UNIVERSIDAD NACIONAL DE COLOMBIA
DEPARTAMENTO DE INGENIERÍA MECÁNICA Y MECATRÓNICA
ROBÓTICA 2020-II
PROFESOR : Pedro Fabían Cárdenas
MONITOR Jurgen Krejci Muñoz

Laboratorio 4 - Simulación con Gazebo

1. Objetivos

- Hacer una introducción al manejo de la herramienta *Gazebo* para la simulación de robots.
- Simular el comportamiento del robot *Phantom X Pincher* teniendo en cuenta propiedades físicas básicas y haciendo uso de paquete ROS control.
- Determinar el modelo cinemático inverso del robot *Phantom X*.
- Generar trayectorias simples a partir del modelo cinemático inverso del robot.
- Implementar el modelo cinemático inverso del robot en MATLAB.

2. Requisitos

- Ubuntu versión 18.04 con ROS Melodic
- Espacio de trabajo para *catkin* correctamente configurado.
- MATLAB 2015b o superior instalado en el equipo.
- Robotics toolbox de Mathworks (Incluido por defecto desde la versión 2015 en adelante).
- Toolbox de robótica de *Peter Corke*.
- Modelo URDF del robot Phantom y archivos de mallado del robot
- Simulador *Gazebo* V11.0 (Incluido en la instalación por defecto de ROS) Link

Figura 1: Robot Phantom X Pincher

3. Ejercicio de laboratorio

A continuación se describen los pasos de la practica a realizar, para guiar y complementar el desarrollo de la misma se recomienda ver la sección de ejemplo y la sección de recursos, donde se encuentra el material guía.

3.1. Modelo URDF robot PhantomX

Como primera parte de la practica se requiere realizar algunos ajustes sobre el modelo URDF que se ha manejo durante la practica anterior, se busca simplificar y facilitar la depuración de este modelo por medio del manejo de macros. Se van a usar las herramientas de Xacro (XML Macros) para hacer la modificación del archivo *phantom_urdf.xacro* que se manejo durante el laboratorio anterior. Para ello:

- Crear un macro que permita hacer la creación de forma parametrica de las juntas de robot *PhantomX*.
- Crear un macro que permita hacer la creación de forma parametrica de los eslabones de robot *PhantomX*.
- Añadir al modelo los archivos de mallado correspondiente al robot PhantomX (geometría real del robot), recordar que se deben usar modelos en extensión *.stl* o *.dae*.
- Realizar los ajustes necesarios en las propiedades de las juntas para tener un gripper funcional.

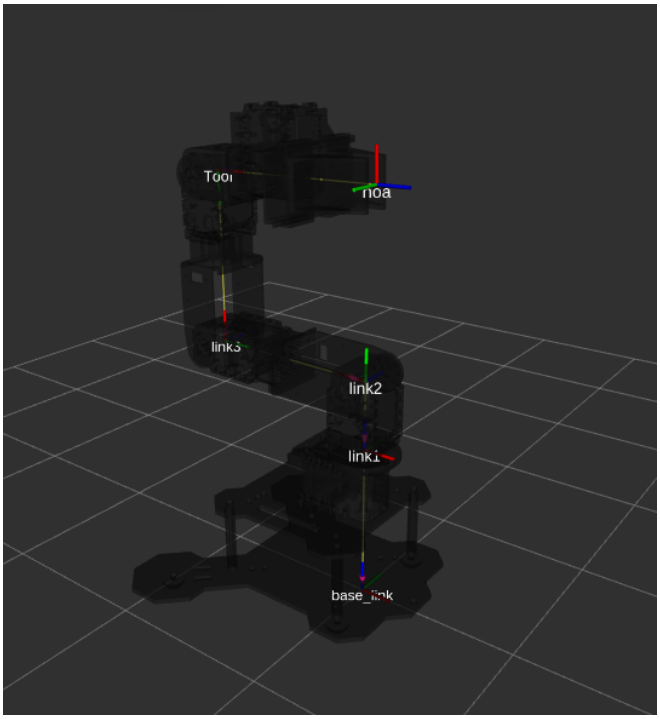

Figura 2: Modelo URDF del Robot PhantomX

3.2. Propiedades de inercia y colisiones

Para poder realizar de forma satisfactoria una simulación de la dinámica del robot en *Gazebo* es necesario especificar dentro del modelo URDF las propiedades de inercia y colisiones que va a tener el robot, para ello:

- Usando algún software CAD calcular para cada eslabón del robot propiedades de masa y momentos de inercia. (Tener en cuenta que para este punto no conocemos con exactitud el material de los eslabones, se sugiere hacer aproximaciones del material, procurar hacer una selección lógica).
- Especificar dentro del archivo URDF *phantom_urdf.xacro*, con ayuda de los macros creados en la sección anterior, para cada eslabón las propiedades de colisiones respectivas, se deja a consideración del grupo de trabajo la geometria a usar para el solido de colisión.

- Especificar dentro del arhivo URDF *phantom_urdf.xacro*, con ayuda de los macros creados en la sección anterior, para cada eslabón las propiedades de inercia respectivas.

3.3. Transmisión en URDF y creación de controladores

En conjunto con el modelo URDF del robot con propiedades de inercia y colisiones especificadas, se requiere hacer la creación de controladores virtuales para cada junta del robot para poder realizar una simulación, para esta sección:

- Especificar dentro del arhivo URDF *phantom_urdf.xacro*, para cada junta del robot, las propiedades de transmisión en el modelo URDF requeridas para describir la relación entre el actuadores y la juntas del robot, hacer uso de los macros creados previamente.
- Hacer uso del paquete **controller_manager** como dependencia de nuestro paquete de robot *PhantomX* para la creación de controladores para cada una de las juntas del robot (interfaz directa, sin PID), para esto se debe crear un archivo tipo *.yaml* donde se especifiquen los parámetros de los controladores a usar.
- Hacer las modificaciones necesarias de los archivos **CMakeList.txt** y **package.xml** de nuestro paquete del robot *PhantomX* para incluir las dependencias necesarias para ejecutar el mismo.

3.4. Simulación en Gazebo + ROS

- Crear un archivo tipo *.launch* que permita iniciar la simulación del robot *PhantomX* dentro del entorno *Gazebo*.
- Verificar los tópicos y nodos creados por el paquete **controller_manager**, especificar las funciones de estos y describir que servicios tienen.
- Desde consola suscribirse a los tópicos de estado de juntas del robot (*robot_state_publisher*), verificar el correcto funcionamiento de los controladores publicando (de forma simultanea) en los tópicos de comandos de cada junta.

3.5. Cinemática Inversa

El problema cinemático inverso consiste en determinar la configuración articular de un manipulador, dadas la posición y orientación del efector final respecto a la base. Este problema puede resolverse mediante métodos geométricos, algebraicos o numéricos. En el caso particular del robot *Phantom X* el cual posee 4 GDL, el enfoque más práctico es combinar el método geométrico con el desacople de muñeca. Considerando que el robot se mueve en un subespacio solución.

- Determine el modelo cinemático inverso del manipulador *Phantom X* (Utilizar la metodologia explicada en clase). Se recomienda usar el *toolbox* para verificar la solución hallada.
- Sabiendo que el robot *Phantom X* posee 4 GDL, de los cuales 3 corresponden a posición y el GDL restante proporciona una medida independiente en orientación, determine ¿A que ángulo de orientación (asuma orientación en ángulos fijos) este corresponde?
- Determine ¿Cuántas soluciones posibles existen para la cinemática inversa del manipulador *Phantom X*?
- Consulte en qué consiste el espacio diestro de un manipulador.

3.6. Matlab

- Implementar en Matlab el modelo cinemático inverso de la sección anterior, de nuevo se recomienda usar el *toolbox* para verificar la solución hallada.
- Esboce el espacio de trabajo del robot *Phantom X*.
- Consulte los métodos disponibles en el *toolbox* para determinar la cinemática inversa de un manipulador. Busque en la literatura y explique al menos uno de los métodos.

- Elabore un programa en el cual se proporcione una MTH para el efector final y el manipulador *Phantom X* alcance dicha configuración desde su configuración actual (Para visualizar la configuración final del robot usar la herramienta plot del toolbox Perte Corke).
- Programe una secuencia simple de movimiento para su robot, se busca una trayectoria de *pick and place* donde el robot se mueva entre 5 puntos básicos punto de aproximación (1), punto de recogida del material (2), punto elevación (3), punto desplazamiento (4) y punto de entrega de material (5). Ver figura 3. Queda a libertad de los estudiantes definir la posición de estos puntos (escoger a su criterio, el fin es demostrativo).

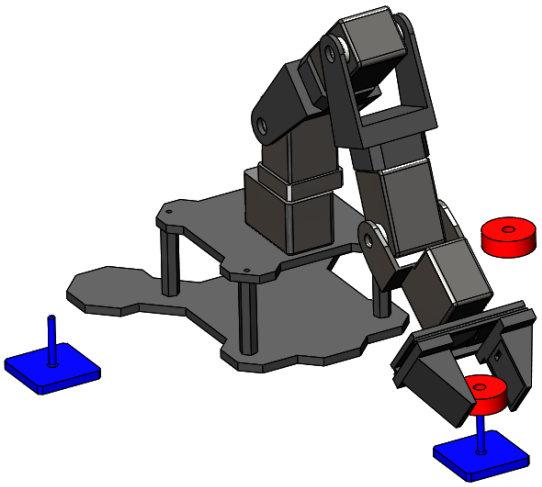

Figura 3: Ejemplo proceso de recogida y desplazamiento de piezas usando robot PhantomX.

3.7. Gazebo + Matlab + ROS

- En Matlab crear una función que permita publicar en el tópico de comando para cada junta, verificar el correcto movimiento de la junta en Gazebo.
- En Matlab crear una función que permita suscribirse al tópico de estado para cada junta (*robot_state_publisher*), describir la información recibida y verificar la variación de los valores en conjunto a la función del punto anterior.
- Modificar la simulación creada en Matlab de la sección anterior, para que las trayectorias de Pick and Place creadas puedan ser ahora simuladas en Gazebo (Tener en cuenta que Gazebo si permite al manipulador interactuar con objetos externos).

Observaciones:

1. El informe deberá en presentarse en **.tex** y **PDF**. Se debe subir al Moodle dentro de las fechas programadas.
2. **Forma de trabajo:** Grupos de laboratorio.
3. Los puntos que requieran implementación de funciones deberán tener comentarios de cómo se utilizan y adjuntar archivos **.m**.
4. Se debe incluir un archivo comprimido (*.rar* o *.zip*) donde se encuentren todo el paquete creado, aquí deberían estar incluidos el modelo URDF, CAD de robot, archivos *.launch*, archivo *.yaml*, entre otros necesarios para ejecutar el paquete.
5. Se deberá incluir un video en el que se muestre el funcionamiento del *script* generado en Matlab que permite controlar el robot simulado en *Gazebo* usando las funciones que integran a ROS con Matlab. El video deberá ser

subido a *Youtube* y enviar el respectivo enlace en el informe, este ítem es de **CARÁCTER OBLIGATORIO** y equivale al 20 % de la nota del laboratorio. El video debe tener una presentación (front) donde se indique la universidad, estudiantes, curso robotica, profesor y año.

6. **Fecha de entrega: 10/11/2020**.

4. Recursos y material de apoyo

A continuación se comparten una serie de recursos web como tutoriales y vídeos donde se trabaja en detalle cada uno de los items de la practica de laboratorio.

1. Gazebo, Robot simulation made easy Link
2. Xacro (XML Macros) Link
3. Using Xacro to Clean Up a URDF File Link
4. Using URDF in Gazebo Link
5. Transmission in URDF (Reference) Link
6. Paquete *robot_state_publisher* Link
7. Tutorial de creación de un manipulador con sus controladores desde cero con URDF para Gazebo Link
8. Creación de controladores con el paquete *controller_manager* para manejo con URDF Link
9. Adding Physical and Collision Properties to a URDF Model Link

Adicionalmente se comparte el repositorio con el ejemplo presentado en esta guía (manipulador 2R), **se recomienda usar este como ejemplo de base para el desarrollo de la guía**. 2R Manipulator URDF by Jurgen Krejci

5. Ejemplo

En esta sección vamos a trabajar nuevamente sobre un robot con configuración 2R como el que se puede ver en la figura 4, partimos de que ya tiene un modelo URDF del robot con sus debidos modelos CAD, juntas creadas y gripper funcional. Para este ejemplo se considera que las juntas del robot están fabricadas en Aluminio 6061, bajo el cual se hacen los cálculos de masas e inercias.

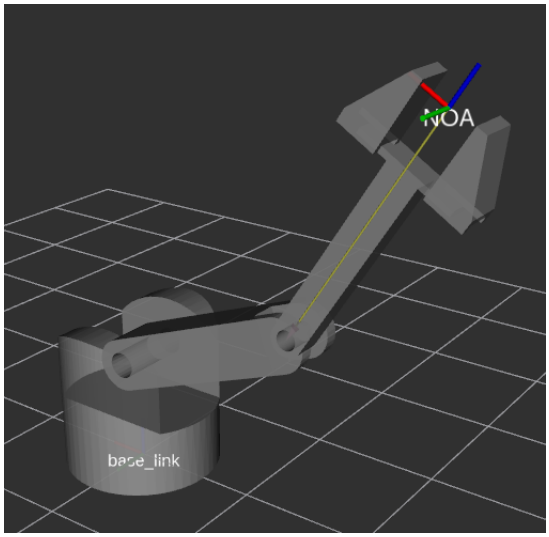

Figura 4: Robot 2R

Articulación (i)	α_{i-1}	a_{i-1}	d_i	θ_i	**Offset**
1	0	0	0	q1	0
2	0	l_1	0	q2	0

Tabla 1: Parametros DHmod robot 2R

5.1. Especificación de propiedades físicas y de colisiones

Conociendo las dimensiones de los elementos que componen al robot 2R y el material del que están hechos, hacemos el calculo de la masa y el tensor de inercias (con respecto al centro de masa) para cada eslabon. Estas propiedades son agregadas en el archivo URDF de descripción del robot, usando la etiqueta $< inertial >$, como se puede ver en el siguiente ejemplo:

```
<link name="link1">
    <visual>
        <origin rpy="0 1.57 0" xyz="0.5 0 0"/>
        <geometry>
            <mesh filename="package://2R_Gazebo/meshes/Link1.stl"
        </geometry>
    </visual>
    <inertial>
        <mass value="10"/>
        <inertia ixx="0.4" ixy="0.0" ixz="0.0" iyy="0.4" iyz="0.0" izz="0.2"/>
    </inertial>
</link>
```

El fragmento anterior corresponde a la creación del eslabón numero 1 del robot 2R, para añadir estas propiedades se debe tener en cuenta:

- La masa del elemento se especifica Kg
- Los momentos de inercia se especifican en Kgm^2
- No olvidar que los momentos de inercia se calculan con respecto al centro de masa del eslabón, se especifica el tensor de inercia (al ser simétrico solo hay que agregar 6 elementos de la matriz):

$$I = \begin{pmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{pmatrix}$$

Para el caso de las propiedades de colisiones se especifica un solido que el simulador va usar para delimitar las fronteras de cada elemento, es usual que los solidos especificados en las propiedades de colisiones sean mas simples que los que se muestran en las propiedades visuales, ya que los limites de colisión son los usados para hacer los cálculos en el motor físico de *Gazebo*, por ello es recomendado, en caso de que no se tenga buenos recursos de computo para la simulación, se haga uso de solidos simples que aproximen las propiedades visuales.

En un archivo URDF las propiedades de colisiones se especifican con la etiqueta $< collision >$:

```
<link name="link1">
    <visual>
        <origin rpy="0 1.57 0" xyz="0.5 0 0"/>
        <geometry>
            <mesh filename="package://2R_Gazebo/meshes/Link1.stl"
        </geometry>
    </visual>
    <collision>
```

```
        <origin rpy="0 1.57 0" xyz="0.5 0 0"/>
        <geometry>
            <mesh filename="package://2R_Gazebo/meshes/Link1.stl"/>
        </geometry>
    </collision>
    <inertial>
        <mass value="10"/>
        <inertia ixx="0.4" ixy="0.0" ixz="0.0" iyy="0.4" iyz="0.0" izz="0.2"/>
    </inertial>
</link>
```

Como se puede ver, el solido de colisión se especifica de la misma forma en que se especificaría un elemento visual, se especifica un origen y la geometría, en este caso se decidió usar la misma malla que se uso en las propiedades visuales (esto implica mayores recursos para el computo, simulación mas lenta) en ves de una geometría simple (simulación mas rápida). Los solidos de colisión con geometrías simples se pueden definir con etiquetas que ya son familiares (de las propiedades visuales ya trabajadas), por ejemplo:

```
<cylinder length="0.6" radius="0.2"/>
<box size="0.6 0.1 0.2"/>
<sphere radius="0.2"/>
```

Para un detalle mas a profundidad de estas propiedades se sugiere revisar el link 9 de la sección de recursos y material de apoyo.

5.2. Creación de macros con Xacro

La herramienta Xacro nos permite crear macros para XML los cuales nos sirven para simplificar y optimizar los archivos de los modelos URDF que estamos usando, con esta herramienta podemos crear constantes, hacer operaciones matemáticas simples y reutilizar fragmentos de código usando macros. Para el caso de nuestro robot 2R se hace un macro para la creación de juntas de forma parametrica y otro para la creación de eslabones de forma parametrica. Vamos a tratar como ejemplo el caso del macro para eslabones:

Codigo para la creación de un eslabón sin macro

```
<link name="link_01">
    <inertial>
        <mass value="1.57" />
        <origin rpy="0 0 0" xyz="0 0 0" />
        <inertia ixx="130.0" ixy="0" ixz="0" iyy="130.0" iyz="0" izz="90.0" />
     </inertial>
    <collision>
        <origin rpy="0 0 0" xyz="0 0 0" />
        <geometry>
            <mesh filename="package://2R_Gazebo/meshes/Link1.stl"/>
        </geometry>
    </collision>
    <visual>
        <origin rpy="0 0 0" xyz="0 0 0" />
        <geometry>
            <mesh filename="package://2R_Gazebo/meshes/Link1.stl"/>
        </geometry>
    </visual>
</link>
```

Valores que se usen en múltiples oportunidades pueden ser utilizados como constantes, para esto de usa la etiqueta $< xacro : property/ >$ la cual nos permite crear una variable que guarde un valor, esto nos permite cambiar fácilmente un valor especifico en todo un archivo sin tener que ir linea a linea buscando donde se ha usado el valor en cuestión. Para el caso del robot 2R se ha creado un archivo adicional al modelo URDF llamado *robot_parameters.xacro* en este se incluyen todos las constantes que eventualmente requieran ser manejadas en el modelo.

Archivo *robot_parameters.xacro*

```
<?xml version="1.0" ?>
<robot xmlns:xacro="http://www.ros.org/wiki/xacro">
  <xacro:property name="link_00_name" value="base_link" />
  <xacro:property name="link_01_name" value="link_01" />
  <xacro:property name="link_02_name" value="link_02" />
  <xacro:property name="link_03_name" value="finger_1" />
  <xacro:property name="link_04_name" value="finger_2" />
  <xacro:property name="joint_01_name" value="joint_1" />
  <xacro:property name="joint_02_name" value="joint_2" />
  <xacro:property name="joint_03_name" value="gripper_1" />
  <xacro:property name="joint_04_name" value="gripper_2" />
</robot>
```

Como se puede ver en este caso se han especificado como variables los nombres de cada eslabón y junta, para que en caso de ser cambiados solo se tenga que modificar una linea, a la hora de definir un link usariamos:

```
<link name="${link_01_name}">
```

Para incluir el archivo de parámetros en nuestro modelo URDF ,usamos el siguiente código (después de la etiqueta $< robot >$ al principio del URDF):

```
<xacro:include filename="$(find 2R_Gazebo)/urdf/robot_parameters.xacro" />
```

Para poder simplificar nuestro modelo URDF vamos crear macros, estos elementos nos permiten fácilmente repetir largas porciones de código donde solo se requiera cambiar algunos elementos específicos, como se puede ver, fragmentos de código como la creación de eslabones son siempre iguales, por lo cual un macro seria bastante útil para definir múltiples eslabones, usando menor cantidad de código.

Macro para la creación de una junta parametrizada

```
<xacro:macro name="m_link_mesh" params="name origin_xyz origin_rpy meshfile mass ixx ixy ixz
    iyy iyz izz">
    <link name="${name}">
        <inertial>
            <mass value="${mass}" />
            <origin rpy="${origin_rpy}" xyz="${origin_xyz}" />
            <inertia ixx="${ixx}" ixy="${ixy}" ixz="${ixz}" iyy="${iyy}" iyz="${iyz}" izz="${izz
                }" />
        </inertial>
        <collision>
            <origin rpy="${origin_rpy}" xyz="${origin_xyz}" />
            <geometry>
                <mesh filename="${meshfile}"/>
            </geometry>
        </collision>
        <visual>
```

```
            <origin rpy="${origin_rpy}" xyz="${origin_xyz}" />
            <geometry>
                <mesh filename="${meshfile}"/>
            </geometry>
        </visual>
    </link>
</xacro:macro>
```

Creación del eslabón haciendo uso del macro y variables creadas

```
  <m_link_mesh name="${link_00_name}"
    origin_xyz="0 0 0"
    origin_rpy="0 0 0"
    meshfile="package://2R_Gazebo/meshes/Base.stl"
    mass="1.024"
    ixx="170.0" ixy="0" ixz="0"
    iyy="170.0" iyz="0"
    izz="170.0"
  />
```

Como se puede ver en este ultimo ejemplo se obtiene un código mucho mas simple dentro del modelo URDF, el macro se crea una única ves (Puede ser dentro del mismo archivo o en uno aparte e incluirlo), como se puede ver el macro usa las mismas etiquetas familiares para crear un eslabón ($< link >$), solo que en este caso los parámetros se dejan especificados de forma general (similar a como se haría con una función en programación convencional) usando el símbolo $ y un par de llaves. Para el caso del robot 2R, se ha creado un archivo adicional llamado *links_joints.xacro*, donde únicamente se ubican los macros creados, en el archivo que contiene el modelo URDF principal hemos incluido el archivo para poder acceder a los macros a usar.

Para complementar el manejo de macros se recomienda revisar el video numero 2 y 3 de del recurso 7 compartido en la sección de material de apoyo, asi como revisar en detalle los archivos de la carpeta **urdf** del paquete **2R_Gazebo** que viene incluido con esta guia.

5.3. Elementos de transmisión y controladores

Para poder hacer control sobre las juntas de nuestro robot necesitamos especificar dentro del modelo URDF la relación que hay entre los eslabones, juntas y actuadores que tiene nuestro robot, para esta tarea los modelos URDF cuentan con elementos llamados **transmission**, estos nos permiten especificar las relaciones mencionadas dentro del modelo. A continuación se presenta la creación del elemento de transmisión para la junta 1 de nuestro robot 2R:

```
<transmission name="trans_link_01">
    <type>transmission_interface/SimpleTransmission</type>
    <joint name="link_01">
        <hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>
    </joint>
    <actuator name="motor_link_01">
        <mechanicalReduction>1</mechanicalReduction>
        <hardwareInterface>hardware_interface/PositionJointInterfacee</hardwareInterface>
    </actuator>
</transmission>
```

Del fragmento de código anterior cabe destacar:

- En el parámetro **joint** se debe especificar la junta correspondiente que se creo en el modelo URDF, se recomienda hacer uso de macros para facilitar esta tarea.
- En el momento el único tipo de transmisión que hay implementada corresponde a *SimpleTransmission*

- Los controladores creados tienen como salidas posición, por ello en la propiedad *hardwareInterface* definimos interfaces con control de posición .
- Si es necesario es posible aplicarle al actuador un factor de reducción mecánica.

Los elementos tipo **transmission** pueden ser creados de forma independiente a las juntas o eslabones, por tanto son declarados con su propia etiqueta, para el caso de robot 2R se crearon 4 transmisiones, 2 para las juntas rotacionales y 2 para el gripper, tener en cuenta que la simulación en *Gazebo* no permite hacer mímicos entre juntas, como se había hecho para la simulación en RViz, pero podemos agrupar 2 controladores de junta para publicar al mismo tiempo en sus tópicos respectivos (ver archivo *.yaml*). Para mas detalle con respecto a la creación de elementos de transmisión en URDF revisar el recurso 5 de la sección de material de apoyo.

Para la implementación de controladores de juntas vamos a hacer uso de los paquetes **controller_manager joint_state_controller** y **robot_state_publisher**, el primero implementa los algoritmos de control que van a ser usados entre cada junta y los otros 2 crean los tópicos necesarios para que la información de la simulación llegue a los controladores y a su ves de estos sea aplicada la señal de control a los actuadores creados, adicionalmente *joint_state_controller* nos ayuda con la creación de tópicos que nos permiten comandar juntas y *robot_state_publisher* con los tópicos para ver su estado actual.

Si se parte del paquete ya creado en el laboratorio previo, se requiere hacer modificaciones en los archivos **CMakeList.txt** y **package.xml** del paquete, esta parte se puede omitir si al crear el paquete se incluyen desde un inicio las dependencias:

```
$ cd catkin_ws/src
$ catkin_create_pkg 2R_Gazebo urdf controller_manager joint_state_controller
    robot_state_publisher
```

Para modificar el paquete de la practica anterior se deben hacer las siguientes modificaciones de forma manual:

- **package.xml**, buscar la etiqueta $<$ *buildtool_depend* $>$ y añadir:

```
  <buildtool_depend>catkin</buildtool_depend>

  <build_depend>urdf</build_depend>
  <build_export_depend>urdf</build_export_depend>
  <exec_depend>urdf</exec_depend>

  <build_depend>controller_manager</build_depend>
  <build_export_depend>controller_manager</build_export_depend>
  <exec_depend>controller_manager</exec_depend>des

  <build_depend>joint_state_controller</build_depend>
  <build_export_depend>joint_state_controller</build_export_depend>
  <exec_depend>joint_state_controller</exec_depend>
```

- **CMakeList.txt**, buscar *find_package* y agregar:

```
find_package(catkin REQUIRED COMPONENTS
  urdf
  controller_manager
  joint_state_controller
  robot_state_publisher
)
```

Adicionalmente dentro del modelo URDF debemos añadir un plugin para que *Gazebo* pueda interactuar desde su entorno con los topicos y nodos que eventualmente crearan estas dependencias. El plugin puede ser agregado en cualquier punto dentro de la etiqueta $<$ *robot* $>$ del modelo.

```
<gazebo>
    <plugin name="gazebo_ros_control" filename="libgazebo_ros_control.so">
    </plugin>
</gazebo>
```

Con los pasos anteriores ya van a cargarse las dependencias requeridas para el manejo de controladores para nuestra simulación. Para especificar los parámetros de nuestros controladores necesitamos del uso de un archivo de configuración tipo *.yaml*, a continuación se presenta el archivo usado para describir los controladores del robot 2R, destacar como se agrupan los 2 controladores del gripper para poder publicar en ambos al mismo tiempo (*JointGroupPositionController* agrupa varios controladores de posición).

Archivo de configuración joints.yaml

```
joint_state_controller:
  type: joint_state_controller/JointStateController
  publish_rate: 50.0

# Position Controllers ---------------------------------------
joint1_position_controller:
  type: position_controllers/JointPositionController
  joint: joint_1
joint2_position_controller:
  type: position_controllers/JointPositionController
  joint: joint_2
gripper_position_controller:
  type: position_controllers/JointGroupPositionController
  joints:
   - gripper_1
   - gripper_2
```

Finalmente solo nos queda crear el archivo *.launch* que nos permita lanzar nuestro modelo de robot con los controladores necesarios para la simulación, para esto creamos un el archivo **spawn.lauch**, a continuación presentamos su contenido:

```
<?xml version="1.0" encoding="UTF-8"?>
<launch>
       <!-- Robot Model -->
    <param name="robot_description" command="$(find xacro)/xacro --inorder '$(find 2R_Gazebo)/
        urdf/2r_description_gazebo.xacro'" />
       <arg name="x" default="0"/>
       <arg name="y" default="0"/>
       <arg name="z" default="0"/>

       <!-- Load controllers -->
    <rosparam command="load" file="$(find 2R_Gazebo)/config/joints.yaml"/>

       <!-- Controllers -->
       <node name="controller_spawner" pkg="controller_manager" type="spawner"
              respawn="false" output="screen"
              args="
              joint_state_controller
              joint1_position_controller
              joint2_position_controller
              gripper_position_controller
              --timeout 60">
```

```
        </node>

        <!-- Gazebo Initialization -->
        <include file="$(find gazebo_ros)/launch/empty_world.launch">
               <arg name="use_sim_time" value="true"/>
        </include>

        <!-- Spawn Robot Model -->
        <node name="mybot_spawn"
               pkg="gazebo_ros"
               type="spawn_model"
               output="screen"
               args="-urdf -param robot_description
               -model 2R -x $(arg x) -y $(arg y) -z $(arg z)" />

        <!-- State Publisher -->
        <node name="robot_state_publisher" pkg="robot_state_publisher" type="
            robot_state_publisher"/>
</launch>
```

Del archivo anterior podemos destacar:

- En el fragmento **Robot Model** se carga el modelo URDF del robot y se especifica la posición de la escena donde va a ser ubicado.
- En el fragmento **Spawn Robot Model** se carga el modelo a Gazebo y se inicia la visualización en simulador.
- En el fragmento **Load Controllers** se carga el archivo de configuración de parámetros de controladores.
- Finalmente en el fragmento **Controllers** se crea el controlador de cada junta con la configuración especificada.

Para un mayor detalle sobre la creación de controladores se sugiere ver el vídeo 5 del recurso 7 en la sección de material de apoyo , asi como revisar el ejemplo del recurso 8.

5.4. Simulación en Gazebo

Para finalmente lanzar nuestra simulación en *Gazebo* debemos contar con el archivo *Spawn.launch* creado, el respectivo modelo URDF con todos los elementos necesarios, el archivo *.yaml* con la configuración de controladores y el paquete con todas las dependencias incluidas.A continuación presentamos el árbol de archivos que tiene el paquete del robot 2R (En el paquete se incluyen los arhcivos para RViz y tambien para Gazebo):

```
2R_Gazebo
├── meshes
│   └── (CAD del robot*)
├── config
│   ├── config.rviz
│   └── joints.yaml
├── launch
│   ├── rviz.launch
│   └── spawn.lauch
├── urdf
│   ├── 2r_description_gazebo.xacro
│   ├── 2r_description.xacro
│   ├── links_joints.xacro
│   └── robot_parameters.xacro
├── CMakeLists.txt
└── package.xml
```

Para la practica se requiere del software *Gazebo*, usualmente al hacer una instalación completa de ROS este software ya viene incluido, se puede verificar la instalación lanzando la simulación de ejemplo (se debe visualizar el modelo de la figura 5):

```
$ sudo apt-get install ros-melodic-urdf-sim-tutorial
$ roslaunch urdf_sim_tutorial gazebo.launch
```

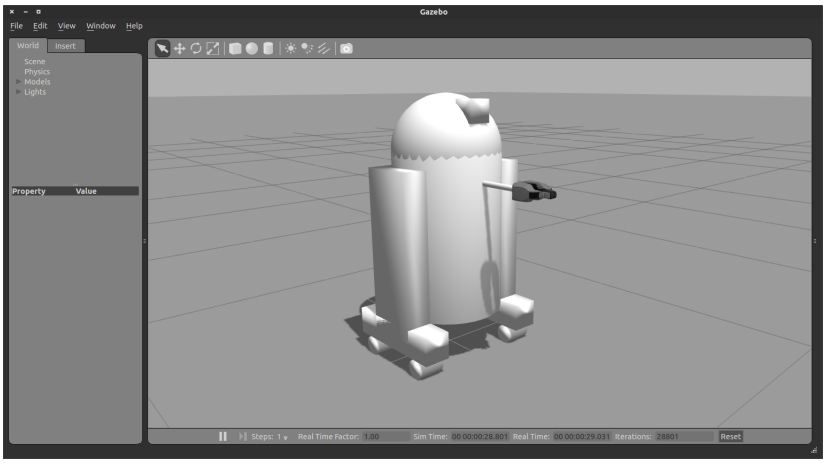

Figura 5: Simulación de prueba con Gazebo

En caso de no tener el software instalado por defecto con ROS, se recomienda revisar el recurso 1 en la zona de material de apoyo, donde se encuentra la información de instalación, adicionalmente se recomienda el siguiente Link.

Para lanzar la simulación, inciamos desde un terminal:

```
$ roslaunch 2R_Gazebo spawn.launch
```

La simulación se debe poder visualizar tal y como se puede apreciar en la figura 6, en este punto solo restaría revisar los tópicos creados y buscar los correspondientes a los controladores de junta (*command* y *state*).

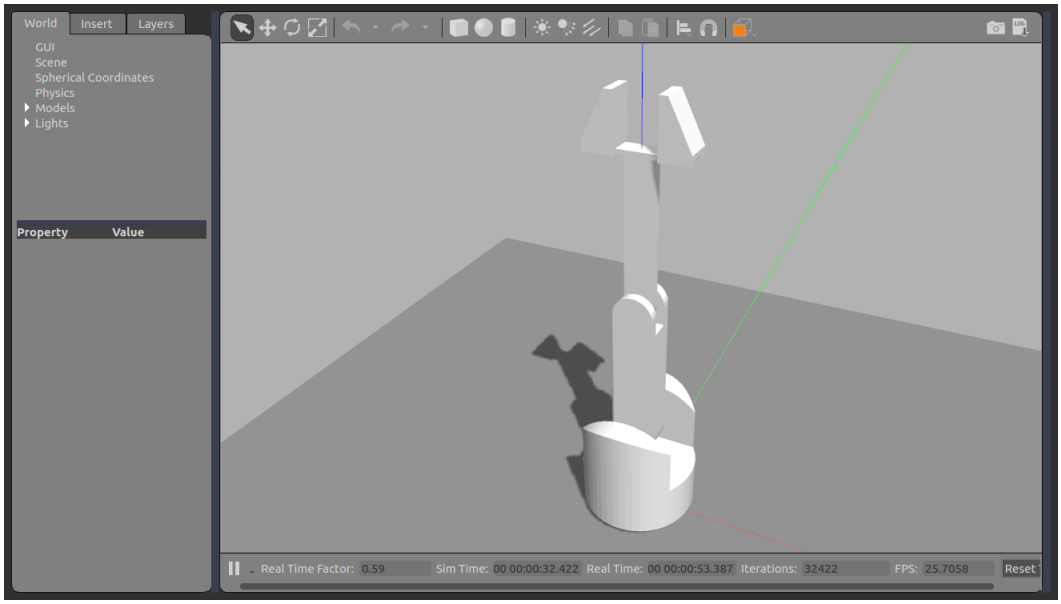

Figura 6: Simulación de robot 2R con Gazebo

Con ánimos de facilitar el manejo de los valores articulares de las juntas se propone hacer uso de la interfaz grafica (GUI), que tiene el paquete *robot_state_publisher*, específicamente el paquete **rqt_publisher**, que nos permite usar sliders desde un GUI para modificar valores de junta (Ver figura 7).

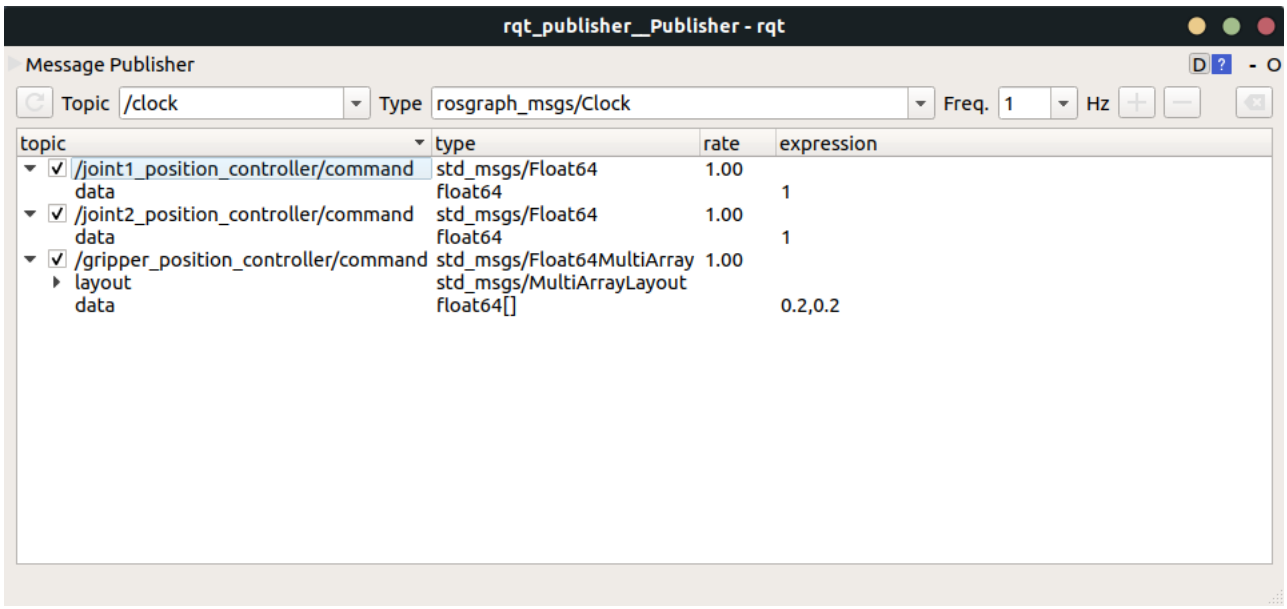

Figura 7: Herramienta gráfica para publicar en tópicos

Para que estas herramienta gráfica se lance en conjunto con la simulación, debemos hacer la siguiente modificación al archivo *spawn.lauch*:

```
<?xml version="1.0" encoding="UTF-8"?>
<launch>

    <!-- Robot Model -->
  <param name="robot_description" command="$(find xacro)/xacro --inorder '$(find 2R_Gazebo)/
      urdf/2r_description_gazebo.xacro'" />
    <arg name="x" default="0"/>
    <arg name="y" default="0"/>
    <arg name="z" default="0"/>

    <!-- Load controllers -->
  <rosparam command="load" file="$(find 2R_Gazebo)/config/joints.yaml"/>

    <!-- Controllers -->
    <node name="controller_spawner" pkg="controller_manager" type="spawner"
          respawn="false" output="screen"
          args="
          joint_state_controller
          joint1_position_controller
          joint2_position_controller
          gripper_position_controller
          --timeout 60">
    </node>


    <!-- Gazebo Initialization -->
    <include file="$(find gazebo_ros)/launch/empty_world.launch">
```

```
        <arg name="use_sim_time" value="true"/>
    </include>

    <!-- Spawn Robot Model -->
    <node name="mybot_spawn"
        pkg="gazebo_ros"
        type="spawn_model"
        output="screen"
        args="-urdf -param robot_description
        -model 2R -x $(arg x) -y $(arg y) -z $(arg z)" />

    <!-- State Publisher -->
    <node name="robot_state_publisher" pkg="robot_state_publisher" type="
        robot_state_publisher"/>

    <!-- rqt -->
    <node name="rqt_publisher" pkg="rqt_publisher" type="rqt_publisher" />

</launch>
```

Referencias

[1] Martinez, A., Fernandez E. *Leaning ROS for robotics programming, PackT Publishing.* 2015.

[2] ROS.org Wiki, *Wiki: urdf (last edited 2019-01-11 01:15:14 by Playfish)*, 2020. *Disponible en: disponible en http://wiki.ros.org/urdf*

[3] Arruda M., *My Robotic Manipulator 1: Basic URDF & RViz*, 2018. *Disponible en: https://www.theconstructsim.com/ros-projects-robotic-manipulator-part-1-basic-urdf-rviz/*