0"  ixz="0"
38               iyy="1" iyz="0"
39               izz="1" />
40    </inertial>
41    <visual>
42      <origin xyz="0.75 0.5 0.73" />
43      <geometry>
44        <box size="1.5 1 0.08" />
45      </geometry>
46    </visual>
47    <collision>
48      <origin xyz="0.75 0.5 0.73" />
49      <geometry>
50        <box size="1.5 1 0.08" />
51      </geometry>
52    </collision>
53  </link>
54  <gazebo reference="table_top_link">
55    <material>Gazebo/LightWood</material>
56    <mu1>50.0</mu1>
57    <mu2>50.0</mu2>
58    <kp>1000000.0</kp>
59    <kd>1.0</kd>
60  </gazebo>
61
62  <joint name="leg1_joint" type="fixed" >
63    <parent link="table_top_link" />
64    <origin xyz="1.5 1 0.75" rpy="0 0 0" />
65    <child link="leg1_link" />
66  </joint>
67  <link name="leg1_link">
68    <inertial>
69      <mass value="1.0" />
70      <origin xyz="0 0 -0.375" />
71      <inertia ixx="0.1" ixy="0"  ixz="0"
72               iyy="0.1" iyz="0"
73               izz="0.01" />
74    </inertial>
75    <visual>
76      <origin xyz="0.0 0.0 -0.375" rpy="0 0 0" />
77      <geometry>
78        <cylinder radius="0.02" length="0.8" />
79      </geometry>
80    </visual>
81    <collision>
82      <origin xyz="0.0 0.0 -0.375" rpy="0.0 0.0 0.0" />
83      <geometry>
84        <cylinder radius="0.02" length="0.8" />
85      </geometry>
86    </collision>
87  </link>
88  <gazebo reference="leg1_link">
```

```
    <material>Gazebo/Red</material>
    <mu1>1000.0</mu1>
    <mu2>1000.0</mu2>
    <kp>10000000.0</kp>
    <kd>1.0</kd>
    <selfCollide>true</selfCollide>
  </gazebo>

  <joint name="leg2_joint" type="fixed" >
    <parent link="table_top_link" />
    <origin xyz="0 1 0.75" rpy="0 0 0" />
    <child link="leg2_link" />
  </joint>
  <link name="leg2_link">
    <inertial>
      <mass value="1.0" />
      <origin xyz="0 0 -0.375" />
      <inertia ixx="0.1" ixy="0"  ixz="0"
               iyy="0.1" iyz="0"
               izz="0.01" />
    </inertial>
    <visual>
      <origin xyz="0.0 0.0 -0.375" rpy="0 0 0" />
      <geometry>
        <cylinder radius="0.02" length="0.8" />
      </geometry>
    </visual>
    <collision>
      <origin xyz="0.0 0.0 -0.375" rpy="0.0 0.0 0.0" />
      <geometry>
        <cylinder radius="0.02" length="0.8" />
      </geometry>
    </collision>
  </link>
  <gazebo reference="leg2_link">
    <material>Gazebo/Red</material>
    <mu1>1000.0</mu1>
    <mu2>1000.0</mu2>
    <kp>10000000.0</kp>
    <kd>1.0</kd>
    <selfCollide>true</selfCollide>
  </gazebo>

  <joint name="leg3_joint" type="fixed" >
    <parent link="table_top_link" />
    <origin xyz="1.5 0 0.75" rpy="0 0 0" />
    <child link="leg3_link" />
  </joint>
  <link name="leg3_link">
    <inertial>
      <mass value="1.0" />
      <origin xyz="0 0 -0.375" />
```

```
141       <inertia ixx="0.1" ixy="0"  ixz="0"
142                iyy="0.1" iyz="0"
143                izz="0.01" />
144     </inertial>
145     <visual>
146       <origin xyz="0.0 0.0 -0.375" rpy="0 0 0" />
147       <geometry>
148         <cylinder radius="0.02" length="0.8" />
149       </geometry>
150     </visual>
151     <collision>
152       <origin xyz="0.0 0.0 -0.375" rpy="0.0 0.0 0.0" />
153       <geometry>
154         <cylinder radius="0.02" length="0.8" />
155       </geometry>
156     </collision>
157   </link>
158   <gazebo reference="leg3_link">
159     <material>Gazebo/Red</material>
160     <mu1>1000.0</mu1>
161     <mu2>1000.0</mu2>
162     <kp>10000000.0</kp>
163     <kd>1.0</kd>
164     <selfCollide>true</selfCollide>
165   </gazebo>
166
167   <joint name="leg4_joint" type="fixed" >
168     <parent link="table_top_link" />
169     <origin xyz="0 0 0.75" rpy="0 0 0" />
170     <child link="leg4_link" />
171   </joint>
172   <link name="leg4_link">
173     <inertial>
174       <mass value="1.0" />
175       <origin xyz="0 0 -0.375" />
176       <inertia ixx="0.1" ixy="0"  ixz="0"
177                iyy="0.1" iyz="0"
178                izz="0.01" />
179     </inertial>
180     <visual>
181       <origin xyz="0.0 0.0 -0.375" rpy="0 0 0" />
182       <geometry>
183         <cylinder radius="0.02" length="0.8" />
184       </geometry>
185     </visual>
186     <collision>
187       <origin xyz="0.0 0.0 -0.375" rpy="0.0 0.0 0.0" />
188       <geometry>
189         <cylinder radius="0.02" length="0.8" />
190       </geometry>
191     </collision>
192   </link>
```

```
193   <gazebo reference="leg4_link">
194     <material>Gazebo/Red</material>
195     <mu1>1000.0</mu1>
196     <mu2>1000.0</mu2>
197     <kp>10000000.0</kp>
198     <kd>1.0</kd>
199     <selfCollide>true</selfCollide>
200   </gazebo>
201   <gazebo>
202     <canonicalBody>table_top_link</canonicalBody>
203   </gazebo>
204
205
206 </robot>
207
208
```

8. ENLACES DE INTERÉS

En este último apartado dejo enlaces de interés que pueden ser útiles cuando se quieran hacer ampliaciones o mejoras.

[1] https://www.youtube.com/watch?v=ayp87SjrwPc&t=706s (Última visita: 07/06/2022)

[2] https://www.youtube.com/watch?v=BxCik8OI1Fw (Última visita: 07/06/2022)

[3] https://ros-planning.github.io/moveit_tutorials/

[4] http://wiki.ros.org/ur3_moveit_config