Laboratorio 2: Modelo Geométrico Directo - ROS+URDF

Luis Antonio Zuluaga Ramirez *, Daniel Andrés Rojas Paredes† and Álvaro Chaves Ladino‡
Departamento de Ingeniería Mecánica y Mecatrónica,
Universidad Nacioanl de Colombia
Bogotá D.C., Colombia
Email: *luazuluagara@unal.edu.co, † daarojaspa@unal.edu.co, ‡achavesl@unal.edu.co

I. Ejercicio de laboratorio

I-A. *Identificación*

- Establezca las longitudes de eslabón para cada articulación del robot *Phantom X Pincher*. Recuerde que la longitud de eslabón es la mínima distancia que conecta dos articulaciones consecutivas. Genere un diagrama como el presentado en la figura **??**. Se incluye una representación de los sistemas sobre el robot real, se asume que las tramas 0 a 2 se encuentran sobre el mismo punto, pero para facilitar la visualización y comprensión se han colocado sobre los ejes correspondientes.

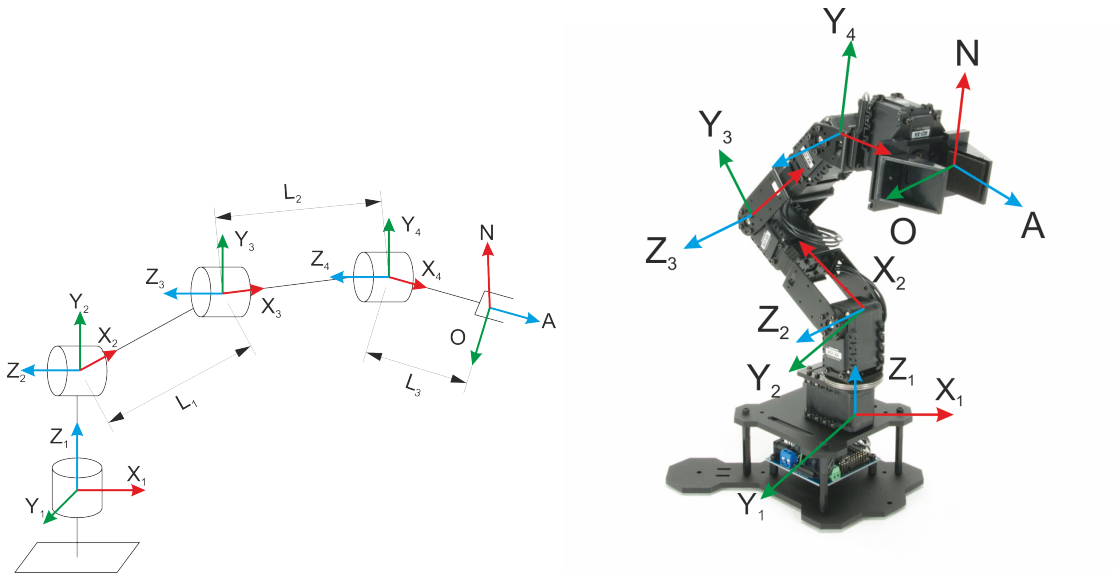


Figura 1: 1) Diagrama Phantom X Pincher 2) Tramas en el robot

I-B. *Análisis*

- Con las dimensiones medidas obtenga los parámetros DHmod del robot *Phantom X PIncher*:

i	α_{i-1}	a_{i-1}	d_i	θ_i
1	0	0	0	q_1
2	90	0	0	q_2
3	0	105	0	q_3
4	0	105	0	q_4

Cuadro I: Tabla de parámetros DHmod

- Genere un diagrama del robot donde se vean claramente los sistemas coordenados, incluya las tablas parámetros articulares.

I-C. *ROS*

- **Cree un paquete para poder realizar la visualización del modelo del robot *Phantom X Pincher*.**

  Se creo un paquete dentro del directorio *catkin_ws/src* con el nombre *phantom_urdf*; dentro de este paquete se crearon

3 subcarpetas. La carpeta launch contiene dos archivos, *rviz.launch* que ejecuta el robot dentro del programa rviz y, *spawn.launch* importa el robot dentro de gazebo. La carpeta meshes, contiene los modelos 3D de las partes que conforman al robot en formato *.stl*. Por ultimo la carpeta *urdf* incluye dos archivos relacionados entre si para la construcción del robot.

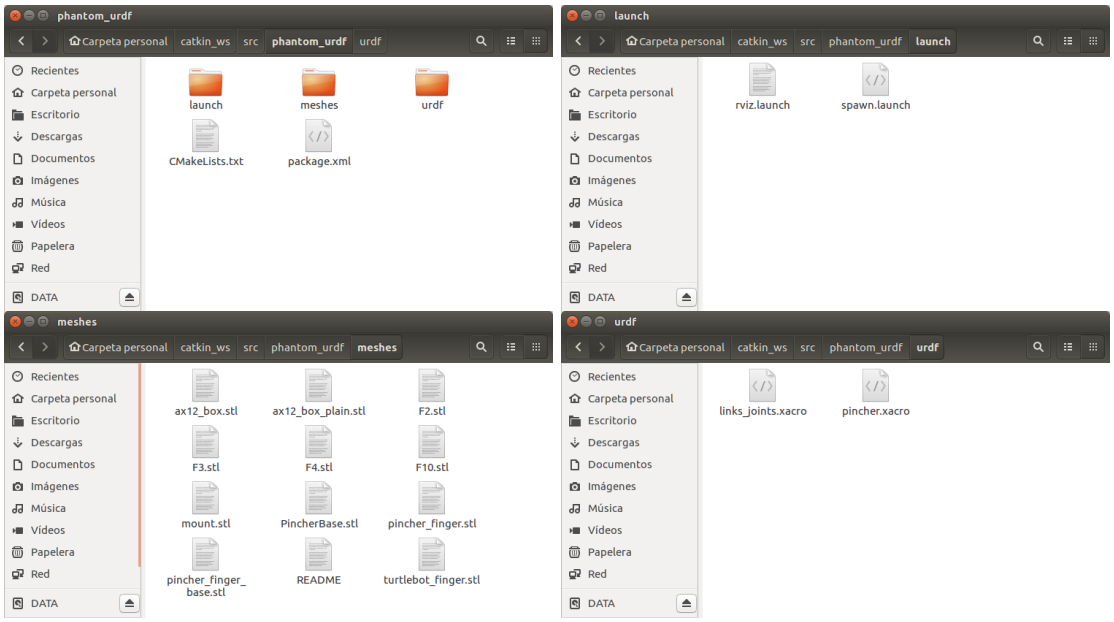


Figura 2: Carpetas contenidas dentro del paquete *phantom_urdf*

- **Construir el archivo de descripción del robot URDF:**

Para la construcción del archivo de descripción del robot se utilizaron las dimensiones que se muestran en la figura

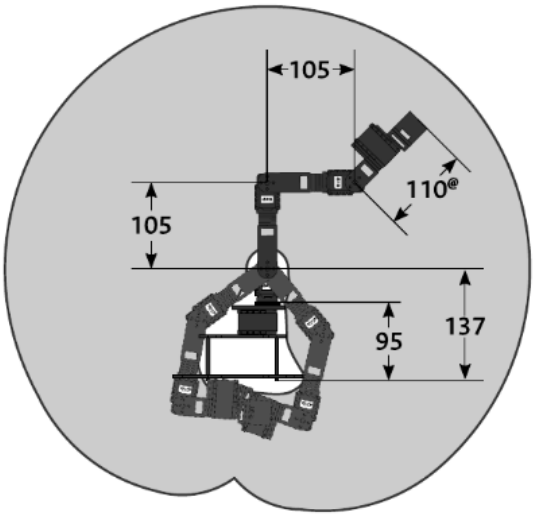


Figura 3: Dimensiones de eslabones en el robot *Phantom X Pincher* [2]

Además se hizo uso de modelos CAD tomados de la pagina web Bioloid [1] (figura 4) para el ensamble del robot se siguieron las instrucciones de PhantomX Pincher Robot Arm Assembly Guide [3]

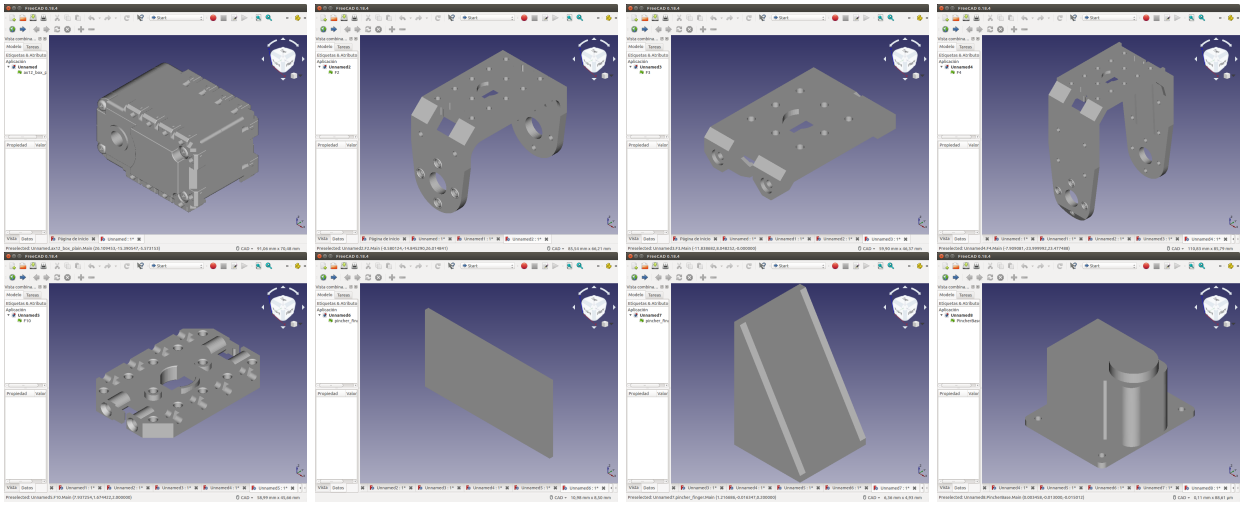


Figura 4: 1)Motor AX-12A. 2)Eslabon Efector Final F2-Bracket. 3)Fijador Motor F3-Bracket. 4)Eslabon Cuerpo F4-Bracket. 5)Espaciador. 6)Base efector final. 7)Dedo Gripper. 8)Base del robot

Ya con todos los elementos listos primero se crea un archivo llamado *links_joints.xacro* en el que se crean las clases para cada elemento CAD como para las uniones. Para comprender mejor como se expresan los elementos en el archivo *.urdf* ver figura 5

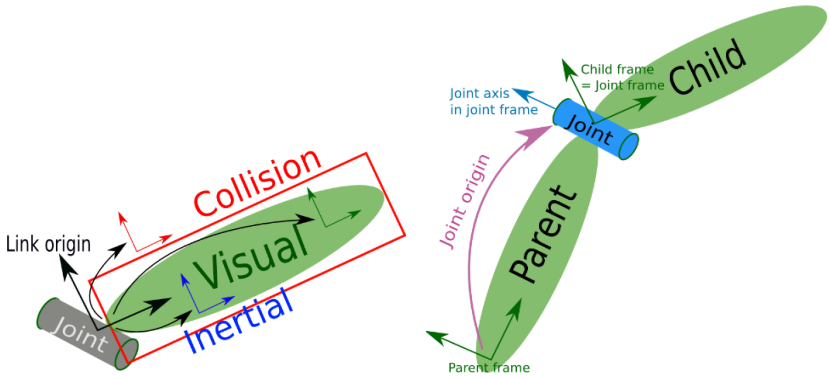


Figura 5: 1)Descripción de Link(eslabón). 2)Descripción unión(joint) [4]

Como se menciono anteriormente se crearon dos archivos para la descripción del robot, el primero es *links_joints.xacro* en el cual se ingresaron cada uno de los elementos como una macro que contiene las características que hacen parte de una unión o un link. por ejemplo en la figura 6 se muestra una pequeña porción del código, en donde se pueden observar dos macros; la primera esta relacionada con las uniones que tendrá el robot, en la primera linea de esta macro se establece el nombre con el que sera llamado en el documento principal para describir cada unión que hay en el robot, y los parámetros que esta recibe:

- **name**: Es como se llamara cada articulación
- **type**: Tipo de articulación (revolute,continuous,prismatic,fixed,floating,planar)
- **axis_xyz**: Sobre que eje funciona la unión, por ejemplo una revolute que gira al rededor del eje Z *axis_xyz="0 0 1"*
- **origin_rpy**: Representa la rotación sobre los ángulos fijos en radianes: roll(x) pitch(y) yaw(z)
- **origin_xyz**: Representa el offset en metros
- **parent**: Link padre (link anterior)
- **child**: Link hijo (link siguiente

La macro correspondiente a la base del robot esta construida de igual forma que para todas las demás piezas que componen al *Phantom*; esta sección se puede dividir en tres partes en la primera parte se describen las propiedades de inercia que de la pieza correspondiente, en donde se tiene en cuenta el peso y el tensor del elemento; la parte visual simplemente hace referencia a la geometría visible de ese eslabón y en donde esta ubicada con un offset y una orientación, en este caso como se utilizaron archivos CAD *.stl* se escribe la dirección de ubicación en el proyecto de la pieza requerida; la parte de colisión es muy similar a la parte visual, es porque esta representa la geometría que define la forma de colisión del enlace.

*links_joints.xacro (~/catkin_ws/src/phantom_urdf/urdf) - gedit

Abrir Guardar

rviz.launch | *pincher.xacro | *links_joints.xacro | spawn.launch | 2r_description.xacro

```
<?xml version="1.0"?>
<robot xmlns:xacro="http://www.ros.org/wiki/xacro">

  <!-- Macro para las uniones-->
  <xacro:macro name="j" params="name type axis_xyz origin_rpy origin_xyz parent child lower upper"><!--Parametros que recibe la macro-->
    <joint name="${name}" type="${type}">  <!--TIpo y nombre de la union-->
      <axis xyz="${axis_xyz}" />          <!-- Eje de funcionamiento-->
      <limit effort="1000.0" lower="${lower}" upper="${upper}" velocity="0.5"/> <!--Limites del movimiento-->
      <origin rpy="${origin_rpy}" xyz="${origin_xyz}" /> <!-- Coordenadas y orientacion-->
      <parent link="${parent}" /> <!--Link anterior-->
      <child link="${child}" />   <!-- LInk siguiente-->
    </joint>
  </xacro:macro>

  <!-- Macro para la base del robot-->
  <xacro:macro name="pincher_base" params="name">  <!--Parametros que recibe la macro-->
    <link name="${name}">
      <inertial>     <!-- Propiedades de inercia-->
        <mass value="0.055"/>   <!--Peso-->
        <origin xyz="0 0 0.0285"/>          <!--Tensor-->
        <inertia ixx="0.000017012" ixy="0.0" ixz="0.0"
                 iyy="0.000013258" iyz="0.0"
                 izz="0.000009483"/>
      </inertial>
      <visual> <!--Parte visible-->
        <origin xyz="0 0 0.0285" rpy="0 0 0"/>
        <geometry>
          <mesh filename="package://phantom_urdf/meshes/PincherBase.stl" scale="1.5 1.5 1.5"/><!--Ubicacion del archivo-->
        </geometry>
        <material name="Black"/>
      </visual>
      <collision>
        <origin xyz="0 0 0.0285" rpy="0 0 0"/>
        <geometry>
          <mesh filename="package://phantom_urdf/meshes/PincherBase.stl" scale="1.5 1.5 1.5"/>
        </geometry>
      </collision>
    </link>
  </xacro:macro>
```

XML Anchura de la pestaña: 8 Ln 5, Col 137 INS

Figura 6: Archivo de descripcion de las partes que componen al robot *(links_joints.xacro*

En la figura 11 se muestra una sección del código del segundo archivo *pincher.xacro*, como se puede ver en la linea 5, para el correcto funcionamiento de este archivo es necesario vincular el archivo anteriormente descrito *links_joints.xacro*; en las siguientes lineas solo se muestra como se utilizan las diferentes macros definidas, una pequeña aclaración para la descripción del offset consiste en que una vez definida la ubicación de la unión, el siguiente link se dibujara una mitad de la pieza sobre la ubicación del link, y la otra mitad por debajo, es por ello que es necesario aplicar un offset correspondiente a la mitad de la altura de la pieza, por ejemplo la unión *j1* se ubica a $0{,}057m$ ya que es la altura de la base del robot, para poder dibujar el motor_1 correctamente es necesario escribir un offset en Z de $0{,}0205m$*origin_xyz="0 0 0.0205"* ya que la altura del motor es de $0{,}041m$.

Todas las macros correspondientes a las piezas siempre funcionan igual en donde se entregan tres parámetros, nombre de identificación, orientación en radianes y utilizando ángulos fijos (rool,pitch,yaw) y el offset; mientras que para las uniones depende del tipo de unión, por ejemplo para una unión fija, la mayoría de valores se vuelven cero ya que se debe restringir el movimiento, para las demás uniones si hay que completar el resto de los parámetros para obtener un correcto funcionamiento.

*pincher.xacro (~/catkin_ws/src/phantom_urdf/urdf) - gedit

Abrir Guardar

rviz.launch | *pincher.xacro | links_joints.xacro | spawn.launch | 2r_description.xacro

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

<robot name="phantom" xmlns:xacro="http://www.ros.org/wiki/xacro"> <!-- Se importa la macro links_joints.urdf-->

  <!-- Se incluye el archivo con los links y uniones predefinidas en el archivo links_joints.xacro-->
  <xacro:include filename="$(find phantom_urdf)/urdf/links_joints.xacro" />

  <!-- Base del robot-->
  <pincher_base name="base_link"/> <!-- Se inserta la base del robot -->

   <!--Union fija de la base con m1-->
  <j name="j0" type="fixed" origin_xyz="0 0 0.057" origin_rpy="1.570795 0 1.570795"
     parent="base_link"  child="m1" axis_xyz="0 0 0" lower="0" upper="0"/>

  <!--Ubicacion m1-->
  <motor name="m1" xyz="0 0.0205 0" rpy="0 0 0" /> <!--Motor que aplicara rotacion sobre el eje Z-->

  <!--Vinculo rotacional sobre el eje Z, primer grado de liberta-->
  <j name="j1" type="revolute" origin_xyz="0 0.035 0" origin_rpy="-1.570795 1.570795 3.14159"
     parent="m1"  child="f3_1" axis_xyz="0 0 1" lower="-3.14" upper="3.14"/>

  <!-- Ubicacion de la pieza de union de los motores F3-->
  <F3 name="f3_1"  rpy="0 0 0" xyz="0 0 -0.0045"/><!--arm_shoulder_lift_servo_joint-->

  <!--Union Fija que une al motor 2 a la pieza F3-->
  <j name="j2" type="fixed" origin_xyz="0 0 -0.0205" origin_rpy="0 3.14159 0"
     parent="f3_1"  child="m2" axis_xyz="0 0 0" lower="0" upper="0"/>

  <motor name="m2" rpy="0 0 0" xyz="0 0 0.025" /> <!--Motor que aplicara una rotacion sobre el eje Y-->

  <!--Union rotacional sobre el eje Y, segundo grado de libertad-->
  <j name="j3" type="revolute" origin_xyz="0 0 0.025" origin_rpy="0 0 0"
     parent="m2"  child="f4_1" axis_xyz="0 1 0" lower="-3.14" upper="3.14"/>

  <!--Eslabon que une m2 con m3-->
  <F4 name="f4_1" xyz="0 0 0.0" rpy="0 0 0"/><!--arm_shoulder_lift_link-->

  <!--Vinculo fijo, union eslabon 1 con pieza F10-->
  <j name="j4" type="fixed" origin_xyz="0 0 0.052" origin_rpy="0 0 0"
     parent="f4_1"  child="f10_1" axis_xyz="0 0 0" lower="0" upper="0"/>
```

XML Anchura de la pestaña: 8 Ln 3, Col 113 INS

Figura 7: Archivo *(pincher.xacro)* que describe la ubicacion y union de las piezas para el robot *Phantom X Pincher*

- **Crear el archivo *rviz.launch* en el cual se haga referencia al archivo URDF creado previamente**

  Con este archivo se hace referencia al archivo *pincher.xacro* para que se ejecute dentro de la aplicacion rviz, además también se hace el llamado a la GUI del paquete *joint_state_publisher* la cual nos ayudara a mover las articulaciones del robot de forma fácil mediante unos botones deslizantes.

rviz.launch (~/catkin_ws/src/phantom_urdf/launch) - gedit

Abrir Guardar

```
<launch>
  <param name="robot_description" command="$(find xacro)/xacro --inorder '$(find phantom_urdf)/urdf/pincher.xacro'"/>
  <node name="robot_state_publisher" pkg="robot_state_publisher" type="state_publisher"/>
  <node name="rviz" pkg="rviz" type="rviz" args="-d $(find phantom_urdf)/launch/config.rviz"/>
  <node name="joint_state_publisher" pkg="joint_state_publisher" type="joint_state_publisher">
    <param name="use_gui" value="True"/>
    <rosparam param="source_list">["phantom/my_values/joint_states"]</rosparam>
  </node>
</launch>
```

Texto plano Anchura de la pestaña: 8 Ln 7, Col 5 INS

Figura 8: Archivo *rviz.launch* muestra el robot mediante rviz

- **Hacer uso del GUI del paquete *joint_state_publisher* para modificar la posición del modelo visualizado**

  Como se puede observar en la GUI de *joint_sate_publisher* hay seis articulaciones *(j1,j3,j10,j17,j21,j22* que pueden moverse pero el robot solo tiene 4 DOF, esto se debe a que las dos ultimas articulaciones *(j21,j22)* corresponden a los dedos del gripper.

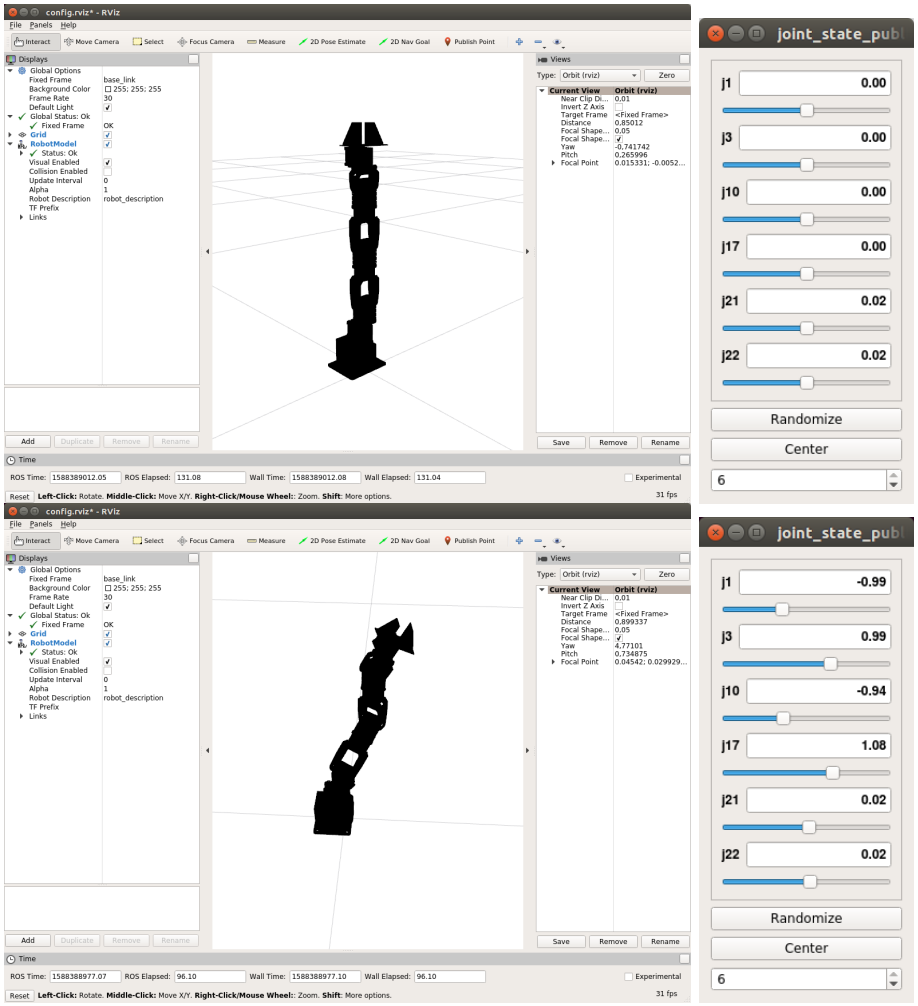


Figura 9: Posición del robot, antes y después de usar *joint_state_publisher*

I-D. Toolbox Peter Corke

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

Para la creación del robot se creo eslabón por eslabón con ayuda de la matriz de parámetros, posteriormente se concatenaron todos haciendo énfasis en que se utilizo la DHmod . posteriormente se roto y traslado el TCP a la posición correspondiente.

```
L1 = 10.5; L2 = 10.5; L3 = 11.0;
theta1 = 0; theta2 = 0; theta3 = 0; theta4 = 0;

%         dh(thetai,      di,      ai-1,   alpha-1,   sigma, offset)

dh1(1,:)= [ theta1,        9.5,       0,         0,      0,      0];
dh1(2,:)= [ theta2,        0,         0,       pi/2,     0,      0];
dh1(3,:)= [ theta3,        0,        L1,         0,      0,      0];
dh1(4,:)= [ theta4,        0,        L2,         0,      0,      0];

% Eslabones
L(1) = Link(dh1(1,:), 'modified');
L(2) = Link(dh1(2,:), 'modified');
L(3) = Link(dh1(3,:), 'modified');
L(4) = Link(dh1(4,:), 'modified');

Phantom = SerialLink(L,'name','Phantom X Pincher');

Phantom.tool = transl(L3,0,0)*troty(pi/2);
```

Figura 10: CREACION ROBOT

- **Obtenga la matriz de transformación homogénea desde la base hasta el efector final, la idea es realizar elanálisis en el TCP, este punto puede ser elegido en la mitad de la pinza (Cinemática directa del robot).**

  Para esto se hizo uso de la funcion **fkine** de **SerialLink** por la longitud del resultado a pesar de los varios intentos por simplificarlo se omite su muestra en este informe sin embargo se adjunta el código que se corrió

```
%% FORWARD KINEMATICS
syms q1 q2 q3 q4
format
format compact
T=simplify(vpa(Phantom.fkine([q1 q2 q3 q4])));
```

Figura 11: cinemática directa

- **Grafique varias posiciones del robot incluyendo la de HOME utilizando las funciones del toolbox (SerialLink.plot).**

  para esta gráfica se hizo uso de ciclos for para hacer un movimiento mas fluido y automático del modelo

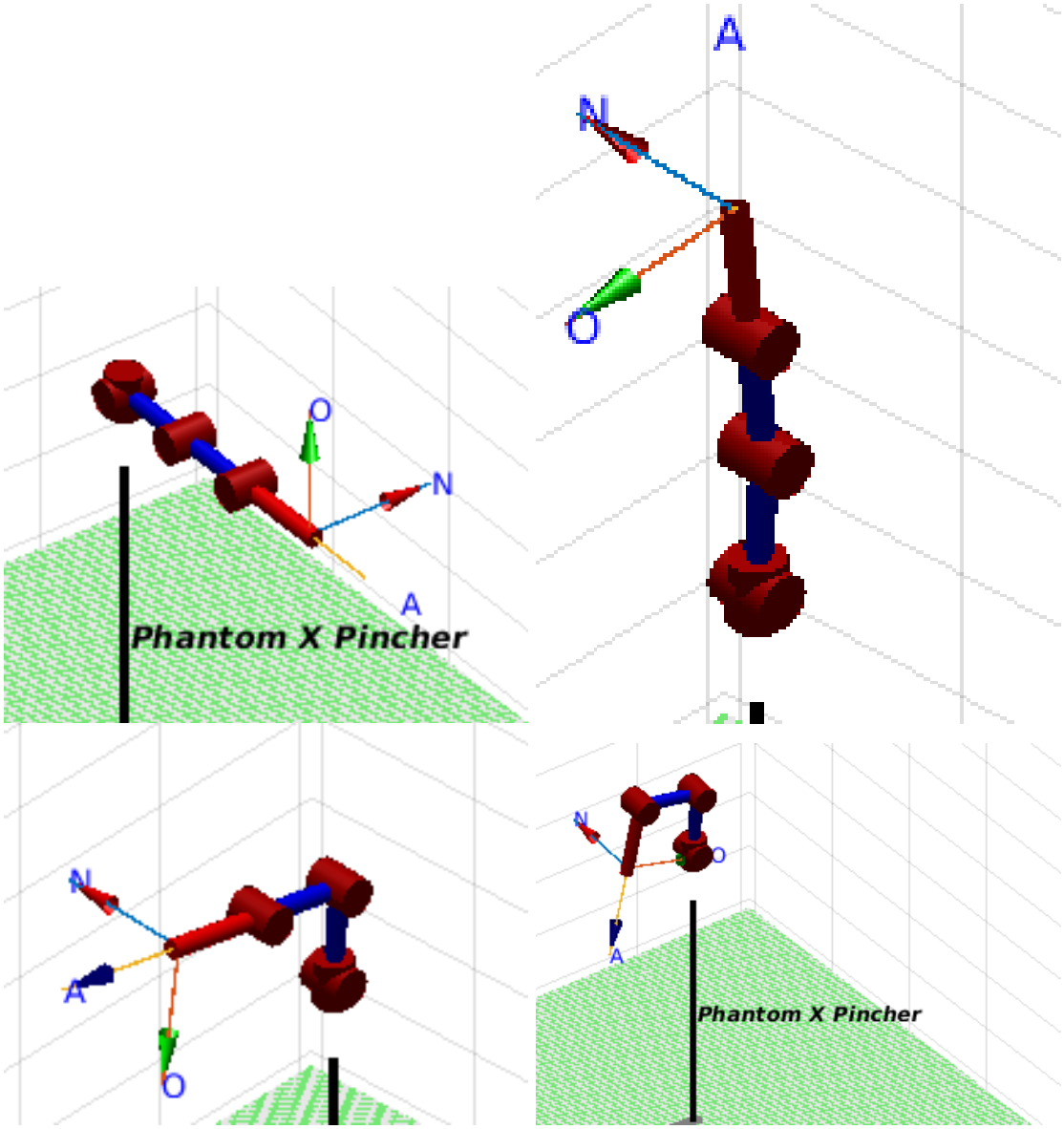
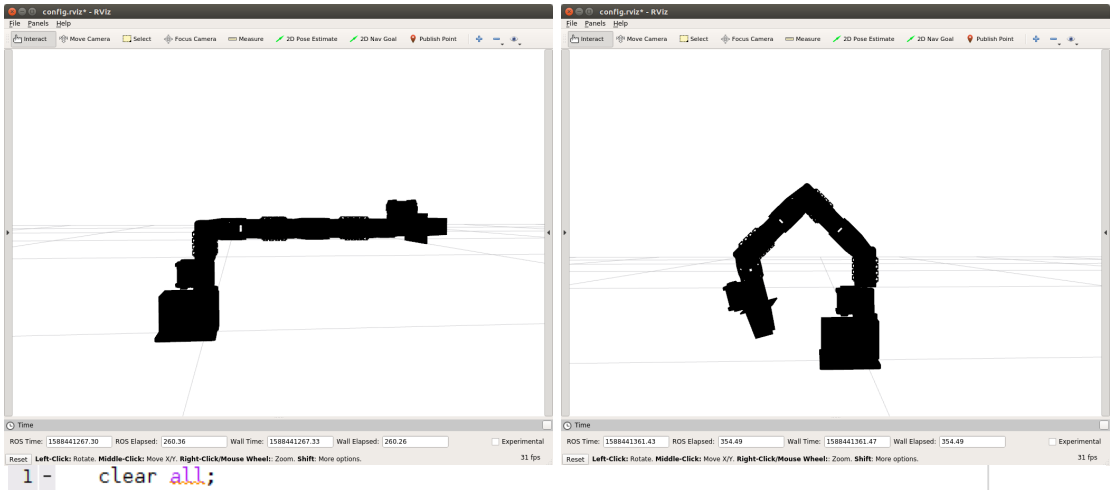


Figura 12: movimiento del en matlab desde home *phantom_urdf*

I-E. Conexión con MATLAB

- **Cree un script que permita publicar en el tópico de valores de junta del modelo**

Se creo un tópico *(phanotom/my_value/joint_states)* para tener un mejor control de las articulaciones, esto se puede ver en la figura 8. El script de MATLAB se realiza una suscripción al tópico */joint_states* y se publica mediante el tópico creado; para poder enviar el mensaje a cada articulación, se crea un ciclo for que recorre los vectores que contienen los nombres que se usaron para cada articulación móvil en el archivo de descripción urdf (figura 11) y el vector de rotación que contiene la rotación que se ejecutara en cada articulación.



```
clear all;
rosinit
rostopic list;
rostopic info /phantom/my_values/joint_states;
suscriptor=rossubscriber("/joint_states","sensor_msgs/JointState");
publicador=rospublisher('/phantom/my_values/joint_states','sensor_msgs/JointState');
msg= rosmessage('sensor_msgs/JointState');
nombres=["j1","j3","j10","j17"];
rotacion = [pi -pi/4 pi/2 pi/3];
for i=1:4
    msg.Name=nombres(i)
    msg.Position=rotacion(i);
    msg.Velocity=0.5;
    send(publicador,msg)
end
rosshutdown
```

Figura 13: Movimiento realizado mediante un script en MATLAB: 1)Posición de home 2)Posición final 3)Script en MATLAB

- **Cree un script que retorne la configuración de los 4 ángulos de las articulaciones en radianes**

  Partiendo del script mostrado en la figura 13 solo se agregan dos lineas antes del shutdown (figura 14), en donde se pide que se retornen los datos del tópico suscrito después, en la segunda linea se muestran las rotaciones en radianes de las cuatro primeras articulaciones, ya que hay 6 contando los dedos del gripper, pero para el ejercicio estos no son necesarios.

```
clear all;
rosinit
rostopic list;
rostopic info /phantom/my_values/joint_states;
suscriptor=rossubscriber("/joint_states","sensor_msgs/JointState");
publicador=rospublisher('/phantom/my_values/joint_states','sensor_msgs/JointState');
msg= rosmessage('sensor_msgs/JointState');
nombres=["j1","j3","j10","j17"];
rotacion = [pi -pi/4 pi/2 pi/3];
for i=1:4
    msg.Name=nombres(i)
    msg.Position=rotacion(i);
    msg.Velocity=0.5;
    send(publicador,msg)
end
msg2=receive(suscriptor,10);
msg2.Position(1:4)
rosshutdown
```

```
  Use showdetails to show the contents of the message

ans =

    3.1416
   -0.7854
    1.5700
    1.0469

Shutting down global node /matlab_global_node_41497 with NodeURI http://luis:46483/
>>
```

Figura 14: Script que retorna la configuración de los 4 ángulos en radianes

II. MATLAB + ROS + TOOLBOX:

Para esta sección es necesario tener el modelo matemático del robot Phantom en Matlab y el modelo físico en RViz. Se empieza por compilar el modelo URDF y el modelo matemático en Matlab. Se procede a realizar la conexión entre Matlab y ROS por medio de la identificación del publicador y suscriptor. Fue necesario modificar el archivo de launch para hacer esto posible, una vez lograda la conexión se puede establecer el intercambio de información. Las configuraciones fueron almacenadas previamente y se van enviado al suscriptor por medio del publicador de Matlab. Se grafica el modelo en Matlab y se observa el resultado en RViz de los mensajes publicados a este robot. Los resultados del proceso se pueden observar a continuación en los cinco pares de imágenes capturadas, uno para ciclo de ejecución/configuración:

- Configuración 1: (home) C1 = [0 0 0 0]

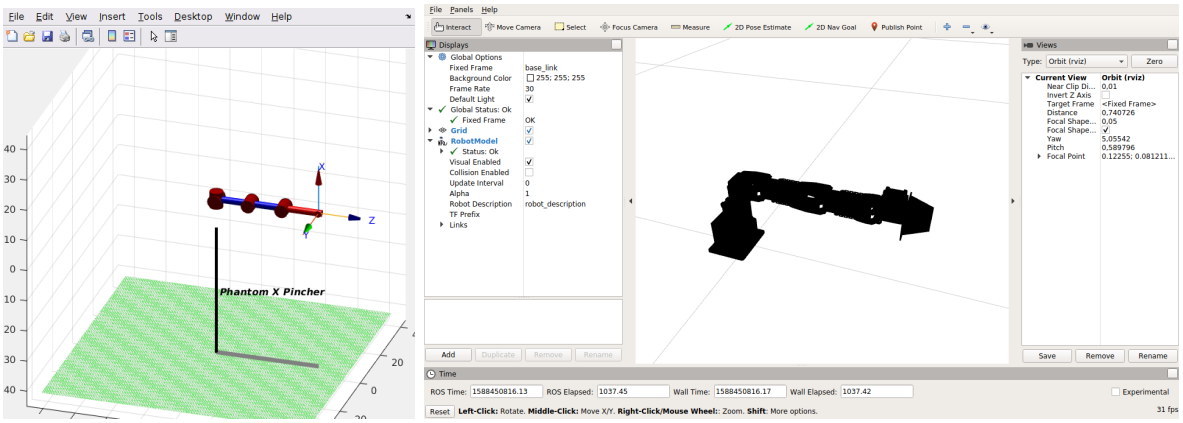


Figura 15: 1) Configuración 1 en Matlab 2) Configuración 1 en RViz

- Configuración 2: C2 = [-20 -20 -20 -20]

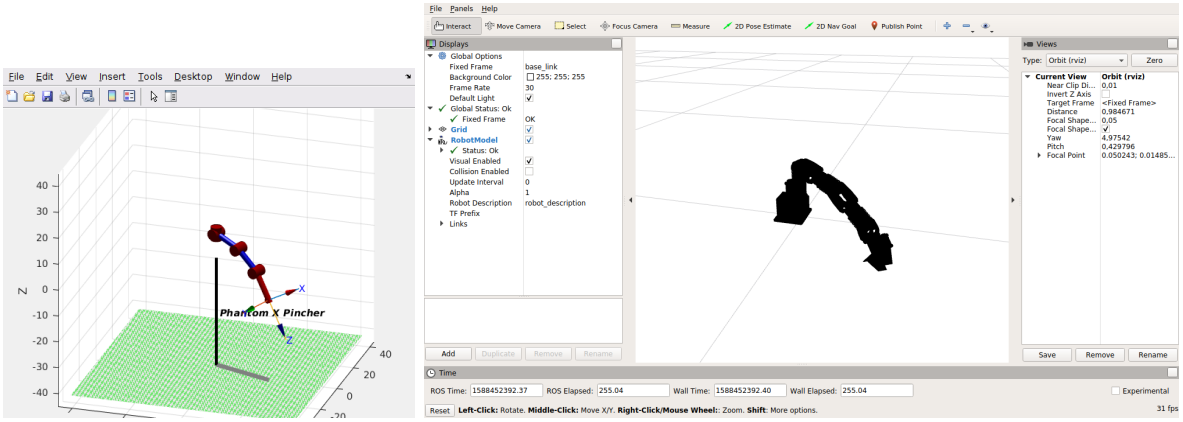


Figura 16: 1) Configuración 2 en Matlab 2) Configuración 2 en RViz

- Configuración 3: C3 = [30 -30 30 -30]

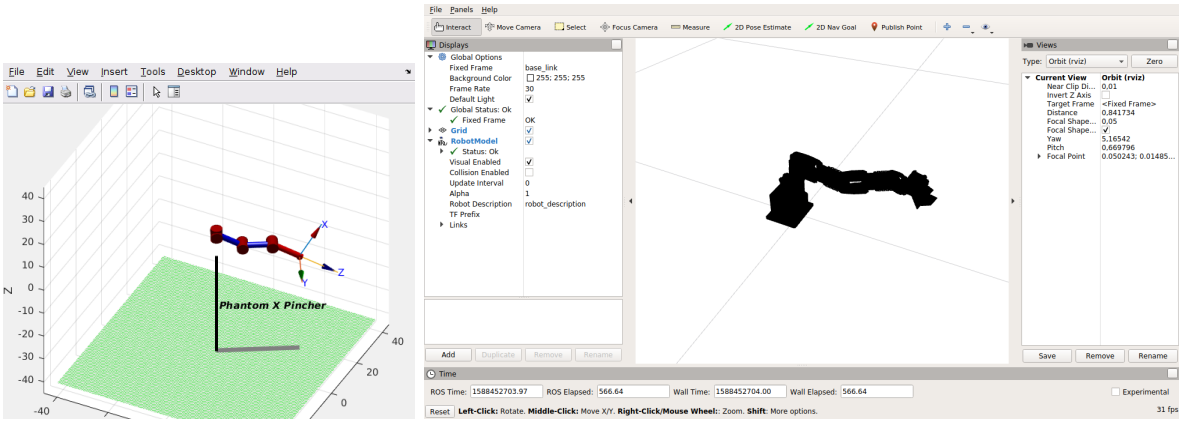


Figura 17: 1) Configuración 3 en Matlab 2) Configuración 3 en RViz

- Configuración 4: C4 = [-90 15 -55 17]

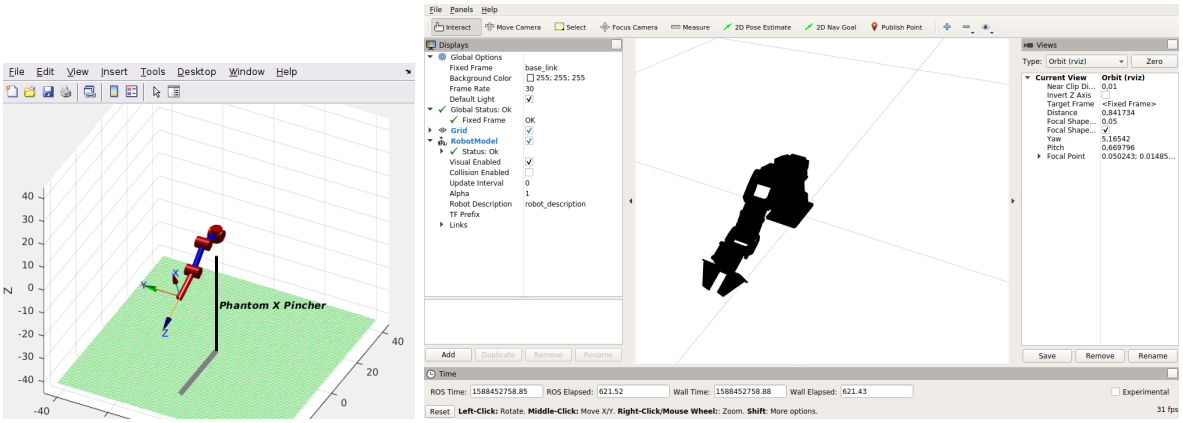


Figura 18: 1) Configuración 4 en Matlab 2) Configuración 4 en RViz

- Configuración 5: C5 = [-90 45 -55 45]

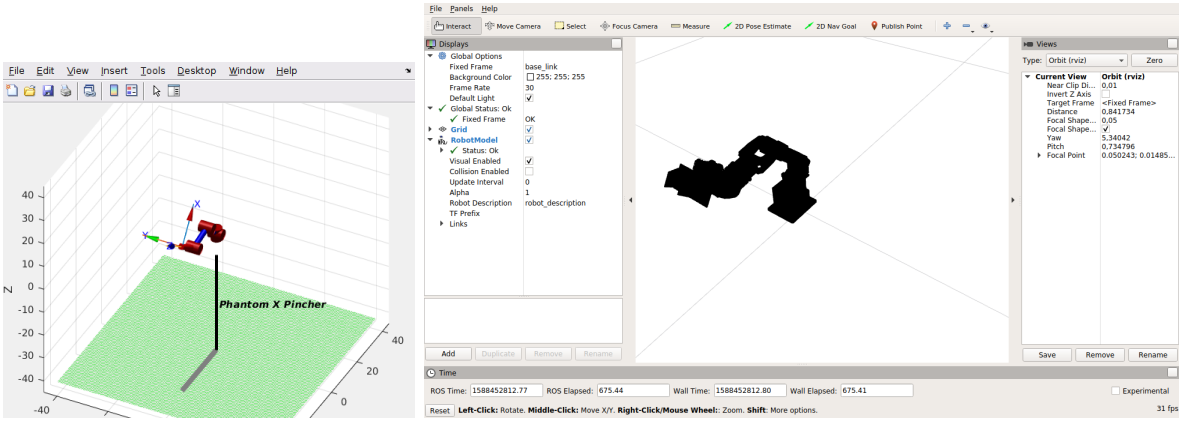


Figura 19: 1) Configuración 5 en Matlab 2) Configuración 5 en RViz

III. LINK VIDEO YOUTUBE

https://youtu.be/mtUvQPt-JGs

REFERENCIAS

[1] Bioloid;*"Bioloid Robot Brackets"*;17/12/2010; tomado de:*https://www.thingiverse.com/thing:5192*; visitado el 01/05/2020.
[2] Hans Toquica;*PhantomX PincherSpecifications*; Enero 2018; tomado de:https://www.researchgate.net/publication/322222351_PhantomX_Pincher_Specificatior visitado el 01/05/2020.
[3] *PhantomX Pincher Robot Arm Assembly Guide*; tomado de: http://www.trossenrobotics.com/productdocs/assemblyguides/phantomx-basic-robot-arm.html; visitado el: 01/05/2020
[4] wiki ROS; *XML specifications*; tomado de: http://wiki.ros.org/urdf/XML, visitado el 01/05/2020.