UNIVERSIDAD NACIONAL DE COLOMBIA
DEPARTAMENTO DE INGENIERÍA MECÁNICA Y MECATRÓNICA
ROBÓTICA 2020-2
PROFESOR : Pedro Fabían Cárdenas
MONITOR Jurgen Krejci Muños

Laboratorio 3 - Modelo geométrico directo - ROS + URDF

1. Objetivos

- Comprender las propiedades y el manejo de archivos URDF en ROS para realizar la descripción y simulación de robots en un entorno virtual.
- Crear un paquete de ROS haciendo uso de la herramienta *URDF* y el software RVIZ que permita visualizar el Robot PhantomX Pincher.
- Conectar el modelo visualizado del robot *Phantom X Pincher* con MATLAB usando ROS.
- Modificar la pose del robot haciendo uso del topico *joint_states* desde MATLAB.

2. Requisitos

- Ubuntu versión 16.xx preferible 18.04 LTS con ROS (Melodic o más reciente).
- Espacio de trabajo para *catkin* correctamente configurado.
- Paquete URFD para ROS (Incluye el visualizador RVIZ), Link.
- 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*.

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.

Identificación

- Para información de los componentes y guía de ensamble de robot phantom. Guia de esamble.
- Establezca las longitudes de eslabón para cada articulación del robot *Phantom X Pincher*, para este proceso apóyese en el siguiente documento. Recuerde que la longitud de eslabón es la mínima distancia que conecta dos articulaciones consecutivas. Genere un diagrama como el presentado en la figura 2 las longitudes de los eslabones del robot.

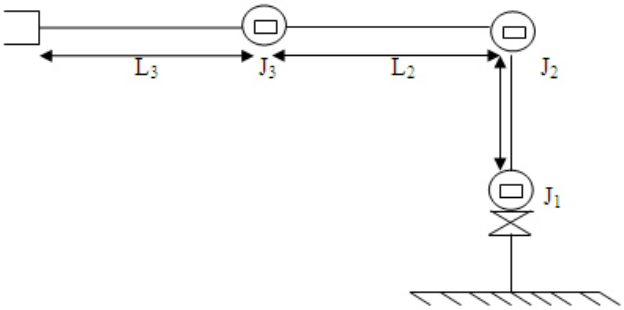

Figura 2: Ejemplo de diagrama

Análisis:

- Con las dimensiones medidas obtenga los parámetros DHmod del robot *Phantom X Pincher*.
- Genere un diagrama del robot donde se vean claramente los sistemas coordenados, incluya las tablas de parámetros articulares, se puede complementar el diagrama que creo en la sección anterior (Utilice algún software de ilustración).

ROS:

- Cree un paquete para poder realizar la visualización del modelo del robot *PhantomX Pincher*, nombre tentativo: ***phantom_urfd***.
- Construir el archivo de descripción del robot *URDF* (extención *.xacro*), en este de debe hacer la descripción del robot, incluyendo la información de las dimensiones y sistemas coordenados de cada eslabon, se propone utilizar para propósitos de la practica geometrías simplificadas (cilindros o prismas), se deja abierta la posibilidad utilizar de modelos CAD del robot para complementar el modelo simulado, se pueden utilizar archivos con extensioń *.stl*. En muchos tutoriales se incluye dentro de la información del *URDF* propiedades físicas (masa e inercia) y de colisiones de las articulaciones, para el proposito de este laboratorio esto no es necesario.
- Crear el archivo ***rviz.launch*** en el cual se haga referencia al archivo *URDF* creado previamente, el cual ejecute la interfaz del software RVIZ, hacer la configuración del software para que se pueda visualizar correctamente el modelo del robot.
- Hacer uso del GUI del paquete *joint_state_publisher_gui* para modificar la posición del modelo visualizado, este paquete se debe incluir dentro del archivo ***rviz.launch***.

Toolbox Peter Corke:

- Utilice el comando SerialLink para crear el robot con los parámetros de su tabla DHmod.

- Obtenga la matriz de transformación homogénea desde la base hasta el efector final, la idea es realizar el análisis en el TCP, este punto puede ser elegido en la mitad de la pinza (Cinemática directa del robot).
- Grafique varias posiciones del robot incluyendo la de HOME utilizando las funciones del toolbox (*SerialLink.plot*).
- **Importante:** Para la correcta orientación del marco de coordenadas del efector final, revise la propiedad *.tool* del robot creado mediante *SerialLink*.

Conexión con MATLAB:

- Cree un *script* que permita publicar en el tópico de valores de junta del modelo, por defecto el nodo del paquete *joint_state_publisher* publica los valores del GUI en el tópico *joint_states*, se recomienda crear un tópico propio para tener control de las articulaciones, este se puede especificar en el archivo ***rviz.launch***.
- Cree un *script* que permita suscribirse al tópico de los estado de cada junta, el *script* debe retornar la configuración de los 4 ángulos de las articulaciones en radianes.

MATLAB + ROS + Toolbox:

- Cree un código en Matlab que envíe la posición en ángulos deseada a cada articulación del robot utilizando las herramientas de ROS, el programa deberá graficar la configuración del robot usando las herramientas del *toolbox*, está configuración deberá coincidir con la obtenida en el robot simulado en RVIZ con la descripción *URDF*, dada.
- Pruebe las siguientes poses generadas a partir de los valores articulares de $q1$, $q2$, $q3$, $q4$ (Recuerde que los valores de configuración se toman respecto a home, para el cual todos los valores articulares son cero):
  1. 0, 0, 0, 0.
  2. -20, 20, -20, 20.
  3. 30,-30, 30, -30.
  4. -90, 15, -55, 17.
  5. -90, 45, -55, 45.

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 el archivo de descripción del modelo del robot *URDF*, el cual debe tener la extensión ***.xacro***.
5. Se deberá incluir un video en el que se muestre el funcionamiento del *script* generado en Matlab que permite modificar la pose del robot visualizado en RVIZ usando ROS. 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: 24/10/2020**.
7. **Bono** de .5 para la el grupo que desarrolle el URDF completo considerando el CAD del phantom y propiedades físicas.

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. Referencia del paquete URDF de ROS, http://wiki.ros.org/urdf
2. Tutorial del manejo del paquete URDF, wiki oficial de ROS (Se recomienda revisar en detalle la sección 2, donde se explica en detalle el manejo de los archivos de descripción URDF), http://wiki.ros.org/urdf/Tutorials
3. Tutorial de manejo del software RVIZ y su configuración, Link
4. ROS Development Studio (Herramienta para trabajar ROS online) http://rds.theconstructsim.com/
5. Referencia del paquete *joint_state_publisher* http://wiki.ros.org/joint_state_publisher
6. Video: Robot State Publisher vs Joint State Publisher Link
7. Especificaciones del robot PhantomX pincher Link
8. Tutorial basico de introducción a ROS http://wiki.ros.org/ROS/Tutorials
9. Tutoriales de manejo de ROS+Matlab Básico, Mensajes, Suscriptores y publicadores
10. Creación del espacio *catkin* Link
11. Canal Yotube LabSIR, se recomiendan los video de instalación de ROS y configuración del espacio de trabajo *catkin* Link LabSIR

5. Ejemplo

En esta sección se presenta un ejemplo de creación de un modelo de robot *URDF* y como hacer su visualización en el software *RVIZ*. Se tomo como ejemplo el robot planar 2R de 2 grados de libertad, se inicia generando la tabla de parámetros DHmod y así obtener los ejes coordenados para cada eslabón.

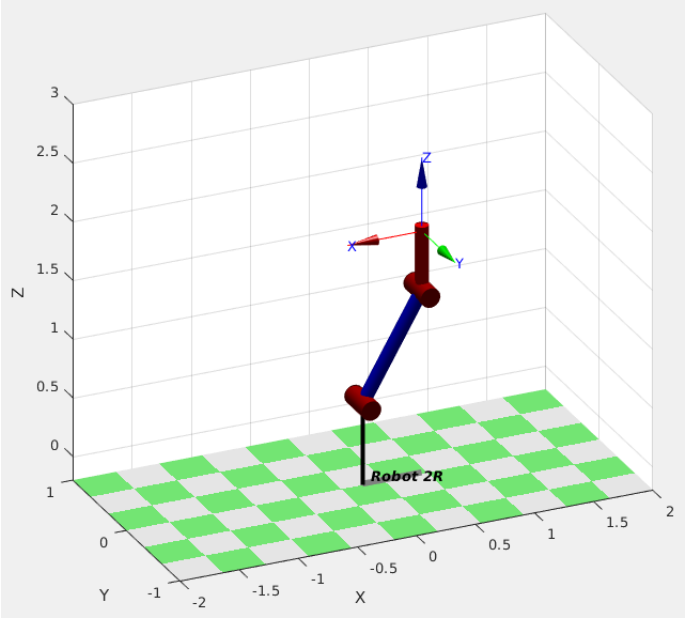

Figura 3: Robot 2R

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

El modelo presentado en la figura 3 de construyo usando las herramientas del Toolbox Peter Corke, con base en la tabla de parámetros presentada, con base en esta información podemos pasar a construir nuestro modelo de robot con URDF en ROS. Antes de iniciar se deben instalar el paquete URDF en ROS y el software RVIZ, para esto podemos ejecutar el siguinete comando en un terminal de linux:

```
$ sudo apt-get install ros-indigo-urdf-tutorial
```

No olvidar cambiar *'indigo'* por la distribución de ROS correspondiente que tenga instalada en su sistema operativo. Este comando nos va permitir cargar modelos de descripción URDF en el software RVIZ, para comprobar la instalación podemos lanzar un paquete con un ejemplo (si no se tiene refereciada la carpeta de instalación del paquete *URDF* en el archivo *setup.bash*, ejecutar la primera linea):

```
$ roscd urdf_tutorial
$ roslaunch urdf_tutorial display.launch model:='$(find urdf_tutorial)/urdf/01-myfirst.urdf'
```

Una vez se ejecute este comando se debería lanzar el paquete y la ventana de ejecución de *RVIZ* se debería mostrar el pantalla con un modelo que consiste de 1 cilindro (Ver fig. 4).

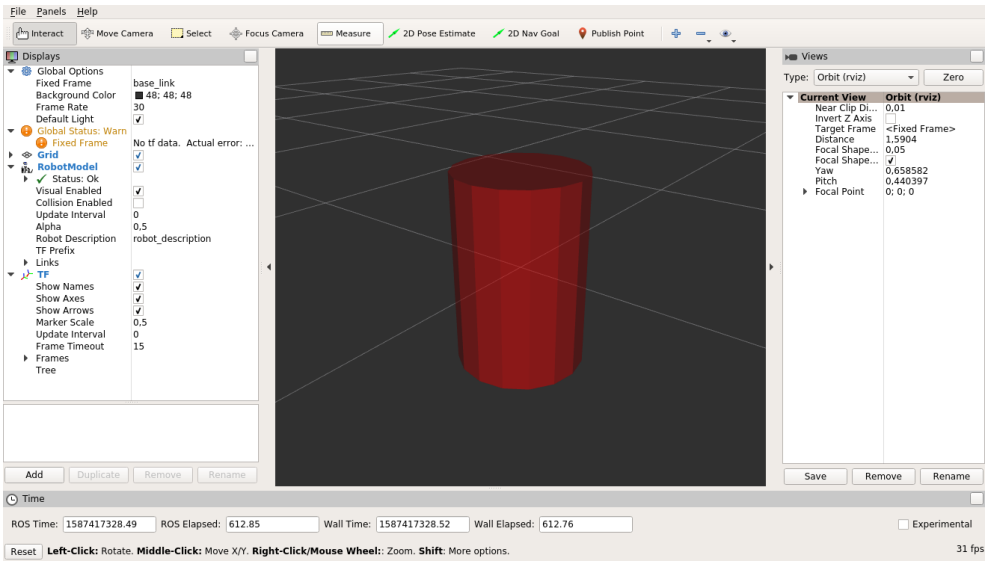

Figura 4: Interfaz grafica de *RVIZ*

Si la ventana presentada se logra visualizar de forma adecuada significa que ya se cuenta con los paquetes necesarios para el desarrollo de la practica. Los pasos descritos a continuación se encuentran explicados en profundo detalle en los recursos 2 y 3 de la sección anterior, se sugiere revisarlos en profundo detalle en conjunto con esta guía.

A continuación debemos crear un paquete de ROS para nuestra simulación, para esta parte se debe haber realizado la configuración correcta del espacio de trabajo *catkin* (Revisar referencia 10), desde el espacio de trabajo *catkin* vamos a abrir una consola y vamos a crear nuestro paquete.

```
$ cd catkin_ws/src
$ catkin_create_pkg 2r_urdf urdf
```

Una vez realizada la creación del paquete, nuestro dentro del espacio de trabajo vamos a ir a la carpeta *2r_urdf* y vamos a crear dos nuevas carpetas (*launch* y *urdf*) conforme al siguiente diagrama:

```
catkin_ws
├── build
├── devel
└── src
```

```
2R_URDF
    launch
    urdf
    CMakeLists.txt
    package.xml
CMakeLists.txt
```

Para esto podemos usar el siguiente comando en consola desde la carpeta *2R_URDF*:

```
$ mkdir launch urdf
```

A continuación vamos a crear el archivo que va a contener el modelo *URDF* de nuestro robot, este archivo se escribe usando el lenguaje de macros *Xacro* (macros de XML), este nos permite describir facilmente los elementos de nuestro modelo robot como eslabones, ejes coordenados, masas, inercias y apariencia, para mayor detalle en la sintaxis y construcción de modelos *URDF* usando *Xacro* referirse a la sección 2.1 y 2.4 del tutorial de URDF para ROS.

Vamos a acceder a la carpeta ***'/catkin_ws/src/2r_urdf/urdf'*** y alli vamos a crear una archivo llamado ***'2r_descrption.xacro'*** para hacer esto podemos desde la consola (en la carpeta) usar el siguiente comando:

```
$ touch 2r_descrption.xacro
```

Usando algún editor de texto (por defecto en linux incluye editores como **gedit** o desde consola **nano**) podemos modificar los contenidos de este, a continuación se presenta el contenido de este archivo para el caso del robot 2R:

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

<robot name="2R" xmlns:xacro="http://www.ros.org/wiki/xacro">

    <material name="orange">
        <color rgba="1 0.5 0 1"/>
    </material>


    <link name="base_link">
        <visual>
            <origin rpy="0 0 0" xyz="0 0 0.25"/>
            <geometry>
                <box size="1 1 0.5"/>
            </geometry>
        <material name="orange"/>
        </visual>
    </link>

    <joint name="joint_1" type="revolute">
        <axis xyz="0 0 1" />
        <limit effort="1000.0" lower="-3.14" upper="3.14" velocity="0.5" />
        <origin rpy="0 -1.57 -1.57" xyz="0 0 0.5"/>
        <parent link="base_link"/>
        <child link="link1"/>
    </joint>

    <link name="link1">
        <visual>
            <origin rpy="0 1.57 0" xyz="0.5 0 0"/>
            <geometry>
```

```
            <cylinder radius="0.1" length="1"/>
        </geometry>
    <material name="orange"/>
    </visual>
</link>

<joint name="joint_2" type="revolute">
    <axis xyz="0 0 1" />
    <limit effort="1000.0" lower="-3.14" upper="3.14" velocity="0.5" />
    <origin rpy="0 0 0" xyz="1 0 0"/>
    <parent link="link1"/>
    <child link="link2"/>
</joint>

<link name="link2">
    <visual>
        <origin rpy="0 1.57 0" xyz="0.5 0 0"/>
        <geometry>
            <cylinder radius="0.1" length="1"/>
        </geometry>
    <material name="orange"/>
    </visual>
</link>

</robot>
```

Como se puede ver en el archivo anterior se hace la creación de 3 eslabones del robot *base_link, link1* y *link2* asi como 2 articulaciones *joint_1* y *joint_2*, para una correctam construccion de estos modelos es clave hacer una buena construcción de la tabla DHmod, tener en cuenta que las rotaciones de los ejes coordenados de cada eslabón están expresadas como ángulos *r,p,y* (Roll, Pitch, Yaw), para el ejemplo se asumio que cada eslabon del robot tiene una longitud de $1m$, la forma de estos se aproximo a cilindros, la base del robot tiene $0{,}5m$ de altura.

Con nuestro *URDF* ya creado procedemos a generar el archivo *.launch* que va a lanzar la interfaz de *RVIZ* y cargar el modelo creado. Para esta tarea nos dirigimos en la consola a la carpeta ***'/catkin_ws/src/2r_urdf/launch'*** y allí vamos a crear una archivo llamado ***'rviz.launch'***:

```
$ touch rviz.launch
```

A continuación se presenta el contenido que debe tener este archivo:

```
<launch>
    <param name="robot_description" command="$(find xacro)/xacro --inorder '$(find 2r_urfd)/
        urdf/2r_description.xacro'"/>
    <node name="robot_state_publisher" pkg="robot_state_publisher" type="robot_state_publisher"
        />
    <node name="rviz" pkg="rviz" type="rviz"/>
    <node name="joint_state_publisher" pkg="joint_state_publisher" type="joint_state_publisher"
         >
        <param name="use_gui" value="True"/>
    </node>
</launch>
```

```
catkin_ws
├── build
├── devel
└── src
    └── 2R_URDF
```

```
├── launch
│   └── rviz.launch
├── urdf
│   └── 2r_descrption.xacro
├── CMakeLists.txt
└── package.xml
└── CMakeLists.txt
```

El codigo presentado anteriormente va a permitir ejecutar el visualizador *RVIZ* con nuestro modelo, como se puede ver se incluyen paquetes como **'robot_state_publisher'** y *joint_state_publisher*, estos van a crear nodos y tópicos encargados de gestionar la información relacionada con las posiciones y valores que puedan tener las articulaciones y ejes coordenados de nuestro robot. Al terminar deberíamos tener el arbol de archivos mostrado previamente.

Finalmente nos resta compilar el paquete creado para poder hacer uso del mismo, para esto nos dirijamos a las carpeta del espacio de trabajo *catkin_ws* y ejecutamos el siguiente comando:

```
$ catkin build
```

Si el proceso finaliza sin errores, deberíamos estar listos para ejecutar el paquete creado, para esto ejecutamos (deberíamos ver como se despliega la ventana de *RVIZ*) usando el comando:

```
$ roslaunch 2r_urdf rviz.launch
```

Importante, una vez se ejecute el comando veremos como carga la ventana pero no aparece ningun modelo, esto ocurre debido a que la configuración de la configuración por defecto de interfaz de *Rviz* no tiene creados los elementos de simulación, para ver nuestro modelo correctamente debemos hacer 3 ajustes:

- Cambiar **Fixed Frame** a *'base_link'*
- Hacer click **'Add'** y añadir un elemento *'TF'*
- Hacer click **'Add'** y añadir un elemento *'Robot Model'*

Una vez realizados estos cambios se deberá poder visualizar el robot creado, tal y como se muestra en la figura 5, adicionalmente el paquete también ejecuta *'joint_state_publisher'*, paquete que desde una pequeña ventana nos permite cambiar los valores de nuestro articulares robot por medio de sliders.

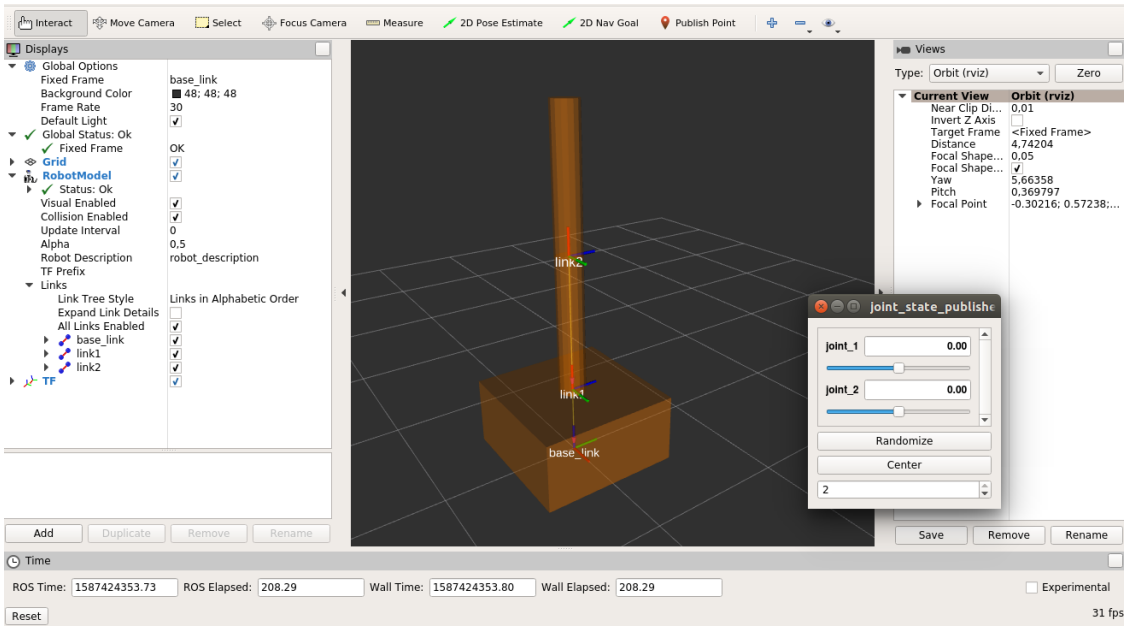

Figura 5: Modelo del robot 2R en *RVIZ*

Con el objetivo de que en futuras ocasiones no tengamos que volver a hacer la configuración de la interfaz podemos hacer click en *'File'* y luego en *'Save config'* (si el dialogo pregunta por ubucación de guaradado usar la carpeta *'/launch'*, guardar como *'config.rviz'*) esto guardara la configuración en archivo tipo *.rviz* para que este siempre quede referenciado al ejecutar el paquete debemos modificar el archivo *'rviz.launch'* de la siguiente forma:

```
<launch>
    <param name="robot_description" command="$(find xacro)/xacro --inorder '$(find 2r_urdf)/
        urdf/2r_description.xacro'"/>
    <node name="robot_state_publisher" pkg="robot_state_publisher" type="state_publisher"/>
    <node name="rviz" pkg="rviz" type="rviz" args=" -d $(find 2r_urdf)/launch/config.rviz"/>
    <node name="joint_state_publisher" pkg="joint_state_publisher" type="joint_state_publisher"
         >
        <param name="use_gui" value="True"/>
    </node>
</launch>
```

De forma final y para facilitar el ejercicio a realizar en Matlab, se hace la invitación a explorar en un nuevo terminal (Mientras se ejecuta el paquete creado) los topicos y nodos que estan ejecutandose en ROS, usar los comandos:

```
  $ rostopic list
  $ rosnode list
```

Entre los topicos, cabe destacar el tópico *'/joint_states'*, en este se publican los valores articulares que las articulaciones tienen en el momento, si se exploran los nodos que publican información en ese tópico podemos hallar el nodo *'/joint_state_publisher'* este nodo corresponde a un paquete con el mismo nombre, el cual por medio de la GUI (ventana pequeña) que acompaña a *RVIZ* nos permite cambiar los valores de las articulaciones. Los mensajes publicados en *'/joint_states'* tienen el siguiente formato:

```
---
header:
  seq: 27227
  stamp:
    secs: 1587426868
    nsecs: 493113040
  frame_id: ''
name: [joint_1, joint_2]
position: [0.0, 0.0]
velocity: []
effort: []
---
```

Como se puede ver, en principio no podemos publicar valores directamente desde Matlab en el tópico *'/joint_states'*, esto debido a que el nodo *'/joint_state_publisher'* ya se encuentra haciendo esta tarea de forma constante. Para solucionar este problema el paquete *'/joint_state_publisher'* permite que su nodo se suscriba a un tópico nuevo creado desde el archivo *rviz.launch*, este nuevo tópico puede llevar cualquier nombre que le especifiquemos. La dinámica consiste en que a este nuevo tópico creado se suscribe el nodo *'/joint_state_publisher'* el cual va a publicar la la información de este en el tópico *'/joint_states'* de nuestro modelo en *RVIZ*, con esto podremos modificar por medio de publicadores la pose de nuestro modelo. El código del archivo *rviz.launch* modificado para realizar esta tarea se muestra a continuación (como se puede ver el tópico creado lleva el nombre ***2R/my_values/joint_states***):

```
<launch>
    <param name="robot_description" command="$(find xacro)/xacro --inorder '$(find 2r_urdf)/
        urdf/2r_description.xacro'"/>
    <node name="robot_state_publisher" pkg="robot_state_publisher" type="state_publisher"/>
    <node name="rviz" pkg="rviz" type="rviz" args=" -d $(find 2r_urdf)/launch/config.rviz"/>
```

```
    <node name="joint_state_publisher" pkg="joint_state_publisher_gui" type="
        joint_state_publisher_gui">
        <rosparam param="source_list">["2R/my_values/joint_states"]</rosparam>
    </node>
</launch>
```

Para mayores detalles en el funcionamiento del paquete *'/joint_state_publisher'* consultar Link 1 y Link 2. Como comentario final, el ejercicio en Matlab se debe centrar en generar la lógica que me permita publicar mensajes en el tópico creado, para ello se recomiendo revisar en la sección de Recursos las referencias 8, 9 y 10.

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/*