Práctica 1

Antonio Jesús Heredia Castillo

5 de abril de 2020

Ejercicio 1

Represente las posiciones de los puntos respecto de la trama $\{A\}$ cuando:

1. La trama $\{B\}$ se gira un ángulo $\alpha = 90^o$ alrededor del eje X_A
2. La trama $\{B\}$ se gira un ángulo $\alpha = 90^o$ alrededor del eje Y_A
3. La trama $\{B\}$ se gira un ángulo $\alpha = 90^o$ alrededor del eje Z_A

Al comienzo las tramas A y B están superpuestas.En las siguientes graficas podremos ver la trama A en color azul y la trama B girada de color naranja. El resultado de girar $\{B\}$ un ángulo $\alpha = 90^o$ alrededor del eje X_A lo podemos ver en la Figura 1.

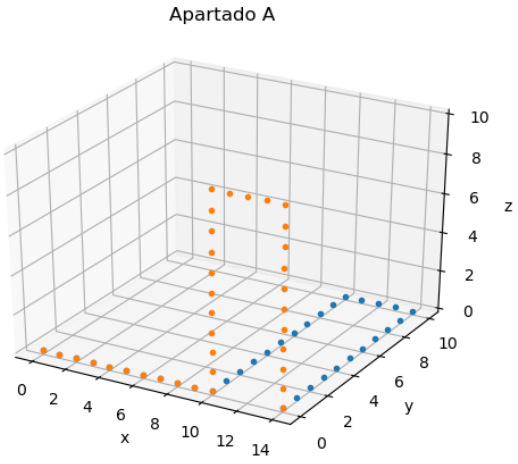

Figura 1: Rotación de 90^o en X_A

Para realizar la rotación anterior he usado la siguiente matriz de rotación:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & cos(90) & -sin(90) \\ 0 & sin(90) & cos(90) \end{pmatrix}$$

El resultado de girar $\{B\}$ un ángulo $\alpha = 90^o$ alrededor del eje Y_A lo podemos ver en la Figura 2.

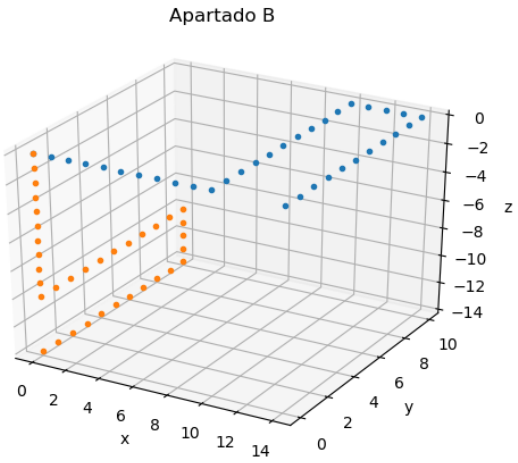

Figura 2: Rotación de 90º en Y_A

Para realizar la rotación anterior he usado la siguiente matriz de rotación:

$$\begin{pmatrix} cos(90) & 0 & sin(90) \\ 0 & 1 & 0 \\ -sin(90) & 0 & cos(90) \end{pmatrix}$$

El resultado de girar $\{B\}$ un ángulo $\alpha = 90^o$ alrededor del eje Z_A lo podemos ver en la Figura 3.

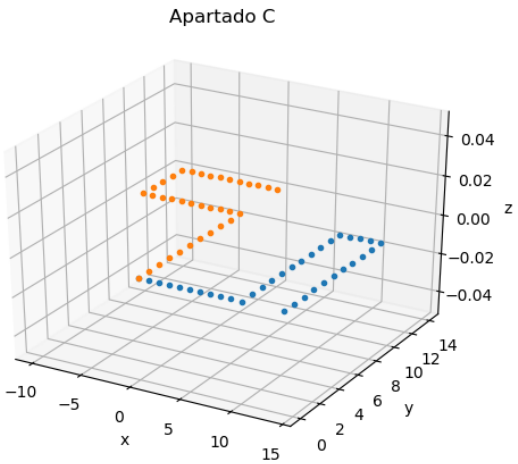

Figura 3: Rotación de 90º en Z_A

Para realizar la rotación anterior he usado la siguiente matriz de rotación:

$$\begin{pmatrix} cos(90) & -sin(90) & 0 \\ sin(90) & cos(90) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Usando simplemente las matriz de rotación podemos conseguir la rotación de todos los puntos respecto a un eje. El único punto que no cambia es el situado en el $\{0, 0, 0\}$ que sea la que sea la matriz de rotación que se le aplique no cambia su posición.

Ejercicio 2

Represente las posiciones de los puntos respecto de la trama $\{A\}$ cuando la trama $\{B\}$ del ejercicio anterior se rota desde la posición original (superpuesta a la trama $\{A\}$) en torno al eje X_B un ángulo $\gamma = 60^o$, a continuación se gira en torno al eje Y_B un ángulo $\beta = 90^o$, y después se gira en torno al eje Z_B un ángulo $\alpha = 30^o$.

Lo primero que voy a hacer es indicar las matrices que voy a usar en este ejercicio aunque son las mismas que en el ejercicio anterior pero cambiando los ángulos. La rotación en X sera la siguiente:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & cos(60) & -sin(60) \\ 0 & sin(60) & cos(60) \end{pmatrix}$$

La rotación en Y sera:

$$\begin{pmatrix} cos(90) & 0 & sin(90) \\ 0 & 1 & 0 \\ -sin(90) & 0 & cos(90) \end{pmatrix}$$

Por ultimo la rotación en Z sera:

$$\begin{pmatrix} cos(30) & -sin(30) & 0 \\ sin(30) & cos(30) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Vamos a ver que ocurre si realizamos las rotaciones en el orden que nos dice el enunciado.
Primero realizamos la rotación de 60^o en el eje X_B como se puede ver en la Figura 4.

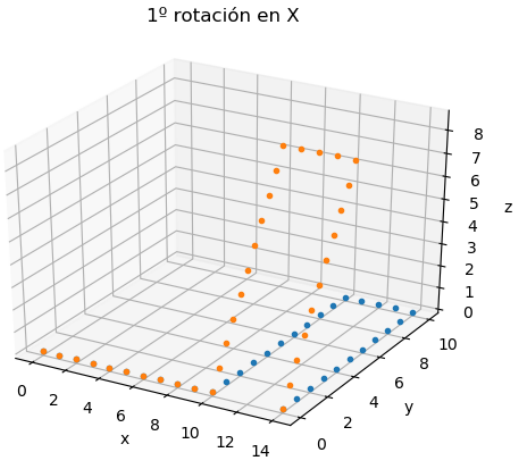

Figura 4: Rotación de 60º en X_B

A continuación realizamos la rotación de 90º en el eje Y_B y obtendremos lo que vemos en la Figura 5.

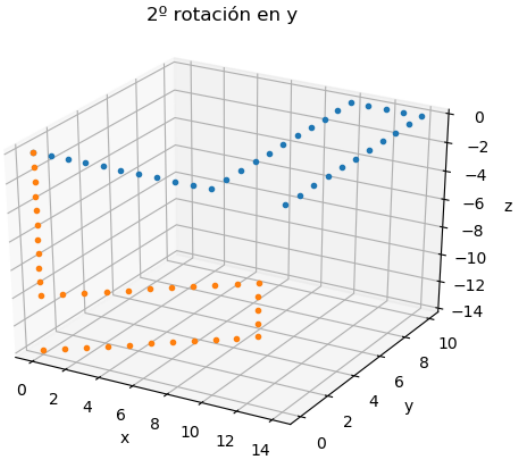

Figura 5: Rotación de 90º en Y_B

Por ultimo realizamos la rotación de 30º en el eje Z_B quedando la Figura 4.

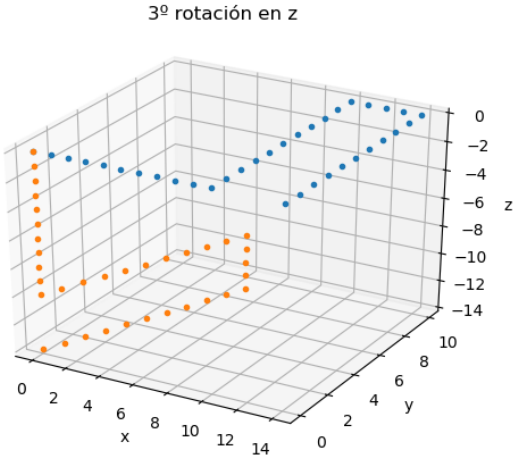

Figura 6: Rotación de 30º en Z_B

Estas rotaciones se pueden ver mejor cuando obtenemos la grafica desde python, ya que podemos mover los ejes y cambiar la vista para tener una mejor perspectiva de la posición. Ya que de la Figura 5 a la Figura 6 parece a primera vista que no cambia mucho, pero si que se nota el cambio.

Ahora vamos a ver que ocurre si realizamos las mismas rotaciones con los mismo grados pero en diferente orden. Primero realizamos la rotación de 60º en el eje X_B como se puede ver en la Figura 7.

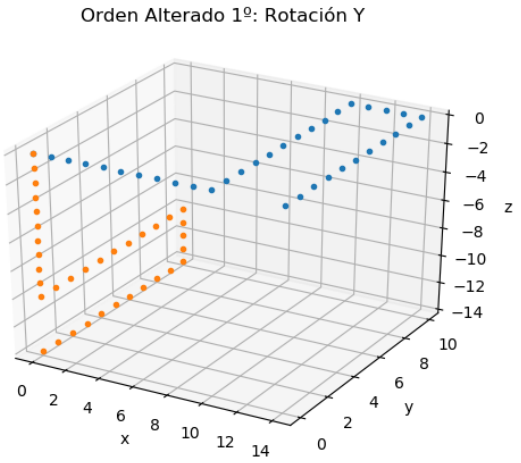

Figura 7: Rotación de 90º en Y_B

A continuación realizamos la rotación de 30º en el eje Z_B y obtendremos lo que vemos en la Figura 8.

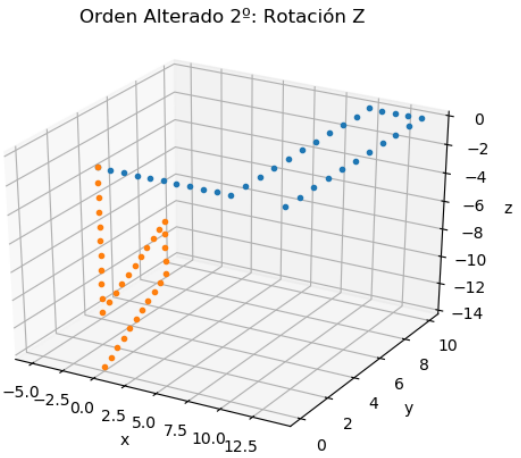

Figura 8: Rotación de 90º en Z_B

Por ultimo realizamos la rotación de 60º en el eje X_B quedando la Figura 9.

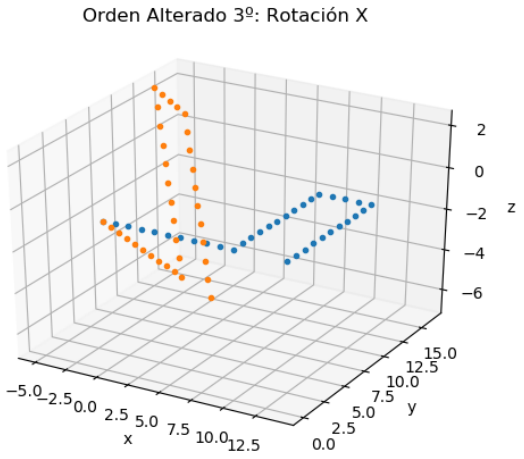

Figura 9: Rotación de 30º en X_B

Comparando la Figura 6 (realizando en el orden adecuado las rotaciones) y la Figura 9 (en un orden distinto al dado), podemos ver que no obtenemos la misma posición final a pesar de haber ejecutado las mismas rotaciones (se puede observar en el código que son las mismas matrices de rotación).

Ejercicio 3

Diseñe e implemente una función en Python que reciba como entrada un numpy array de dimensión $n \times 4$ con los parámetros de Denavit-Hartenberg de un manipulador cualquiera, y devuelva como salida la matriz de transformación homogénea (como un numpy array de dimensión 4×4) que relaciona el sistema de coordenadas del efector y el sistema de coordenadas de la base. Contemple la posibilidad de que los ángulos de rotación y los desplazamientos sean variables simbólicas (use el paquete sympy). Compruebe el correcto funcionamiento de la función con algunos ejemplos.

Para implementar la función he usado un "bucle for"que recorra las diferentes filas de la matriz de entrada y para cada una de las cuatros variables genere su matriz de rotación o de traslación correspondiente. Una vez obtenidas las matrices que llamare rz, dz, dx y rx las multiplicara para obtener la matriz ${}^{i}T_{i+1}$ que multiplicada por la matriz homogénea anterior nos dará ${}^{0}T_{i+1}$. En el primer paso del "bucle for"la matriz homogénea sera la matriz identidad. Cuando termina el "bucle forza tenemos la matriz homogénea pa-

ra pasar desde 0 hasta n. Esto se puede ver de forma mas clara en el código que se adjunta.

Una vez implementada la función, pase a comprobar que funcionaba usando el mismo ejemplo de las transparencias de clase, para tener un ejemplo que se que era correcto. Teniendo una entrada como la siguiente:

q	θ_i	d_i	a_i	α_i
1	q_1	l_1	0	0
2	90°	q_2	0	90°
3	0	$l_3 + q_3$	0	0

Debemos obtener (y obtenemos) el siguiente resultado:

$$\begin{pmatrix} -sin(q_1) & 0 & cos(q_1) & (l_3 + q_3) * cos(q_1) \\ cos(q_1) & 0 & sin(q_1) & (l_3 + q_3) * sin(q_1) \\ 0 & 1 & 0 & (l_1 + q_2) \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Podemos ver el resultado de la ejecución en la Figura 10

```
Los datos de entrada han sido:
[q1 l1 0 0]
[1.5707963267948966 q2 0 1.5707963267948966]
[0 l3 + q3 0 0]
El resulado han sido:
[-sin(q1), 0, cos(q1), (l3 + q3)*cos(q1)]
[cos(q1), 0, sin(q1), (l3 + q3)*sin(q1)]
[0, 1, 0, l1 + q2]
[0, 0, 0, 1]
```

Figura 10: Resultado de la función usando los datos de ejemplo

También lo he comprobado usando los datos del "Ejercicio 1" de la relación de ejercicios 3. Los datos de entrada serán:

q	θ_i	d_i	a_i	α_i
1	q_1	0	l_1	0
2	q_2	0	l_2	0

Y debemos de obtener como resultado:

$$\begin{pmatrix} cos(q_1)*cos(q_2)-sin(q_1)*sin(q_2) & -sin(q1)*cos(q2)-sin(q2)*cos(q1) & 0 & l1*cos(q1)-l2*sin(q1)*sin(q2)+l2*cos(q1)*cos(q2) \\ sin(q1)*cos(q2)+sin(q2)*cos(q1) & -sin(q1)*sin(q2)+cos(q1)*cos(q2) & 0 & l1*sin(q1)+l2*sin(q1)*cos(q2)+l2*sin(q2)*cos(q1) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Que es justo lo que obtenemos como podemos ver en la Figura 11.

```
Los datos de entrada han sido:
[q1 0 l1 0]
[q2 0 l2 0]
El resulado han sido:
[-sin(q1)*sin(q2) + cos(q1)*cos(q2), -sin(q1)*cos(q2) - sin(q2)
*cos(q1), 0, l1*cos(q1) - l2*sin(q1)*sin(q2) + l2*cos(q1)*cos(q
2)]
[sin(q1)*cos(q2) + sin(q2)*cos(q1), -sin(q1)*sin(q2) + cos(q1)*
cos(q2), 0, l1*sin(q1) + l2*sin(q1)*cos(q2) + l2*sin(q2)*cos(q1
)]
[0, 0, 1, 0]
[0, 0, 0, 1]
```

Figura 11: Resultado de la función usando los datos del ejercicio 1

Los resultados se pueden comprobar ejecutando el programa "Denavit_Hatenberg.py".

Archivos de código

ejercicio1.py

```
import numpy as np
import matplotlib.pyplot as plt
import mpl_toolkits.mplot3d
import math
pxB = np.array ([ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 10,
                  10, 10, 10, 10, 10, 10, 10, 10, 10, 11, 12, 13,
                  14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14])
pyB = np.array ([ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
                  2, 3, 4, 5, 6, 7, 8, 9, 10, 10, 10, 10,
                  10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0])
pzB = np.array ([ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
pB = np.array ([ pxB, pyB, pzB ])
pA = np.copy(pB)



# Apartado a
fig1 = plt.figure ()
ax = fig1.add_subplot (111 , projection ='3d')
ax.title.set_text("Apartado A")
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')
ax.plot(pA[0],pA[1],pA[2],'.')
#Creo una funcion lambda para calcular rapida la matriz de rotación de x
rx= lambda x:np.asarray([
        [1,0,0],
        [0,math.cos(x*math.pi/180),-math.sin(x*math.pi/180)],
        [0,math.sin(x*math.pi/180),math.cos(x*math.pi/180)]],
        dtype=np.float32)
#Genero la matriz de rotación de 90Âº en x
rotacion =rx(90)
#Multiplico todos los puntos por la matriz de rotacion
pB1 = rotacion.dot(pB)
# Pinto la trama rotada
ax.plot(pB1[0],pB1[1],pB1[2],'.')
```

```
# Apartado b
fig2 = plt.figure ()
ax1 = fig2.add_subplot (111 , projection ='3d')
ax1.title.set_text("Apartado B")
ax1.set_xlabel('x')
ax1.set_ylabel('y')
ax1.set_zlabel('z')
ax1.plot(pA[0],pA[1],pA[2],'.')
#Creo una funcion lambda para calcular rapida la matriz de rotación de y
ry= lambda x:np.asarray([
        [math.cos(x*math.pi/180),0,math.sin(x*math.pi/180)],
        [0,1,0],
        [-math.sin(x*math.pi/180),0,math.cos(x*math.pi/180)]],
        dtype=np.float32)
#Genero la matriz de rotación de 90ÂºC en y
rotacion =ry(90)
#Multiplico todos los puntos por la matriz de rotacion
pB2 = rotacion.dot(pB)
# Pinto la trama rotada
ax1.plot(pB2[0],pB2[1],pB2[2],'.')

# Apartado c
fig3 = plt.figure ()

ax2 = fig3.add_subplot (111 , projection ='3d')
ax2.title.set_text("Apartado C")
ax2.set_xlabel('x')
ax2.set_ylabel('y')
ax2.set_zlabel('z')
ax2.plot(pA[0],pA[1],pA[2],'.')
#Creo una funcion lambda para calcular rapida la matriz de rotación de z
rz= lambda x:np.asarray([
        [math.cos(x*math.pi/180),-math.sin(x*math.pi/180),0],
        [math.sin(x*math.pi/180),math.cos(x*math.pi/180),0],
        [0,0,1]],
        dtype=np.float32)
#Genero la matriz de rotación de 90ÂºC en z
rotacion =rz(90)
#Multiplico todos los puntos por la matriz de rotación
pB3 = rotacion.dot(pB)
```

```
# Pinto la trama rotada
ax2.plot(pB3[0],pB3[1],pB3[2],'.')
```

ejercicio2.py

```
import numpy as np
import matplotlib.pyplot as plt
import math
pxB = np.array ([ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 10,
                  10, 10, 10, 10, 10, 10, 10, 10, 10, 11, 12, 13,
                  14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14])
pyB = np.array ([ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
                  2, 3, 4, 5, 6, 7, 8, 9, 10, 10, 10, 10,
                  10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0])
pzB = np.array ([ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
pB = np.array ([ pxB, pyB, pzB ])
pA = np.copy(pB)

#Funciones lambda con las que puedes obtener
#la rotación para cualquiera angulo en cualquier eje
rx= lambda x:np.asarray([
        [1,0,0],
        [0,math.cos(x*math.pi/180),-math.sin(x*math.pi/180)],
        [0,math.sin(x*math.pi/180),math.cos(x*math.pi/180)]],
        dtype=np.float32)
ry= lambda x:np.asarray([
        [math.cos(x*math.pi/180),0,math.sin(x*math.pi/180)],
        [0,1,0],
        [-math.sin(x*math.pi/180),0,math.cos(x*math.pi/180)]],
        dtype=np.float32)
rz= lambda x:np.asarray([
        [math.cos(x*math.pi/180),-math.sin(x*math.pi/180),0],
        [math.sin(x*math.pi/180),math.cos(x*math.pi/180),0],
        [0,0,1]],
        dtype=np.float32)

primera_r = rx(60)
```

```
segunda_r = ry(90)
tercera_r = rz(30)


fig1 = plt.figure ()
ax1 = fig1.add_subplot (111 , projection ='3d')
ax1.title.set_text("1Â° rotación en X")
ax1.set_xlabel('x')
ax1.set_ylabel('y')
ax1.set_zlabel('z')
ax1.plot(pA[0],pA[1],pA[2],'.')

pB1 = primera_r.dot(pB)
# Pinto la trama rotada
ax1.plot(pB1[0],pB1[1],pB1[2],'.')


fig2 = plt.figure ()
ax2 = fig2.add_subplot (111 , projection ='3d')
ax2.title.set_text("2Â° rotación en y")
ax2.set_xlabel('x')
ax2.set_ylabel('y')
ax2.set_zlabel('z')
ax2.plot(pA[0],pA[1],pA[2],'.')

pB2 = segunda_r.dot(pB1)
# Pinto la trama rotada
ax2.plot(pB2[0],pB2[1],pB2[2],'.')


fig3 = plt.figure ()
ax3 = fig3.add_subplot (111 , projection ='3d')
ax3.title.set_text("3Â° rotación en z")
ax3.set_xlabel('x')
ax3.set_ylabel('y')
ax3.set_zlabel('z')
ax3.plot(pA[0],pA[1],pA[2],'.')

pB3 = tercera_r.dot(pB2)
# Pinto la trama rotada
ax3.plot(pB3[0],pB3[1],pB3[2],'.')
```

```
#Ahora veremos que pasa si las reotaciones se realizan en otro orden.
fig4 = plt.figure ()

ax4 = fig4.add_subplot (111 , projection ='3d')
ax4.title.set_text("Orden Alterado 1Âº: Rotación Y")
ax4.set_xlabel('x')
ax4.set_ylabel('y')
ax4.set_zlabel('z')
ax4.plot(pA[0],pA[1],pA[2],'.')
fig4.show()
pB1 = segunda_r.dot(pB)
# Pinto la trama rotada
ax4.plot(pB1[0],pB1[1],pB1[2],'.')

fig5 = plt.figure ()
ax5 = fig5.add_subplot (111 , projection ='3d')
ax5.title.set_text("Orden Alterado 2Âº: Rotación Z")
ax5.set_xlabel('x')
ax5.set_ylabel('y')
ax5.set_zlabel('z')
ax5.plot(pA[0],pA[1],pA[2],'.')

pB2 = tercera_r.dot(pB1)
# Pinto la trama rotada
ax5.plot(pB2[0],pB2[1],pB2[2],'.')

fig6 = plt.figure ()
ax6 = fig6.add_subplot (111 , projection ='3d')
ax6.title.set_text("Orden Alterado 3Âº: Rotación X")
ax6.set_xlabel('x')
ax6.set_ylabel('y')
ax6.set_zlabel('z')
ax6.plot(pA[0],pA[1],pA[2],'.')

pB3 = primera_r.dot(pB2)
# Pinto la trama rotada
ax6.plot(pB3[0],pB3[1],pB3[2],'.')
```

Denavit_Hartenberg.py

```
import numpy as np
import sympy as sp
def algoritmo(parametros):
    #Creo la matriz homogenea como la matriz identidad
    #para que en la primera multiplicacion no cambie nada
    matriz_homogenea = np.array([[1,0,0,0],
                                 [0,1,0,0],
                                 [0,0,1,0],
                                 [0,0,0,1]])
    for i in parametros:
        i = np.asarray(i).flatten()
        #Ponemos las cuatros matrices, las dos de rotación
        #y las dos de translacion
        rz = np.array([[sp.cos(i[0]),-sp.sin(i[0]),0,0],
                    [sp.sin(i[0]),sp.cos(i[0]),0,0],
                    [0,0,1,0],
                    [0,0,0,1]])
        dz = np.array([[1,0,0,0],[0,1,0,0],[0,0,1,i[1]],[0,0,0,1]])
        dx= np.array([[1,0,0,i[2]],[0,1,0,0],[0,0,1,0],[0,0,0,1]])
        #Realizamos la multiplicaciones para obtener el paso de  i a i+1
        rx = np.array([[1,0,0,0],
                       [0,sp.cos(i[3]),-sp.sin(i[3]),0],
                       [0,sp.sin(i[3]),sp.cos(i[3]),0],
                       [0,0,0,1]])
        resultado = rz.dot(dz)
        resultado = resultado.dot(dx)
        resultado = resultado.dot(rx)

        #Multiplicamos para pasar del 0 a i
        matriz_homogenea = matriz_homogenea.dot(resultado)

    return matriz_homogenea

#Creo los simobolos qeu voy a usar en este ejemplo
q1 , l1,l2, q2, l3, q3 = sp.symbols ('q1,l1,l2,q2,l3,q3')
#Creo la matriz de entrada para el denavit_hatemberg, en este caso de 3x4
entrada = np.array([[q1,l1,0,0],
                    [np.pi/2,q2,0,np.pi/2],
```

```
                    [0,l3+q3,0,0]])
#Ejecuto el algoritmo
resultado = algoritmo(entrada)
#Simplifico el resultado
resultado = sp.nsimplify(resultado,tolerance=1e-10)
#Y finalmente lo muestro
print("Los datos de entrada han sido:")
for i in entrada:
    print(i)
print("El resulado han sido:")
#Lo realizo asi para que se vea algo mejor
for i in resultado:
    print(i)

print("----------Otro ejemplo-----------")
entrada2 = np.array([[q1,0,l1,0],[q2,0,l2,0]])
resultado2 = algoritmo(entrada2)
resultado2 = sp.nsimplify(resultado2,tolerance=1e-10)
print("Los datos de entrada han sido:")
for i in entrada2:
    print(i)
print("El resulado han sido:")
for i in resultado2:
    print(i)
```