Control de Brazo Robot

1st Walther Alejandro Bejarano Romero
Ingenieria Electronica
UNIVERSIDAD MANUELA BELTRAN
BOGOTA

2nd Martin Parada Rodriguez
Ingenieria Electronica
UNIVERSIDAD MANUELA BELTRAN
BOGOTA

***Abstract*—In the following document it will be shown how with the help of Matlab and ROS you can control the movement of a robot with respect to the use of a proportional controller by means of a trajectory function, in this case a circular trajectory .**

En el siguiente documento se va mostrar de como con ayuda de Matlab y ROS se puede controlar el movimiento de un robot respecto al uso de un controlador proporcional por medio de una función de trayectoria en este caso una trayectoria circular .

***Index Terms*—xml , grados de libertad, articulaciones**

I. Introduction

En esta sesión importaremos el robot a un modelo URDF del entorno Simscape Multibody de Matlab utilizando la función smimport con el nombre de archivo URDF como argumento principal, al obtener el modelo aplicaremos el uso de algebra lineal como una herramienta para el análisis de sistemas de rotación y traslación, es decir que se calcula la localización del efector final del robot en el espacio cartesiano , en base del calculo de las matriz de transformación homogénea se puede obtener el jacobiano para el desplazamiento del robot , en base de eso obtendremos en control proporcional de la función de trayectoria y con ayuda de ros podremos hacer el movimiento de forma física .

II. Desarollo

A. Software

Durante la practica se utiliza lo que se conoce como URDF o Formato de descripción de robótica unificada, este paquete contiene una serie de especificaciones XML para modelos de robots, sensores, escenas, etc. Cada especificación XML tiene un analizador correspondiente en uno o más idiomas. Se usa principalmente a en el mundo académico y la industria para modelar sistemas multicuerpo, como brazos manipuladores robóticos para la fabricación de líneas de montaje y robots animatrónicos para parques de atracciones.

Para el laboratorio también fue necesario usar en Matlab algo conocido como Simscape Multibody lo cual proporciona un entorno de simulación multicuerpo para sistemas mecánicos 3D, como robots, suspensiones de vehículos, maquinaria de construcción y trenes de aterrizaje de aeronaves. Puede modelar sistemas multicuerpo utilizando bloques que representan cuerpos, articulaciones, restricciones, elementos de fuerza y sensores. Simscape Multibody formula y resuelve las ecuaciones de movimiento de todo el sistema mecánico. [1]

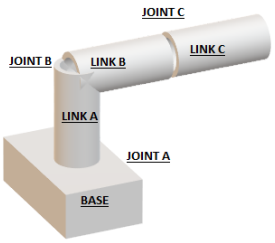


Fig. 1. Simscape Multibody

B. Cinemáticas

La cinemática se encarga de estudiar el movimiento de este con respecto a un eje de referencia. Así la cinemática se encarga de estudiar la descripción analítica del movimiento espacial del robot con una función de tiempo. [3]

En general existen dos problemas dentro de la cinemática de un robot llamados: **problema cinemático directo**, que consiste en determinar cuál es la posición y orientación del extremo final del robot respecto a un sistema coordenado tomado de referencia, y el segundo es **problema cinemático inverso**, que resuelve la configuración que debe tomar el robot para determinar una posición y orientación dentro de su espacio de trabajo. [3]

Un robot manipulador puede ser modelado como una cadena cinemática abierta, la manipulación de la pieza llevada a cabo por el robot implica el movimiento espacial de su extremo y para que el robot pueda manipular una pieza, es necesario conocer su LOCALIZACIÓN, es decir la posición y orientación de ésta con respecto a la base del robot.

Lab_3_s0

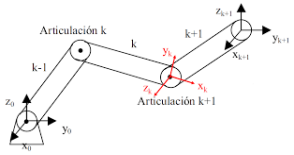


Fig. 2. Cinemática

C. Transformación de Matrices Homogéneas

La representación mediante coordenadas homogéneas de la localización de sólidos e n un espacio dimensional se realiza a través de coordenadas de un espacio (n+1)-dimensional. Es decir, un espacio n-dimensional se encuentra representado en coordenadas homogéneas por (n+1) dimensiones, de tal forma que un vector p(x,y,z) vendrá representado por p(wx,wy,z,w), donde w tiene un valor arbitrario y representan un factor de escala. [5]
A partir de la definición de las coordenadas homogéneas surge inmediatamente el concepto de matriz de transformación homogénea. Se define como matriz de transformación homogénea T a una matriz de dimensión 4x4. [5]
la matriz puede obtenerse de la siguiente manera:

$$Mth = \begin{bmatrix} R_{3x3} & P_{3x1} \\ f_{1x3} & W_{1x1} \end{bmatrix} = \begin{bmatrix} Rotación & Traslación \\ Perspectiva & Escalado \end{bmatrix} \quad (1)$$

Así pues, se puede considerar que una matriz homogénea se haya compuesta por cuatro submatrices de distinto tamaño: una submatriz R3x3 que corresponde a una matriz de rotación; una submatriz P3x1 que corresponde al vector de traslación; una submatriz f1x3 que representa una transformación de perspectiva, y una submatriz w1x1 que representa un escalado global. [5]

Con ayuda de la matriz de transformación homogénea se puede obtener el nuevo punto en caso de que se aplique una rotación en alguno de las articulaciones.

$$Punto \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = Mth. \begin{bmatrix} X' \\ Y' \\ Z' \\ 1 \end{bmatrix} \quad (2)$$

D. *Metodo de Denavit-Hartenberg para la obtención del modelo*

Se trata de un procedimiento sistemático para describir la estructura cinemática de una cadena articulada constituida por articulaciones con un solo grado de libertad. Para ello, a cada articulación se le asigna un Sistema de Referencia Local con origen en un punto Qi y ejes ortonormales Xi,Yi,Zi comenzando con un primer Sistema de Referencia fijo e inmóvil dado por los ejes X0,Y0,Z0 anclado a un punto fijo Q0 de la Base sobre la que está montada toda la estructura de la cadena. Este Sistema de Referencia no tiene por qué ser el Universal con origen en (0,0,0) y la Base canónica.

1) *Algoritmo*:

1) Numerar los eslabones comenzando con 1 (primer eslabón móvil de la cadena) y acabando con n (último eslabón móvil). Se numerara como eslabón 0 a la base fija del robot.

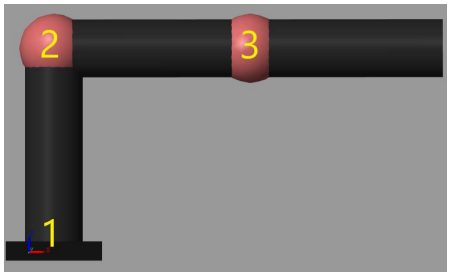


Fig. 3. Numero de articulaciones

2) Numerar cada articulación comenzando por 1 (la correspondiente al primer grado de libertad y acabando en n).
3) Localizar el eje de cada articulación. Si esta es rotativa, el eje será su propio eje de giro. Si es prismática, será el eje a lo largo del cual se produce el desplazamiento.

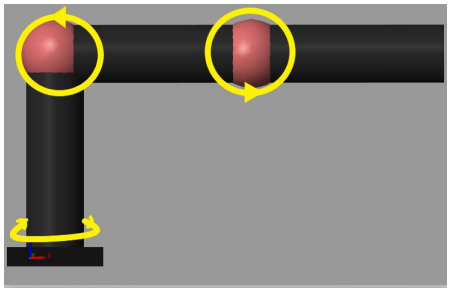


Fig. 4. eje Z

4) Para i de 0 a n-1, situar el eje Zi, sobre el eje de la articulación i+1.

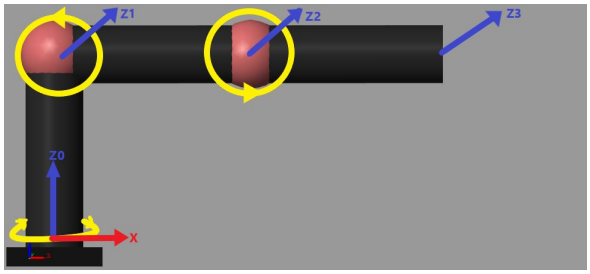


Fig. 5. eje Z

5) Situar el origen del sistema de la base (S0) en cualquier punto del eje Z0. Los ejes X0 e Y0 se situaran dé modo que formen un sistema dextrógiro con Z0.
6) Para i de 1 a n-1, situar el sistema (Si) (solidario al eslabón i) en la intersección del eje Zi con la línea normal común a Zi-1 y Zi. Si ambos ejes se cortasen se situaría (Si) en el punto de corte. Si fuesen paralelos (Si) se situaría en la articulación i+1.
7) Situar Xi en la línea normal común a Zi-1 y Zi.

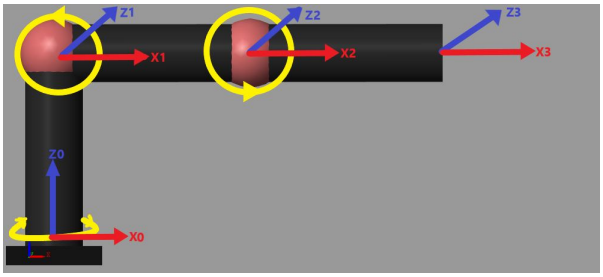


Fig. 6. eje X

8) Situar Yi de modo que forme un sistema dextrógiro con Xi y Zi.
9) Situar el sistema (Sn) en el extremo del robot de modo que Zn coincida con la dirección Zn-1 y Xn sea normal a Zn-1 y Zn.

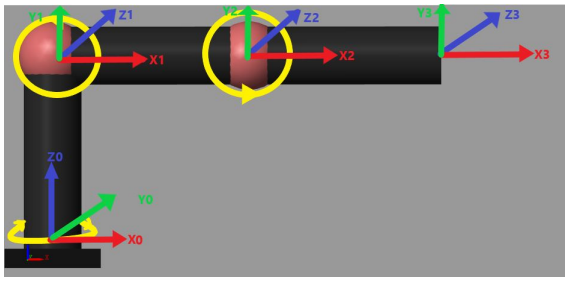


Fig. 7. eje Y

10) Obtener ai como el ángulo que habría que girar entorno a Xi (que ahora coincidiría con Xi-1), para que el nuevo (Si-1) coincidiese totalmente con (Si).
11) Obtener las matrices de transformación i-1Ai.
12) Obtener la matriz de transformación que relaciona el sistema de la base con el del extremo del robot T = 0Ai, 1A2... n-1An.
13) La matriz T define la orientación (submatriz de rotación) y posición (submatriz de traslación) del extremo referido a la base en función de las n coordenadas articulación.

q	Θz	dz	dx	αx
1	Θ1	L1	0	$-\pi/2$
2	Θ2	0	L2	0
3	Θ3	0	L3	0

Fig. 8. Tabla

Con los valores de esta tabla podemos obtener la matriz de transformación homogénea por la ecuación,cabe decir que toca hacer 3 de estas matrices, debido a que existen 3 articulaciones.

$$Mth = R(\theta_z).T(z).T(x).R(X) \tag{3}$$

Como ejemplo de la primera fila de la tabla , es decir la primera articulación. (La multiplicación de las diferentes matrices tiene que hacerse en el mismo orden con respecto a la ecuación anterior , debido a que la multiplicación en matrices no es conmutativa).
L1:altura del cilindro

$$R(\theta_z) = \begin{bmatrix} cos(\theta_z) & -sen(\theta_z) & 0 & 0 \\ sen(\theta_z) & cos(\theta_z) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4}$$

$$T(z) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & L1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5}$$

$$T(X) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6}$$

$$R(x) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos(\frac{-\pi}{2}) & sen(\frac{-\pi}{2}) & 0 \\ 0 & sen(\frac{-\pi}{2}) & cos(\frac{-\pi}{2}) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7}$$

Al realizar la multiplicación de estas matrices nos dará como resultado la matriz de transformación homogénea.

E. Jacobiano

El jacobiano o matriz jacobiana constituye una de las más importantes herramientas para la caracterización de un manipulador, ya que es bastante útil para problemas como [8]:

- Encontrar singularidades
- Análisis de redundancias
- Determinación del algoritmo de cinemática inversa
- Descripción del mapeo entre las fuerzas aplicadas al efector final y los torques resultantes en las articulaciones. [8]

La matriz Jacobiana es una matriz formada por las derivadas parciales de primer orden de una función. [8]

$$f(x_1, x_2, x_3 ... x_n) = (f_1, f_2 ... f_m) \tag{8}$$

$$J = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_1} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial f_m}{\partial x_1} & \frac{\partial f_m}{\partial x_2} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix} \tag{9}$$

Por tanto, las matrices Jacobianas siempre tendrán tantas filas como funciones escalares $(f_1, f_2, \ldots, f_m)$ tenga la función, y el número de columnas coincidirá con el número de variables $(x_1, x_2, \ldots, x_n)$.

Por otro lado, a esta matriz también se la conoce como diferencial jacobiana o aplicación lineal jacobiana. De hecho,

a veces también se escribe con la letra D en vez de la letra J.

Dependiendo de qué matriz, de las planteadas anteriormente, sea singular, un mecanismo de lazo cerrado podría tener una configuración singular de cinemática directa, una de cinemática inversa o ambas. Así, la planeación de trayectorias se desarrollará por medio del algoritmo de cinemática inversa, el cual puede ser usado para obtener un modelo matemático que represente un movimiento diferencial, el cual se ve representado en [9].

F. Software ROS

ROS (Robot Operating System) es un kit de desarrollo de software de código abierto para aplicaciones de robótica. ROS ofrece una plataforma de software estándar para desarrolladores de todas las industrias que los llevará desde la investigación y la creación de prototipos hasta la implementación y la producción. [6]

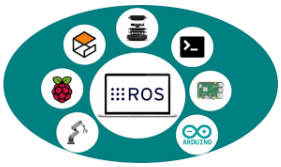


Fig. 9. ROS

ROS proporciona las herramientas, bibliotecas y capacidades que necesita para desarrollar sus aplicaciones de robótica, lo que le permite dedicar más tiempo al trabajo que es importante para su negocio. Debido a que es de código abierto, tiene la flexibilidad de decidir dónde y cómo usar ROS, así como la libertad de personalizarlo según sus necesidades. Además, ROS no es exclusivo, no necesita elegir entre ROS o alguna otra pila de software; ROS se integra fácilmente con su software existente para llevar sus herramientas a su problema. [6]

ROS es y siempre será de código abierto, lo que garantiza que nuestra comunidad global tenga acceso gratuito y sin restricciones a un SDK de robótica de alta calidad, el mejor en su clase y con todas las funciones. Construimos ROS sobre otros proyectos de código abierto y aprovechamos y seguimos estándares abiertos (como el DDS de OMG) siempre que sea posible. [6]

G. RASPBERRY PI

Raspberry Pi es el nombre de una serie de computadoras de placa única fabricadas por la Fundación Raspberry Pi , una organización benéfica del Reino Unido que tiene como objetivo educar a las personas en informática y facilitar el acceso a la educación informática.

El Raspberry Pi se lanzó en 2012 y desde entonces se han lanzado varias iteraciones y variaciones. El Pi original tenía una CPU de 700 MHz de un solo núcleo y solo 256 MB de RAM, y el último modelo tiene una CPU de cuatro núcleos con más de 1,5 GHz y 4 GB de RAM. El precio de Raspberry Pi siempre ha sido inferior a $100 (generalmente alrededor de $35 USD), más notablemente el Pi Zero, que cuesta solo $5. [7]

Fig. 10. Raspberry Pi

El sistema operativo de todos los productos Raspberry Pi es Linux. Linux es un sistema operativo de código abierto que interactúa entre el hardware y los programas de software de la computadora. El lenguaje utilizado con Raspberry Pi es Python, un lenguaje de programación de alto nivel y propósito general que se utiliza para desarrollar aplicaciones, sitios web y aplicaciones web de interfaz gráfica de usuario (GUI). Uno de los beneficios de Raspberry Pi es que no es necesario tener un conocimiento profundo de Linux o Python antes de comenzar un proyecto con Raspberry Pi. De hecho, el propósito del producto es enseñar el sistema y el idioma a través de proyectos atractivos. [7]

III. Desarrollo de la Practica

A. Control proporcional

Es la acción de control consiste en la multiplicación entre la señal de error actuante y la sensibilidad proporcional o ganancia como para que hagan que el error en estado estacionario sea casi nulo. Control proporcional Es una de las acciones de control empleadas en los lazos de regulación automática. En este caso el control proporcional para robot se ve de la forma :

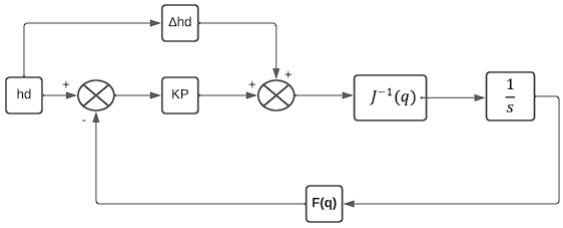


Fig. 11. control proporcional

para dar una mejor compresión de los diferentes bloques existentes en la figura anterior:

- hd: función de trayectoria (asigna los puntos del movimiento , es decir que coloca las coordenadas para que complete un movimiento)
- KP: constante de proporcionalidad
- hd : derivada de la función de trayectoria

- $J^{-1}(q)$: Jacobiano inverso para el cambio de los ángulos de las 4 diferentes articulaciones .
- $\frac{1}{s}$: integrador
- F(q): función para encontrar los puntos por medio de los angulos , es decir una cinematica directa por medio de DENAVIT-HARTENBERG .

B. Codigo de función de trayectoria

Para este caso era necesario calcular los diferentes puntos de la trayectoria ya que si se toman puntos que el brazo robot no llega a alcanzar se colocan diversos errores durante el transcurso de la simulación en simulink , por lo que la trayectoria quedo de la siguiente forma :

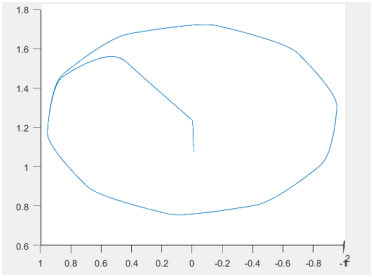


Fig. 12. trayectoria

Para ver el codigo , se colocara como anexo.

C. Codigo Con DENAVIT-HARTENBERG Para cinematica directa

Con ayuda e Matlab se pudo obtener los nuevos puntos, siguiendo el algoritmo de DENAVIT-HARTENBERG, el cual fue necesario aplicarlo 3 veces debido al numero de articulaciones ya anterior mostradas en las imágenes.

El código lo encontraran en el anexo 1 , para la mejor compresión.

D. Jacobiano Inverso

Con ayuda de esto podemos obtener el movimiento ya que constituye una de las más importantes herramientas para la caracterización de un manipulador, entonces obtendremos los valores de los ángulos de cada articulación con ayuda del integrador , todo el código del jacobiano se anexara debido a que estos resulta demasiado extenso y bastante complicado.

Da como terminado el control total a nivel de programación:

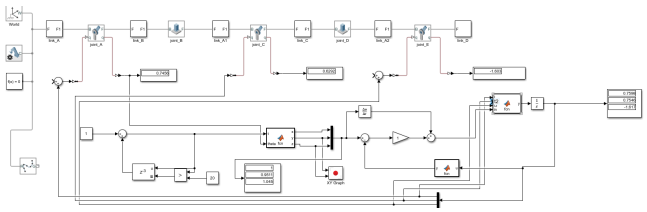


Fig. 13. simulink control

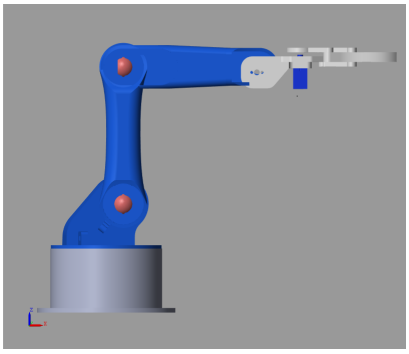


Fig. 14. simulacion

E. composicion de ROS

Teniendo en cuenta todos los datos anterior se da por comienzo el desarrollo del proyecto, el cual en este proyecto utilizaremos como controlador físico y computacional una Raspberry pi 4 en su versión de 4 Gb de ram, a esta placa le instalaremos el sistema operativo Ubuntu/Linux en su versión 20.04 LTS de 64 bits, por medio de este nos conectaremos por ssh desde un computador local ya que este sistema operativo no tiene interfaz ya siendo todo por consola, como primer paso tenderemos que configurar el wifi de la raspberry por medio de código, para eso utilizamos la siguiente línea desde el ssh:

```
sudoedit /etc/netplan/50-cloud-init.yaml
```

Aquí abrirá un documento que nos muestra el SSID y la contraseña para la conexión aquí pondremos la de nuestra red.

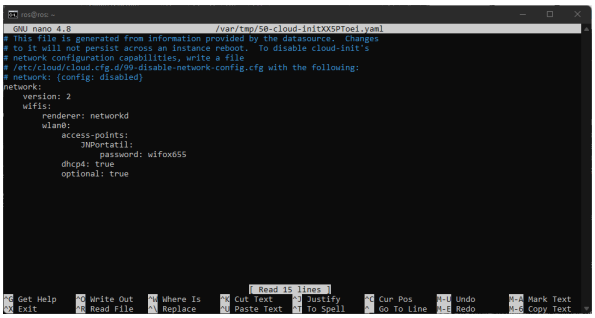


Fig. 15. abrir documento

se da ctrl+x para salir, y para guardar y enter, una vez en la consola utilizaremos dos lineas de codigo para aplicar y actualizar el netplan:

```
    sudo netplan apply
sudo netplan --debug apply
```

Para el aplicativo ROS Noetic, lo instalaremos utilizando las siguientes líneas de código:

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

```
sudo apt install curl

curl -s https://raw.githubusercontent.
com/ros/
rosdistro/master/ros.asc | sudo
apt-key add -

sudo apt update

sudo apt install ros-noetic-desktop

source /opt/ros/noetic/setup.bash

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

source ~/.bashrc

sudo apt install python3-rosdep
python3-rosinstall
python3-rosinstall-generator
python3-wstool build-essential

sudo apt install python3-rosdep

sudo rosdep init

rosdep update
```

Consiguiente a esto, necesitamos instalar la librería de pip para consiguiente a esto instalara la librería rpi.gpio, estas librerías nos permitirá hacer utilización de cada uno de los pines de la Raaspberry Pi 4 por consiguiente ingresamos las siguientes lineas de codigo:

```
sudo apt-get install python3-pip

sudo apt-get install python3-
dev python3-rpi.gpio
```

También tendremos que instalar unos paquetes de Ros llamados catkin los cuales son del sistema de compilación oficial de ROS y el sucesor del sistema de compilación original de ROS, rosbuild. Esta combina macros de CMake y secuencias de comandos de Python para proporcionar alguna funcionalidad además del flujo de trabajo normal de CMake. Para la instalación utilizaremos las siguientes lineas de codigo:

```
    sudo apt-get install ros-noetic-catkin

sudo apt-get install cmake
python3-catkin-pkg python3-empy
python3-nose python3-setuptools
libgtest-dev build-essential
```

Por medio de catkin necesitaremos crear el espacio de trabajo, el paquete de ROS y la configuración de del espacio de trabajo para ello las siguientes lineas de codigo:

```
// Crear el workspace de catkin
source /opt/ros/noetic/setup.bash

mkdir -p ~/catkin_ws/src

cd ~/catkin_ws/

catkin_make

catkin_make -DPYTHON_EXECUTABLE=
/usr/bin/python3

source devel/setup.bash

echo $ROS_PACKAGE_PATH

// Crear el paquete de ROS
////Creando paquete catkin
cd ~/catkin_ws/src

catkin_create_pkg brazorobot
std_msgs rospy roscpp

////COnstruyendo el espacio de trabajo
catkin

cd ~/catkin_ws

catkin_make

. ~/catkin_ws/devel/setup.bash

rospack depends1 brazorobot

roscd brazorobot

cat package.xml

///Dependencias indirectas

rospack depends1 rospy

/// ubicar espacio de trabajo

~/catkin_ws

source devel/setup.bash

rospack find brazorobot

rospack depends brazorobot
```

Para ROS tendremos que configurar la ip tanto de la Raspberry Pi como del pc que nos genera el modem al cual nos estamos conectando, tener en cuenta que todo esto es en red de área local pr eso utilizamos:

```
nano ~/.bashrc
```

y dentro del archivo texto que nos abrió, al final de este escribiremos estasa lineas con la ip generada tanto para el pc como para la rsapberry (192.168.1.7 raspberry - 192.168.1.3 pc):

```
export ROS_MASTER_URI=
http:/192.168.1.3:11311
export ROS_HOSTNAME=
192.168.1.3

export ROS_MASTER_URI=
http:/192.168.1.3:11311
export ROS_HOSTNAME=192.168.1.7

cd catkin_ws/src/brazorobot

mkdir scripts

cd scripts/

sudo nano subscriptor.py
```

Por último para la instalación y configuración implementaremos el codigo del suscriptor con los siguientes comandos:

```
cd catkin_ws/src/brazorobot

mkdir scripts

cd scripts/

sudo nano subscriptor.py
```

Aquí nos abrirá un editor de texto vacio en el cual como vemos utilizamos el lenguaje de programación Python el cual utilizaremos en su versión 3, por medio de este lenguaje realizamos un código el cual será el influyente en el trámite de datos y la conexión con Matlab/Simulink, este código nos permite capturar los datos que son enviados de Matlab/Simulink para el movimiento físico del Brazo, la primera parte del código es el siguiente:

```python
#!/usr/bin/env python3

import rospy
import RPi.GPIO as GPIO
import time

from this import d
from geometry_msgs.msg import Twist

servo1 = 18 #a
servo2 = 12 #b
servo3 = 13 #c
#servo4 = 19 #d

theta1 = 0
theta2 = 0
theta3 = 0
theta4 = 0

GPIO.setmode(GPIO.BCM)
GPIO.setup(servo1, GPIO.OUT)
GPIO.setup(servo2, GPIO.OUT)
GPIO.setup(servo3, GPIO.OUT)
#GPIO.setup(servo4, GPIO.OUT)

a = GPIO.PWM(servo1, 50)  #GPIO 18 con un PWM de 50Hz
b = GPIO.PWM(servo2, 50)  #GPIO 12 con un PWM de 50Hz
c = GPIO.PWM(servo3, 50)  #GPIO 13 con un PWM de 50Hz
#d = GPIO.PWM(servo4, 50)  #GPIO 19 con un PWM de 50Hz

a.start(12.5) #Inicializacion
b.start(0)
c.start(0)
#d.start(0)
```

Fig. 16. codigo python

Importa primero rospy, rospy es una biblioteca de Python para ROS. La API del cliente rospy permite a los programadores de Python interactuar rápidamente con los temas, servicios y parámetros de ROS. El diseño de rospy favorece la velocidad de implementación (es decir, el tiempo del desarrollador) sobre el rendimiento del tiempo de ejecución para que los algoritmos puedan crear prototipos y probarse rápidamente dentro de ROS.
Siguiente a este importa RPi.GPIO, este paquete proporciona un módulo de Python para controlar el GPIO en una Raspberry Pi o puertos de una Raspberry Pi.
El módulo time de la biblioteca estándar de Python proporciona un conjunto de funciones para trabajar con fechas y/o horas.
$Geometry_msgs/Twist$ es un módulo que viene integrado con el aplicativo ROS, este modulo permite permite comandos de movimiento muy simples, esto quiere decir, que permite el movimiento en sus ejes y en direcciones indicadas, directamente podemos especificar movimientos lineales o angulares, una vez que establezca los parámetros requeridos en las variables en $geometric_msgs/Twist$ y lo publique, el robot en este caso el brazo se moverá en consecuencia.

Servo1 – 2 – 3 – 4 estas variables son definidas para los pines de salida de la raspberry pi, en este caso definimos los pines 18, 12, 13 y 19 los cuales permiten salida de PWM para los servomotores que generan el movimiento del brazo. Theta1 – 2 – 3 – 4 estas son variables las cual definimos para el ingreso del valor que será enviado desde Matlab/Simulink, estas serán actualizadas constantemente para el movimiento del brazo.

GPIO.setmode(GPIO.BCM) y GPIO.setup(servo1 – 2 – 3 – 4 , GPIO.OUT) estos dos parámetros son invocados de la librería RPi.GPIO, le primero GPIO.setmode(GPIO.BCM) el cual nos sirve para definir los puertos usando la numeración del chip BROADCOM directamente y en el siguiente parámetro GPIO.setup(servo1 – 2 – 3 – 4 , GPIO.OUT) definimos los pines que colocamos al inicio del código con la variable servo1… , y por medio de GPIO.OUT especificamos de que estos son unos puertos de salida.
a = GPIO.PWM(servo1, 50), esta línea de código especificaremos en una variable la cual nos referimos que a ese puerto se realizara un pwm el cual será de 50Hz. a.start(12.5) para esta primera parte del código utilizamos esta línea de código para darle un valor inicial al robot, en esta línea podemos definir el punto "home" del brazo, este se maneja con valores entre 2.5 que caracterizando el robot seria nuestro 0°, 7.5 que serían 90° y 12.5 que serían 180°.

```
36  def callback(data):
37
38      theta1 = data.linear.x
39      a.ChangeDutyCycle(theta1)
40      print('Theta1 =', theta1)
41
42      theta2 = data.linear.y
43      b.ChangeDutyCycle(theta2)
44      print('Theta2 =', theta2)
45
46      theta3 = data.linear.z
47      c.ChangeDutyCycle(theta3)
48      print('Theta3 =', theta3)
49
50      #theta4 = data.linear.x
51      #d.ChangeDutyCycle(theta4)
52      #print('Theta4 =', theta4)
53
54  def listener():
55
56      rospy.init_node('suscriptor',anonymous=True)
57      rospy.Subscriber("datarobot", String, callback)
58      rospy.spin()
59
60  if __name__ == '__main__':
61      listener()
```

Fig. 17. python

En esta que es la segunda parte del código ya tenemos la captura de datos para los valores emitidos desde Matlab/Simulink, el envío de los datos a los servos, la creación del nombre del suscriptor o listener y la creación del topic de donde se ejecutan directamente el código.
Definimos un callback con una variable data, esto nos permite estar llamando y actualizando los datos que se emiten por todo el servidor de ros y enviados de Matlab/Simulink.
Invocamos la variable theta1 y por medio de date.linear.x que son variables definidas en simulink con el aplicativo de ros para el modulo $Geometry_msgs/Twist$ las cuales invocan los valores que envía la cinemática al servidor para el movimiento de los servos, la cual se da con la siguiente línea del código a.ChangeDutyCycle(theta1) aquí estamos ingresando ese dato capturado que como dijimos anteriormente en la línea de código de inicialización utilizamos en el rango de 2.5 a 12.5 equivalente de 0° a 180° siendo los valores de ciclo de trabajo, esto lo hacemos para cada uno de los servos y valores de captura y salida a los servos.
Definimos los parámetros para a el listener o suscriptor, estos parámetros son dos, el nombre del suscriptor y el topic de donde será invocado todo el código para la lectura de Matlab/Simulink. Ya terminado este código lo guardamos y cerramos al igual que l configuración del WI-FI de nuestra Raspberry Pi, lo siguiente a esto es darle permisos de administrador para esto utilizamos la siguiente línea de codigo: sudo chmod +x subscriptor.py Lo siguiente es ejecutar el servidor de ROS para poder hacer la conexión conMatlab/Simulink para ello utilizaremos el comando: roscore

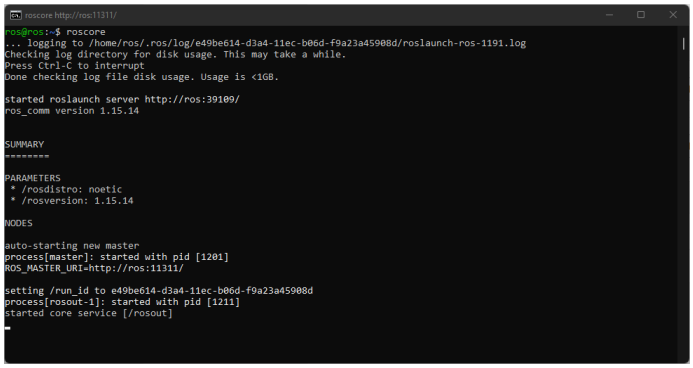
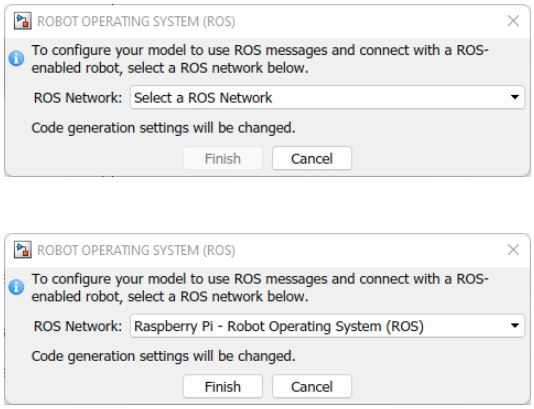


Fig. 18. compilacion

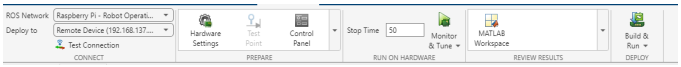
Una vez realizado todo en nuestra Raspberry Pi, nos dirigimos a Matlab/Simulink a realizar la conexión con el servidor de ROS, buscamos el aplicativo de ROS en symulink:

Fig. 19. simulink control

Seleccionamos el aplicativo y escogemos la red de ROS autilizar:



En este caso seleccionamos "Rspberry Pi – Robot Operaatin System (ROS)" para la conexión de este, se nos abrira una pestaña del aplicativo donde nos podremos conectar al servidor:



En la opción deploy to seleccionaremos la opción Manage Remote Device: se nos abrirá una pestaña adicional que nos pedirá el Device address o ip, username y contrseña definida:

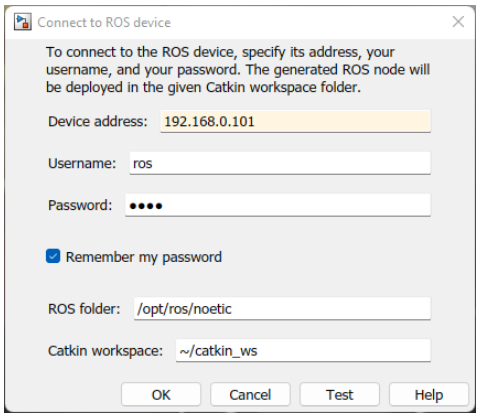


Fig. 20. prueba de conexion

Realizamos un test para verificar la red y la conexión con la ip que nos será mostrada en la consola de symulink:

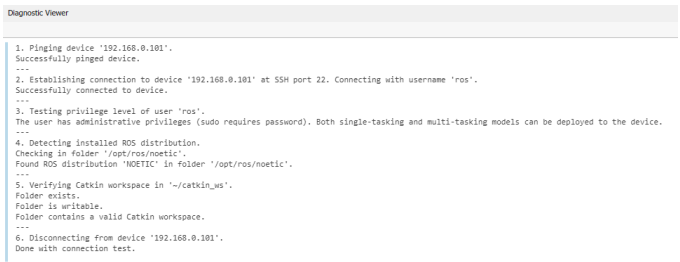


Fig. 21. verificacion de red

Una vez verificada esta conexión agregaremos un bloque de parámetros de ROS que será el "Publicdor" en este bloque también configuraremos conexión para la ubicación de los nodos:

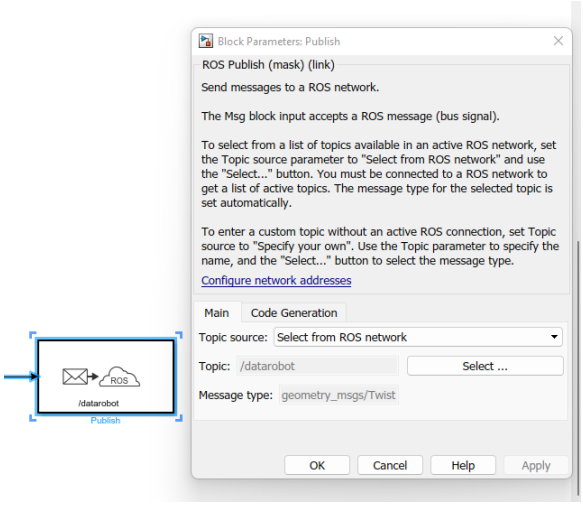
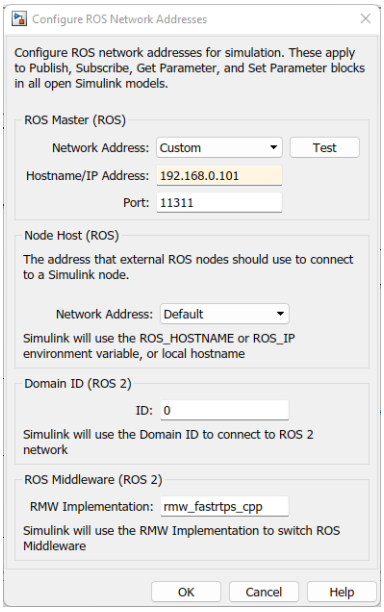
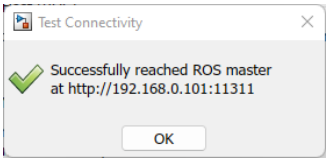


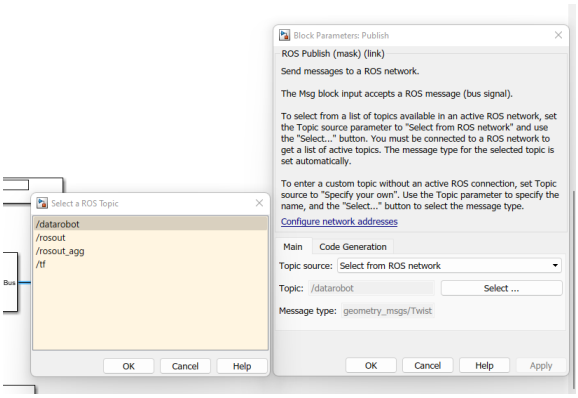
Fig. 22. seleccion de ROS

Configuraremos nuevamente los parámetros de la ip y el puerto, nuevamente haremos un test de conexión que se encuentra en esa pestaña:





Gracias a estas conexiones podremos ver los topics que se encuentran en nuestra Raspberry Pi y aquellos que creamos anteriormente con catkin:



En este caso y anteriormente en el código de Python habíamos creado el topic llamado /datarobot el cual también se había configurado en el espacio de trabajo de catkin, a este topic ejecutaremos toda la conexión y parámetros.

CONCLUSIÓN

En conclusión, este proyecto ha sido completado de forma satisfactoria, cumpliendo los objetivos planteados y conseguir unos resultados satisfactorios. Durante la práctica la creación y desarrollos de un sistema de control y comunicación con una Raspberry para la simulación e implementación de un trayectoria circular.

REFERENCES

[1] Simscape Multibody. https://la.mathworks.com/products/simscape-multibody.html
[2] SDFormat, http://sdformat.org/spec?ver=1.9elem=link

[3] Fundamentos de Robótica, Barrientos Antonio, Aracil Rafael, Universidad Politécnica de Madrid, McGraw-Hill, Año 1997, 1ra. Edición en español, pag. 93. http://proton.ucting.udg.mx/materias/robotica/r166/r81/r81.htm
[4] SDFormat, http://sdformat.org/spec?ver=1.9elem=link

[5] Realidad Virtual: aplicasiones practicas en los negocios y la industria, Chorafas, Dimitris N., Prentice-Hall, México 1996.
[6] ROS, ¿Por qué ROS? https://www.ros.org/blog/why-ros/
[7] Raspberry Pi Foundation, What is a Raspberry Pi? https://www.raspberrypi.org/help/what-
[8] B. Siciliano, L. Sciavicco, L. Villani and G. Oriolo, Robotics: Modeling, Planning and Control. London, United Kingdom: Springer, 2009. DOI: http://dx.doi.org/10.1007/978-1-84628-642-1.
[9] L. Urrea and S. Medina, “Diseño e implementación de una plataforma robótica tipo Delta”. Trabajo de Grado, Ingeniería en Mecatrónica, Facultad de Ingeniería, Universidad Militar Nueva Granada, Bogotá D.C., Colombia, Sep. 2012.