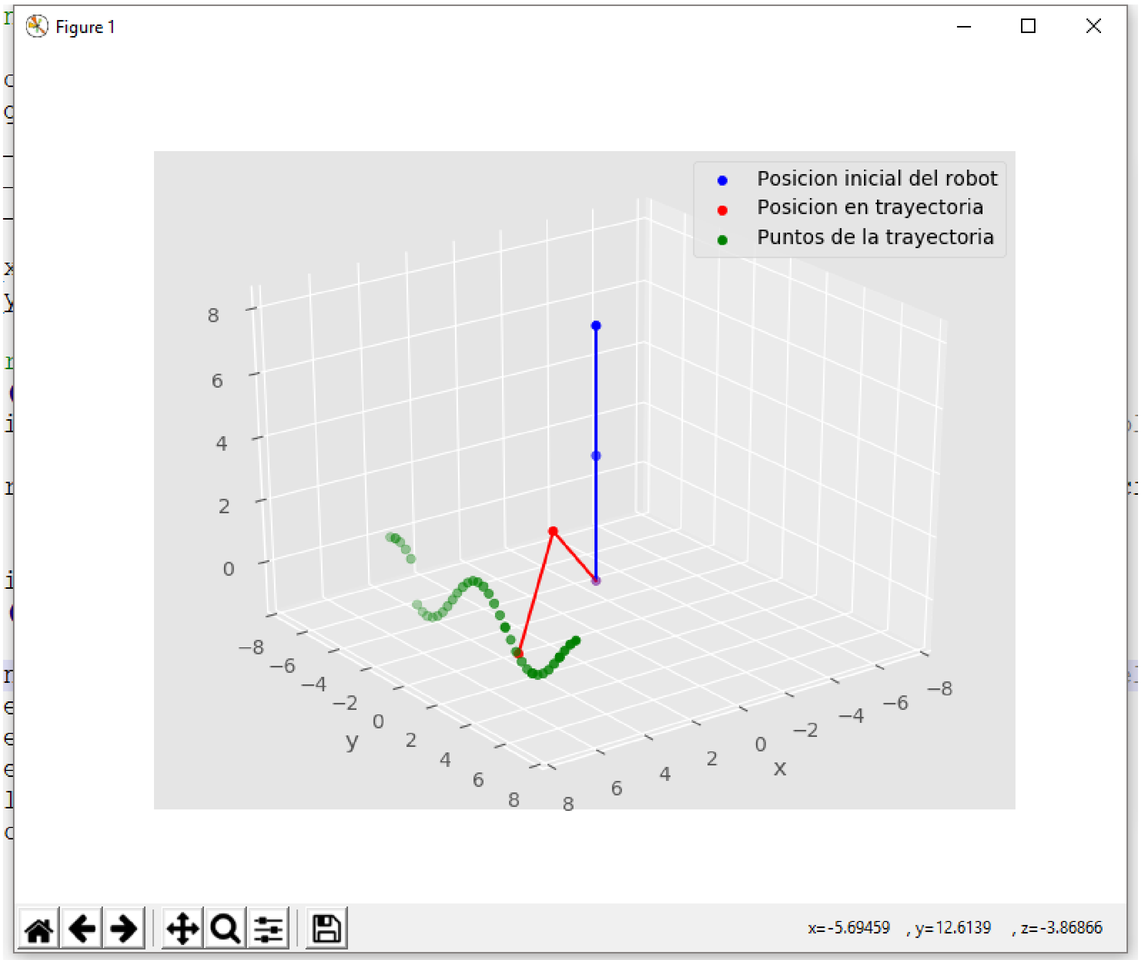
Estudio cinemático del robot manipulador PUMA



Librería desarrollada en ejecución

Alejandro Lobo Porras

Guillermo Marco Remón

Noviembre de 2018

Contenido

M1. Configuración geométrica

La primera parte es completamente teórica y consta de la definición de referenciales que permiten describir todos los movimientos que realiza el brazo y de la tabla de parámetros de Denavit-Hartenberg asociada con tales referenciales.

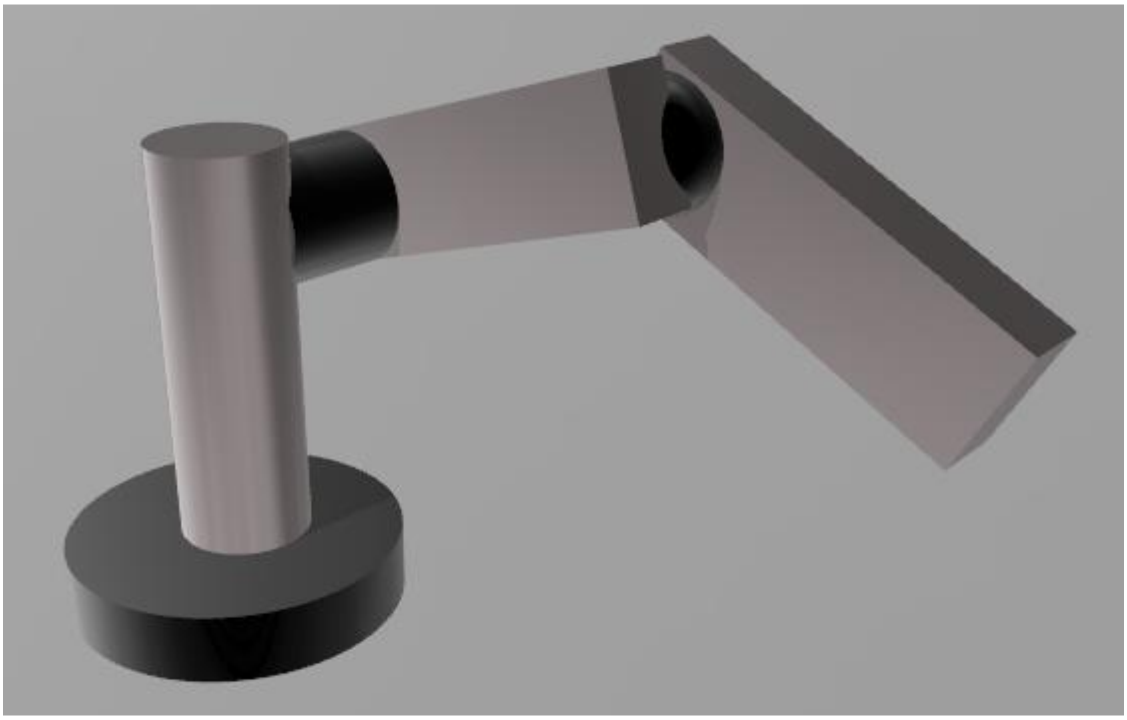


Figura 1.1 Representación esquemática del manipulador Puma a estudiar.

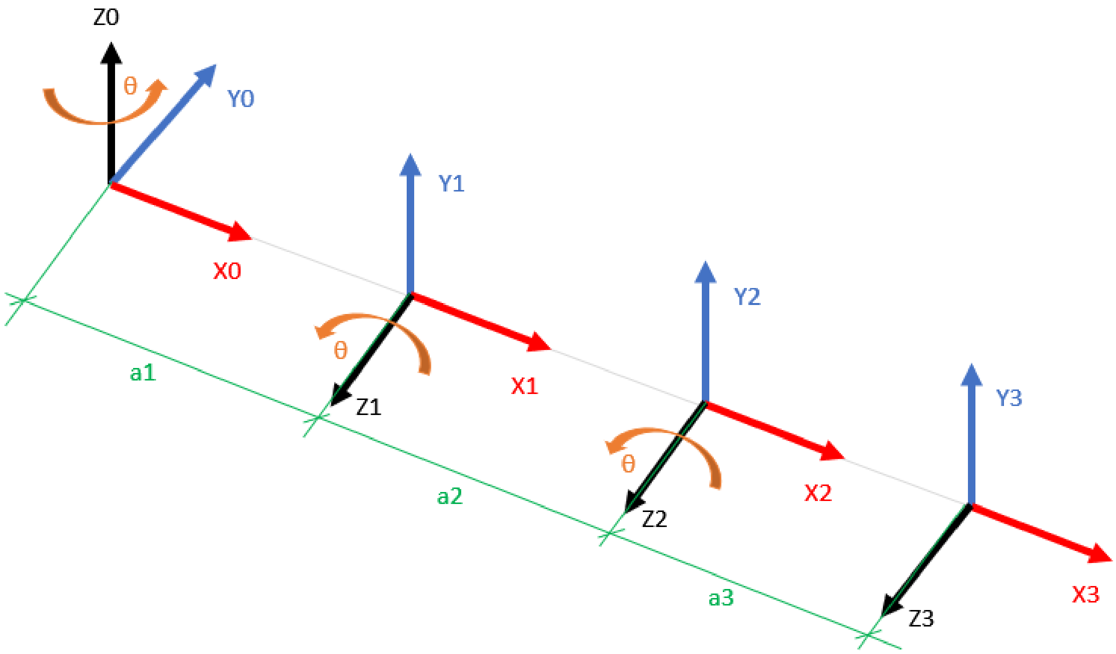


Figura 1.2 Referenciales que describen el movimiento del robot

i	θ_i	d_i	a_i	α_i
1	θ_1	0	0	$\pi/2$
2	θ_2	0	a_2	0
3	θ_3	0	a_3	0

Tabla 1: Parámetros de Denavit-Hartenberg

La programación de la librería del manipulador se compone de un objeto RobotPUMA(), cuyo constructor recibe los parámetros de Denavit-Hartenberg. Los parámetros variables de las distintas cinemáticas se pasan a la función correspondiente del objeto (ver Apéndice 1).

M2.1 Cinemática directa

Ecuaciones de la cinemática directa en forma matricial del manipulador y el estudio de los casos $\vec{q} = (0,0,0), \vec{q} = (0, \pi/2, 0), \vec{q} = (-\pi/2, \pi/2, 0)$ y $\vec{q} = (\pi, 0, \pi/2)$.

$$A_0^1 = \begin{pmatrix} C_1 & 0 & S_1 & 0 \\ S_1 & 0 & -C_1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad A_1^2 = \begin{pmatrix} C_2 & -S_2 & 0 & a_2C_2 \\ S_2 & C_2 & 0 & a_2S_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Vamos a calcular la matriz A_0^2 que sale del producto: $A_0^2 = A_0^1 * A_1^2$

$$A_0^2 = \begin{pmatrix} C_1C_2 & -C_1S_2 & S_1 & a_2C_1C_2 \\ S_1C_2 & -S_1S_2 & C_1 & a_2S_1S_2 \\ S_2 & C_2 & 0 & a_2S_2 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$A_2^3 = \begin{pmatrix} C_3 & -S_3 & 0 & a_3C_3 \\ S_3 & C_3 & 0 & a_3S_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Por último, vamos a calcular la matriz A_0^3 que sale del producto: $A_0^3 = A_0^2 * A_2^3$

$$A_0^3 \begin{pmatrix} C_1C_2C_3 - C_1S_2S_3 & -C_1C_2S_3 - C_1S_2C_3 & S_1 & a_3C_1C_2C_3 - a_3C_1S_2S_3 + a_2C_1C_2 \\ S_1C_2C_3 - S_1S_2S_3 & -S_1C_2S_3 - S_1S_2C_3 & -C_1 & a_3S_1C_2C_3 - a_3S_1S_2S_3 + a_2S_1S_2 \\ S_2C_3 + C_2S_3 & C_{23} & 0 & a_3S_2C_3 + a_3C_2S_3 + a_2S_2 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

La matriz A_0^3 obtenida puede ser simplificada para operar de manera más simple con ella usando las siguientes propiedades de los senos y cosenos:

$$sen(a+b) = sen\ a * \cos b + \cos a * sen\ b$$

$$sen(a-b) = sen\ a * \cos b - \cos a * sen\ b$$

$$cos(a+b) = cos\ a * \cos b + \mathrm{sen}\, a * sen\ b$$

$$cos(a-b) = cos\ a * \cos b - \mathrm{sen}\, a * sen\ b$$

Entonces para la posición (1,1) aplicando las propiedades, sacando factor común y simplificando quedaría:

$$C_1C_2C_3 - C_1S_2S_3 = C_1(C_2C_3 - S_2S_3) = C_1C_{23}$$

Para la posición (1,2):

$$-C_1C_2S_3 - C_1S_2C_3 = -C_1(C_2S_3 - S_2C_3) = -C_1S_{23}$$

Para la posición (1,4):

$$a_3C_1C_2C_3 - a_3C_1S_2S_3 + a_2C_1C_2 = a_3C_1(C_2C_3 - S_2S_3) + a_2C_1C_2 = a_2C_1C_2 + a_3C_1C_{23}$$

Para la posición (2,1):

$$S_1C_2C_3 - S_1S_2S_3 = S_1(C_2C_3 - S_2S_3) = S_1C_{23}$$

Para la posición (2,2):

$$-S_1C_2S_3 - S_1S_2C_3 = -S_1(C_2S_3 + S_2C_3) = -S_1S_{23}$$

Para la posición (2,4):

$$a_3S_1C_2C_3 - a_3S_1S_2S_3 + a_2S_1S_2 = a_3S_1(C_2C_3 - S_2S_3) + a_2S_1S_2 = a_2C_2S_1 + a_3S_1C_{23}$$

Para la posición (3,1):

$$S_2C_3 + C_2S_3 = S_{23}$$

Para la posición (3,4):

$$a_3S_2C_3 + a_3C_2S_3 + a_2S_2 = a_3(S_2C_3 + C_2S_3) + a_2S_2 = a_2S_2 + a_3S_{23}$$

Quedando de la forma:

$$A_0^3 \begin{bmatrix} C_1C_{23} & -C_1S_{23} & S_1 & a_2C_1C_2 + a_3C_1C_{23} \\ S_1C_{23} & -S_1S_{23} & -c_1 & a_2C_2S_1 + a_3S_1C_{23} \\ S_{23} & C_{23} & 0 & a_2S_2 + a_3S_{23} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Caso de estudio: $\vec{q} = (0,0,0)$

$$A_0^1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad A_1^2 = \begin{pmatrix} 1 & 0 & 0 & a_2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad A_2^3 = \begin{pmatrix} 1 & 0 & 0 & a_3 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$A_0^3 \begin{pmatrix} 1 & 0 & 0 & a_2 + a_3 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Caso de estudio: $\vec{q} = (0, \pi/2, 0)$

$$A_0^1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad A_1^2 = \begin{pmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & a_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad A_2^3 = \begin{pmatrix} 1 & 0 & 0 & a_3 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$A_0^3 \begin{pmatrix} 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 1 & 1 & 0 & a_3 + a_2 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Caso de estudio: $\vec{q} = (-\pi/2, \pi/2, 0)$

$$A_0^1 = \begin{pmatrix} 0 & 0 & -1 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} A_1^2 = \begin{pmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & a_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad A_2^3 = \begin{pmatrix} 1 & 0 & 0 & a_3 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$A_0^3 \begin{pmatrix} 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & -a2 \\ 1 & 0 & 0 & a3 + a2 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Caso de estudio: $\vec{q} = (\pi, 0, \pi/2)$

$$A_0^1 = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad A_1^2 = \begin{pmatrix} 1 & 0 & 0 & a_2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad A_2^3 = \begin{pmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & a_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$A_0^3 \begin{pmatrix} 0 & 1 & 0 & -a_2 \\ 0 & 0 & -1 & 0 \\ 1 & 1 & 0 & a_3 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

M1.2 Programación cinemática directa

El método principal utiliza otro que devuelve la matriz A de DH para unos parámetros dados. La función devuelve las coordenadas de cada referencial adjunto a cada articulación para poder graficar el brazo completo. Los parámetros constantes de DH se han establecido en el constructor del objeto.

```
#############################################
###         Cinematica directa            ###
#############################################

def cinematica_directa(self, theta):
    self.theta = theta;

    #Matriz A de cada eje respecto al anterior
    A10 = self.matrizA(self.theta[0] , self.d[0], self.a[0], self.alfa[0])
    A21 = self.matrizA(self.theta[1], self.d[1], self.a[1], self.alfa[1])
    A32 = self.matrizA(self.theta[2], self.d[2], self.a[2], self.alfa[2])

    #Multiplicamos respecto a la trama adjunta a la base
    A20 = np.dot(A10, A21)
    A30 = np.dot(A20, A32)

    #Devolvemos las coordenadas de cada trama adjunta a cada articulacion
    #para poder graficar el brazo entero
    return np.transpose([A10[0:3, 3], A20[0:3, 3], A30[0:3, 3]])

def matrizA(self, theta_i, d_i, a_i, alfa_i):
    return np.array([[m.cos(theta_i),   -m.sin(theta_i) * m.cos(alfa_i),    m.sin(theta_i) * m.sin(alfa_i),     a_i * m.cos(theta_i)], \
            [m.sin(theta_i),            m.cos(theta_i) * m.cos(alfa_i),    -m.sin(alfa_i)*m.cos(theta_i),      a_i * m.sin(theta_i)], \
            [0.0,                   m.sin(alfa_i),                          m.cos(alfa_i),                      d_i], \
            [0.0,                   0.0,                                    0.0,                                1.0]])
```

Figura M1.2.1 Código de la cinemática directa

Se han introducido los casos de prueba:

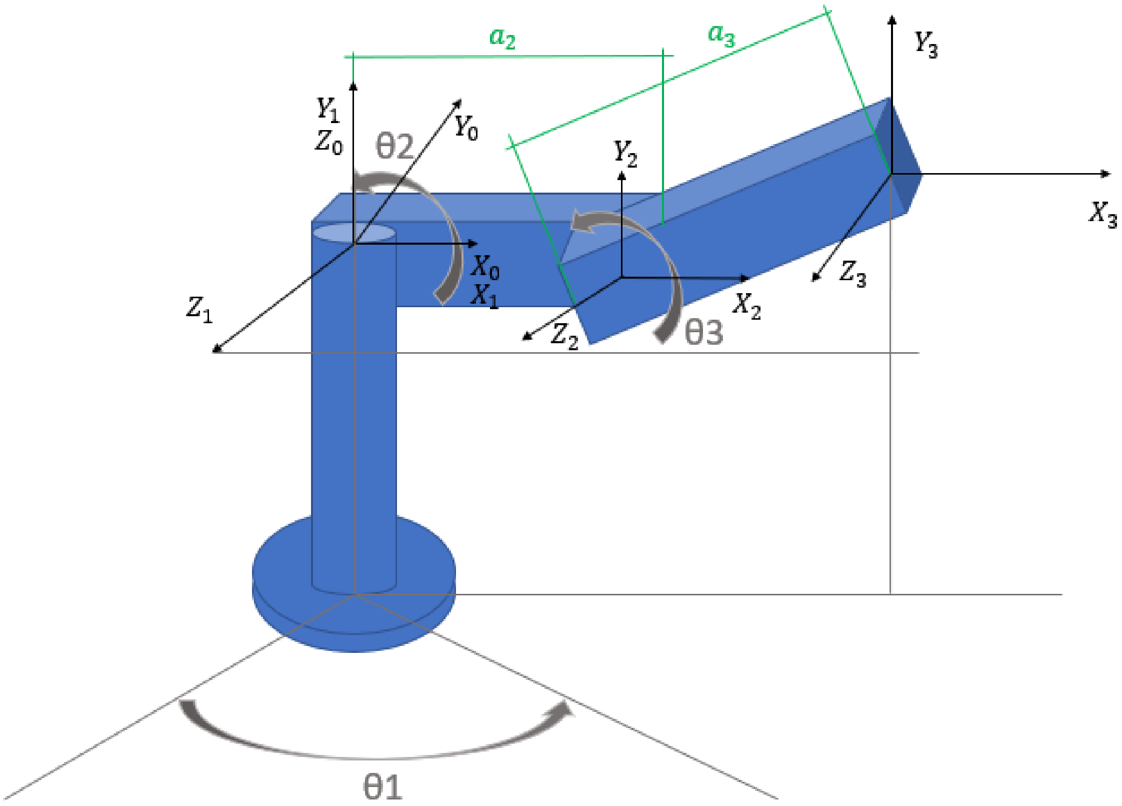
```
Símbolo del sistema
tre\Robotica\Robotica\programacion\PUMA_ROBOTICA.py"
#############################################
###              EJEMPLOS PROPORCIONADOS                ###
#############################################
Theta1: [0, 0, 0]
Theta2: [0, 1.5707963267948966, 0]
Theta3: [-1.5707963267948966, 1.5707963267948966, 0]
Theta4: [3.141592653589793, 0, 1.5707963267948966]

Directa 1:
[[0. 1. 2.]
 [0. 0. 0.]
 [0. 0. 0.]]
Directa 2:
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 1. 2.]]
Directa 3:
[[ 0.  0.  0.]
 [-0. -0. -0.]
 [ 0.  1.  2.]]
Directa 4:
[[-0. -1. -1.]
 [ 0.  0.  0.]
 [ 0.  0.  1.]]
```

Figura M1.2.2. Ejecución de casos de la cinemática directa para a1=1 y a2=1

Como se observa en la figura, dan los mismos resultados que los calculados analíticamente en el apartado anterior.

M3.1. Cinemática inversa

Se van a obtener las ecuaciones de la cinemática inversa del manipulador y se hará el estudio de los casos a los que se llega como resultado en el apartado anterior. Se compararán los $\vec{q}$ propuestos en dicho apartado.



Representación esquemática del manipulador Puma.

Vamos a solucionar el problema de la cinemática inversa del robot por métodos geométricos basándonos en el dibujo del manipulador Puma propuesto anteriormente. El primer que vamos a hallar el cual es inmediato es $\theta 1$, cómo podemos observar el robot posee una estructura planar quedando este plano definido por el ángulo anteriormente mencionado. Por lo que $\theta 1$ sería:

$$\theta_1 = arctg\left({}^{p_y}/_{p_x}\right)$$

Sabiendo el teorema del coseno y trabajando con las dos extremidades exteriores del robot podemos sacar la siguiente relación:

$$r^2 = p_x^2 + p_y^2$$

$$r^2 + p_x^2 = a_2^2 + a_3^2 + 2a_2a_3cos\theta_3$$

Para facilitar las cuentas vamos a usar la expresión de la arcotangente en lugar del arcoseno. Dado que:

$$sen\theta_3 = \pm\sqrt{1 - \cos^2\theta_3}$$

Obteniéndose que:

$$\theta_3 = arctg\left(\pm\sqrt{1-\cos^2\theta_3}\Big/_{cos\theta_3}\right)$$

Donde

$$cos\theta_3 = {p_x^2 + p_y^2 + p_z^2 - a_2^2 - a_3^2}\Big/_{2a_2a_3}$$

En este caso se tendrán dos posibles soluciones para q_3 según se tome el signo positivo o el negativo. Que serían las configuraciones de tener el codo hacia arriba o hacia abajo.

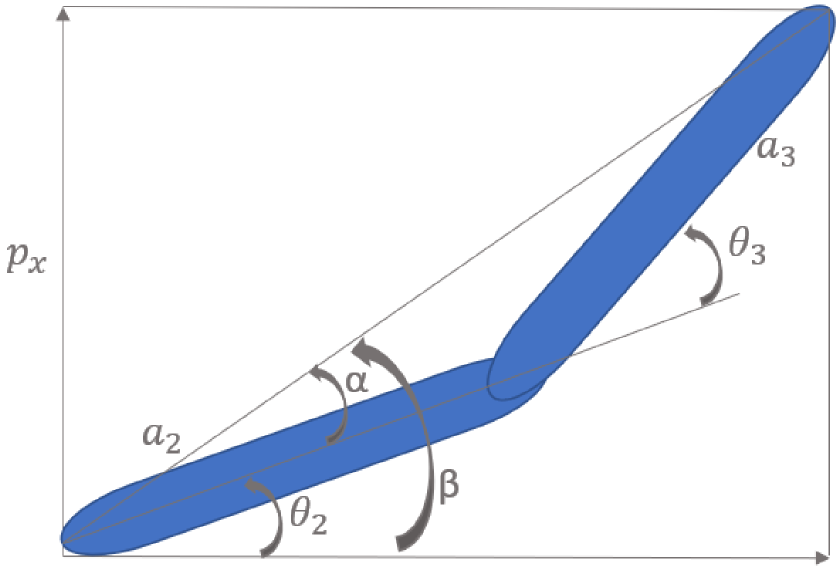


Figura 3.1 Esquema de movimiento de la parte del codo del Puma.

Por último, para calcular $\theta_2$vamos a hacer la diferencia entre β y α:

$$\theta_2 = \beta - \alpha$$

Donde:

$$\beta = arctg\left(\frac{p_z}{r}\right) = arctg\left(\frac{p_z}{\pm\sqrt{p_x^2 + p_y^2}}\right)$$

$$\alpha = arctg\left(\frac{a_3 sen\theta_3}{a_2 + a_3 cos\theta_3}\right)$$

Obteniendo así:

$$\theta_2 = arctg\left(\frac{p_z}{\pm\sqrt{p_x^2+p_y^2}}\right) - arctg\left(\frac{a_3 sen\theta_3}{a_2+a_3 cos\theta_3}\right)$$

Para el desarrollo de los cálculos se ha sustituido la longitud de las variables $a_2 y\ a_3$ por el valor 1 para así simplificar los cálculos, aunque se podría elegir cualquier otro valor.

Caso de estudio: $p_x = \mathrm{a}_2 + a_3,\ p_y = 0,\ p_z = 0$

$$\theta_1 = arctg\left(\frac{0}{a_2 + a_3}\right) = arctg\left(\frac{0}{1+1}\right) = 0$$

$$cos\theta_3 = \frac{(a_2 + a_3)^2 + 0^2 + 0^2 - 1^2 - 1^2}{2 * 1 * 1} = \frac{2}{2} = 1$$

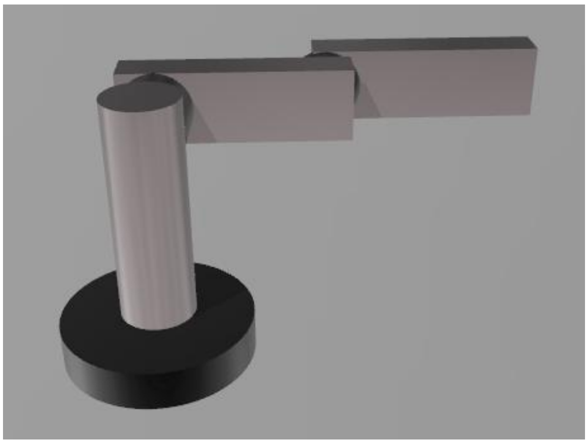
$$sen\theta_3 = \pm\sqrt{1 - \cos^2\theta_3} = \pm\sqrt{1 - (1)^2} = 0$$

$$\theta_2 = arctg\left(\frac{0}{\sqrt{(a_2 + a_3)^2 + 0^2}}\right) - arctg\left(\frac{1 * sen\theta_3}{1 + 1 * cos\theta_3}\right) = 0 - 0 = 0$$

$$\theta_3 = \frac{\pm\sqrt{1 - \cos^2\theta_3}}{cos\theta_3} = 0$$

Por lo que: $\theta_1 = 0;\ \theta_2 = 0;\ \theta_3 = 0.$

Quedando en brazo del robot en la siguiente posición:



Posición del manipulador para el caso de estudio: $\vec{q} = (0, 0, 0)$

Caso de estudio: $p_x = 0,\ p_y = 0,\ p_z = \mathrm{a}_3 + a_2$

$$\theta_1 = arctg\left(\frac{0}{0}\right) = \rightarrow Singularidad$$

Hay infinitas soluciones para $p_x = 0$. Por lo que $\theta_1$es singularidad.

$$cos\theta_3 = \frac{0^2 + 0^2 + (a_3 + a_2)^2 - 1^2 - 1^2}{2 * 1 * 1} = \frac{2}{2} = 1$$

$$sen\theta_3 = \pm\sqrt{1 - \cos^2\theta_3} = 0$$

$$\theta_2 = arctg\left(\frac{\mathrm{a}_3 + a_2}{\sqrt{0^2 + 0^2}}\right) - arctg\left(\frac{1 * sen\theta_3}{1 + 1 * cos\theta_3}\right) = 1{,}570796 = \frac{\pi}{2}$$

$$\theta_3 = arctg\left(\frac{\pm\sqrt{1 - \cos^2\theta_3}}{cos\theta_3}\right) = 0$$

Por lo que: $\theta_1 \rightarrow Singularidad;\ \theta_2 = \frac{\pi}{2};\ \theta_3 = 0.$

Posición del manipulador para el caso de estudio: $\vec{q} = (0, \frac{\pi}{2}, 0)$

Caso de estudio: $p_x = 0,\ p_y = -a_2,\ p_z = a_3 + a_2$

$$\theta_1 = arctg\left(\frac{-a_2}{0}\right) = arctg\left(\frac{-1}{0}\right) \rightarrow Singularidad$$

Hay infinitas soluciones de θ_1 para $p_x = 0$.

$$cos\theta_3 = \frac{0^2(-a_2)^2 + (a_3 + a_2)^{\ 2} - 1^2 - 1^2}{2 * 1 * 1} = \frac{3}{2}$$

$$sen\theta_3 = \pm\sqrt{1 - \cos^2\theta_3} = \nexists$$

$$\theta_2 = arctg\left(\frac{a_3 + a_2}{\pm\sqrt{0^2 + {a_2}^2}}\right) - arctg\left(\frac{1 * sen\theta_3}{1 + 1 * cos\theta_3}\right) = \frac{\pi}{2}$$

$$\theta_3 = arctg\left(\frac{\pm\sqrt{1 - \cos^2\theta_3}}{cos\theta_3}\right) = 0$$

Por lo que: $\theta_1 \rightarrow Singularidad;\ \theta_2 = \frac{\pi}{2};\ \theta_3 = 0.$

Posición del manipulador para el caso de estudio: $\vec{q} = (-\frac{\pi}{2}, \frac{\pi}{2}, 0)$

Caso de estudio: $p_x = -\mathrm{a}_2,\ p_y = 0,\ p_z = a_3$

$$\theta_1 = arctg\left(\frac{0}{-a_2}\right) = arctg\left(\frac{0}{-1}\right) = 0$$

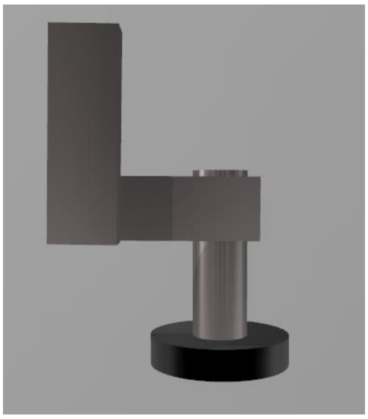
$$cos\theta_3 = \frac{-a_2^2 + 0^2 + a_3^2 - 1^2 - 1^2}{2 * 1 * 1} = -1$$

$$sen\theta_3 = \pm\sqrt{1 + \cos^2\theta_3} = 1{,}4647$$

$$\theta_2 = arctg\left(\frac{a_3}{\sqrt{-a_2^2 + 0^2}}\right) - arctg\left(\frac{1 * sen\theta_3}{1 + 1 * cos\theta_3}\right) = 0$$

$$\theta_3 = \frac{\pm\sqrt{1 + \cos^2\theta_3}}{cos\theta_3} \rightarrow \text{Singularidad para } \theta_3.$$

Por lo que: $\theta_1 = 0;\ \theta_2 = 0;\ \theta_3 \rightarrow Singularidad.$



Posición del manipulador para el caso de estudio: $\vec{q} = (\pi, 0, \frac{\pi}{2})$

M3.2. Programación de la cinemática inversa

Se parte de las ecuaciones de cinemática inversa del apartado anterior. Para evitar errores de ejecución se hace cierto control de singularidades (división por cero o raíces negativas).

En la figura M3.2.2. se observa que los resultados son iguales que los calculados. Cabe comentar que para los ejemplos 2 y 3 se produce una singularidad en theta1 y en theta3 para el ejemplo 4, tal y como se estudió.

```python
##############################################
###         Cinematica inversa             ###
##############################################

def cinematica_inversa(self, posicion):

    theta = [0 for i in range(3)]
    px = round(posicion[0])
    py = round(posicion[1])
    pz = round(posicion[2])

    #Conocemos analiticamente los valores que producen una singularidad


    #Theta1
    if(px != 0):
        theta[0] = m.atan2(py,px)
    else:
        theta[0] = "Infinitas soluciones para theta1"
    #Primero calculamos theta3, ya que theta2 se resuelve en funcion de esta
    cos_theta3 = (m.pow(px,2) + m.pow(py,2) + m.pow(pz,2) - m.pow(a[1],2) - m.pow(a[2],2))/(2*a[1]*a[2])
    cos_theta3_cuadrado = m.pow(cos_theta3, 2)

    #Posible situacion en que el coseno calculado geometricamene con la posicion del extremo
    #sea mayor que uno (no se puede alcanzar el punto por la logitud del brazo)
    #o el coseno, que se encuentra en el denominador de theta3
    #es cero
    if(abs(cos_theta3)>1 or 0==cos_theta3):
        theta[2] = "Singularidad theta3"
    else:
        theta[2] = m.atan2(m.sqrt(1-cos_theta3_cuadrado), cos_theta3)

    #Finalmente calculamos theta2



    sen_theta3 = m.sqrt(1-m.pow(cos_theta3,2))

    beta = m.atan2(m.sqrt(m.pow(px,2)+m.pow(px,2)), pz)
    #gamma = m.atan2(a[1] + a[2]*m.cos(theta[2]), a[2]*m.sin(theta[2]))
    gamma = m.atan2(a[1] + a[2]*cos_theta3, a[2]*sen_theta3)
    theta[1] = gamma - beta

    return theta
```

Figura M3.2.1. Código de la cinemática inversa

```
Inversa 1:
[0.0, 0.0, 0.0]
Inversa 2:
['Infinitas soluciones para theta1', 1.5707963267948966, 0.0]
Inversa 3:
['Infinitas soluciones para theta1', 1.5707963267948966, 0.0]
Inversa 4:
[3.141592653589793, -0.16991845472706102, 'Singularidad theta3']
```

Figura M3.2.2. Ejecución para los ejemplos del enunciado de la cinemática inversa

El resultado para theta 2 de Inversa 4 no es exactamente cero debido a que el coseno se obtiene por métodos geométricos, no directamente con el ángulo y se acumula cierto error.

M4.1. Jacobiana

Matriz jacobiana del manipulador y el estudio de los casos $\vec{q}$=(0,0,π/2) y $\vec{q}$=(0,0,π/90); $\vec{q}$=(0,π/4,−π/4) y $\vec{q}$=(0,−π/90,π/90); $\vec{q}$=(π /2,0,−π/2) y $\vec{q}$=(0,π/90,0); y $\vec{q}$=(0,π/2,0) y $\vec{q}$=(0,0,−π/90). Para todos los casos se expresa gráficamente la situación de la que parte en manipulador.

$$J = \begin{bmatrix} \frac{\delta x_e}{\delta\theta_1} & \frac{\delta x_e}{\delta\theta_2} & \frac{\delta x_e}{\delta\theta_3} \\ \frac{\delta x_y}{\delta\theta_1} & \frac{\delta x_y}{\delta\theta_2} & \frac{\delta x_y}{\delta\theta_3} \\ \frac{\delta x_z}{\delta\theta_1} & \frac{\delta x_z}{\delta\theta_2} & \frac{\delta x_z}{\delta\theta_3} \\ \frac{\delta \phi_x}{\delta\theta_1} & \frac{\delta \phi_x}{\delta\theta_2} & \frac{\delta \phi_x}{\delta\theta_3} \\ \frac{\delta \phi_y}{\delta\theta_1} & \frac{\delta \phi_y}{\delta\theta_2} & \frac{\delta \phi_y}{\delta\theta_3} \\ \frac{\delta \phi_z}{\delta\theta_1} & \frac{\delta \phi_z}{\delta\theta_2} & \frac{\delta \phi_z}{\delta\theta_3} \end{bmatrix}$$

Donde las coordenadas x, y, z han sido sacadas directamente de la cinemática directa calculada en el apartado M1:

$$x = a_3C_1C_2C_3 - a_3C_1S_2S_3 + a_2C_1C_2$$

$$y = a_3S_1C_2C_3 - a_3S_1S_2S_3 + a_2S_1S_2$$

$$z = a_3S_2C_3 + a_3C_2S_3 + a_2S_2$$

Lo primero que se va a hacer para calcular la matriz jacobiana J es dividir el proceso en dos, calculando primero J_1que se corresponde con ____, y posteriormente se calculará J_2que se corresponde con___.

$$\begin{bmatrix} x_e \\ y_e \\ z_e \end{bmatrix} = \begin{bmatrix} a_3C_1C_2C_3 - a_3C_1S_2S_3 + a_2C_1C_2 \\ a_3S_1C_2C_3 - a_3S_1S_2S_3 + a_2S_1S_2 \\ a_3S_2C_3 + a_3C_2S_3 + a_2S_2 \end{bmatrix} \quad J_1 = \begin{bmatrix} \frac{\delta x_e}{\delta\theta_1} & \frac{\delta x_e}{\delta\theta_2} & \frac{\delta x_e}{\delta\theta_3} \\ \frac{\delta x_y}{\delta\theta_1} & \frac{\delta x_y}{\delta\theta_2} & \frac{\delta x_y}{\delta\theta_3} \\ \frac{\delta x_z}{\delta\theta_1} & \frac{\delta x_z}{\delta\theta_2} & \frac{\delta x_z}{\delta\theta_3} \end{bmatrix} \rightarrow$$

$$\begin{bmatrix} -a_3S_1C_2C_3 + a_3S_1S_2S_3 - a_2S_1C_2 & -a_3C_1S_2C_3 - a_3C_1C_2S_3 - a_2C_1S_2 & -a_3C_1C_2S_3 - a_3C_1S_2C_3 \\ a_3C_1C_2C_3 - a_3C_1S_2S_3 + a_2C_1S_2 & -a_3S_1S_2C_3 - a_3S_1C_2S_3 + a_2S_1C_2 & -a_3S_1C_2S_3 - a_3S_1S_2C_3 \\ 0 & a_3C_2C_3 - a_3S_2S_3 + a_2C_2 & -a_3S_2S_3 + a_3C_2C_3 \end{bmatrix}$$

La matriz J_1 obtenida puede ser simplificada para operar de manera más simple con ella usando las siguientes propiedades de los senos y cosenos:

$$sen(a+b) = sen\ a * \cos b + \cos a * sen\ b$$

$$sen(a-b) = sen\ a * \cos b - \cos a * sen\ b$$

$$cos(a+b) = cos\ a * \cos b + \text{sen}\ a * sen\ b$$

$$cos(a-b) = cos\ a * \cos b - \operatorname{sen} a * sen\ b$$

Entonces para la posición (1,1) aplicando las propiedades, sacando factor común y simplificando quedaría:

$$-a_3S_1C_2C_3 + a_3S_1S_2S_3 - a_2S_1C_2 = -a_3S_1(+C_2C_3 - S_1S_3) - a_2S_1C_2 = -a_3S_1C_{23}\ - a_2S_1C_2$$

Para la posición (1,2):

$$-a_3C_1S_2C_3 - a_3C_1C_2S_3 - a_2C_1S_2 = -a_3C_1(S_2C_3 + C_2S_3) - a_2C_1S_2 = -a_2C_1S_2 - a_3C_1S_{23}$$

Para la posición (1,3):

$$-a_3C_1C_2S_3 - a_3C_1S_2C_3 = -a_3C_1(C_2S_3 + S_2C_3) = -a_3C_1S_{23}$$

Para la posición (2,1):

$$a_3C_1C_2C_3 - a_3C_1S_2S_3 + a_2C_1S_2 = a_3C_1(C_2C_3 - S_2S_3) + a_2C_1S_2 = a_2C_1S_2 + a_3C_1C_{23}$$

Para la posición (2,2):

$$-a_3S_1S_2C_3 - a_3S_1C_2S_3 + a_2S_1C_2 = -a_3S_1(S_2C_3 + C_2S_3) + a_2S_1C_2 = a_2S_1C_2 - a_3S_1S_{23}$$

Para la posición (2,3):

$$-a_3S_1C_2S_3 - a_3S_1S_2C_3 = -a_3S_1(C_2S_3 + S_2C_3) = -a_3S_1S_{23}$$

Para la posición (3,2):

$$a_3C_2C_3 - a_3S_2S_3 + a_2C_2 = a_3(C_2C_3 - S_2S_3) + a_2C_2 = a_2C_2 + a_3C_{23}$$

Por último, para la posición (3,3):

$$-a_3S_2S_3 + a_3C_2C_3 = a_3(S_2S_3 - C_2C_3) = a_3C_{23}$$

Quedando entonces la matriz J_1de la siguiente forma:

$$J_1 = \begin{bmatrix} -a_3S_1C_{23}\ - a_2S_1C_2 & -a_2C_1S_2 - a_3C_1S_{23} & -a_3C_1S_{23} \\ a_2C_1S_2 + a_3C_1C_{23} & a_2S_1C_2 - a_3S_1S_{23} & -a_3S_1S_{23} \\ 0 & a_2C_2 + a_3C_{23} & a_3C_{23} \end{bmatrix}$$

Ahora se va a calcular la matriz J_2:

$$\begin{bmatrix}\phi_x\\ \phi_y\\ \phi_z\end{bmatrix} = \begin{bmatrix}0\\ 0\\ \theta_1+\theta_2+\theta_3\end{bmatrix} \quad J_2 = \begin{bmatrix}\frac{\delta\phi_x}{\delta\theta_1} & \frac{\delta\phi_x}{\delta\theta_2} & \frac{\delta\phi_x}{\delta\theta_3}\\ \frac{\delta\phi_y}{\delta\theta_1} & \frac{\delta\phi_y}{\delta\theta_2} & \frac{\delta\phi_y}{\delta\theta_3}\\ \frac{\delta\phi_z}{\delta\theta_1} & \frac{\delta\phi_z}{\delta\theta_2} & \frac{\delta\phi_z}{\delta\theta_3}\end{bmatrix} \rightarrow \begin{bmatrix}0 & 0 & 0\\ 0 & 0 & 0\\ 1 & 1 & 1\end{bmatrix}$$

Por lo que como se ha explicado juntando las matrices J_1 y J_2 obtenemos la matriz jacobiana J.

$$J = \begin{bmatrix}J_1\\ J_2\end{bmatrix}$$

$$J = \begin{bmatrix}-a_3S_1C_{23} - a_2S_1C_2 & -a_2C_1S_2 - a_3C_1S_{23} & -a_3C_1S_{23}\\ a_2C_1S_2 + a_3C_1C_{23} & a_2S_1C_2 - a_3S_1S_{23} & -a_3S_1S_{23}\\ 0 & a_2C_2 + a_3C_{23} & a_3C_{23}\\ 0 & 0 & 0\\ 0 & 0 & 0\\ 1 & 1 & 1\end{bmatrix}$$

Caso de estudio: $\vec{q}$=(0,0,π/2) y $\vec{q}$=(0,0,π/90)

$$J = \begin{bmatrix}0 & -a_3 & -a_3\\ 0 & 0 & 0\\ 0 & a_2 & 0\\ 0 & 0 & 0\\ 0 & 0 & 0\\ 1 & 1 & 1\end{bmatrix} * \begin{bmatrix}0\\ 0\\ \pi/90\end{bmatrix} = \begin{bmatrix}-a_3\pi/90\\ 0\\ 0\\ 0\\ 0\\ \pi/90\end{bmatrix} = \begin{bmatrix}-\pi/90\\ 0\\ 0\\ 0\\ 0\\ \pi/90\end{bmatrix}$$

Caso de estudio: $\vec{q}$=(0,π/4,−π/4) y $\vec{q}$=(0,−π/90,π/90)

$$J = \begin{bmatrix}0 & -a_30.5 - a_3 - 0.5 - a_20.7 & -a_3 - 0.5 - a_30.5\\ a_30.5 - a_3 - 0.5 + a_20.7 & 0 & 0\\ 0 & a_30.5 - a_{2\_}0.5 + a_20.7 & -a_3 - 0.5 + a_30.5\\ 0 & 0 & 0\\ 0 & 0 & 0\\ 1 & 1 & 1\end{bmatrix} =$$

$$\begin{bmatrix} 0 & -a_2 0.7 & 0 \\ a_3 + a_2 0.7 & 0 & 0 \\ 0 & a_3 0.5 + a_2 1.2 & a_3 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} * \begin{bmatrix} 0 \\ -\pi/90 \\ \pi/90 \end{bmatrix} = \begin{bmatrix} -a_2 0.7\left(-\frac{\pi}{90}\right) \\ 0 \\ \left(-\frac{\pi}{90}\right)(a_3 0.5 + a_2 1.2) + a_3 \frac{\pi}{90} \\ 0 \\ 0 \\ -\frac{\pi}{90} + \frac{\pi}{90} \end{bmatrix} =$$

$$\begin{bmatrix} a_2 0.7\left(\frac{\pi}{90}\right) \\ 0 \\ \left(-\frac{\pi}{90}\right)(a_3 0.5 + a_2 1.2) + a_3 \frac{\pi}{90} \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.7\left(\frac{\pi}{90}\right) \\ 0 \\ \left(-\frac{\pi}{90}\right)(0.5 + 1.2) + \frac{\pi}{90} \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Caso de estudio: $\vec{q}$=(π /2,0,−π/2) y $\vec{q}$=(0,π/90,0)

$$J = \begin{bmatrix} -a_2 & 0 & 0 \\ 0 & a_3 + a_2 & a_3 \\ 0 & a_2 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} * \begin{bmatrix} 0 \\ \pi/90 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ a_3\pi/90 + a_2\pi/90 \\ a_2\pi/90 \\ 0 \\ 0 \\ 0 \\ \pi/90 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ \pi/90 + \pi/90 \\ \pi/90 \\ 0 \\ 0 \\ 0 \\ \pi/90 \\ 0 \end{bmatrix}$$

Caso de estudio: $\vec{q}$= (0, π/2,0) y $\vec{q}$=(0,0,−π/90)

$$J = \begin{bmatrix} 0 & -a_3 - a_2 & -a_3 \\ a_2 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} * \begin{bmatrix} 0 \\ 0 \\ -\pi/90 \end{bmatrix} = \begin{bmatrix} a_3\pi/90 \\ 0 \\ 0 \\ 0 \\ 0 \\ -\pi/90 \end{bmatrix} = \begin{bmatrix} \pi/90 \\ 0 \\ 0 \\ 0 \\ 0 \\ -\pi/90 \end{bmatrix}$$

M4.2. Programación jacobianas

Se introduce en el código la expresión explicita simplificada en el apartado anterior. Se ha decido esto en lugar de hacer el código generalizado por sencillez y por el coste computacional que tiene calcular las derivadas. En la generación de trayectorias, que se verá más adelante, se necesita calcular cada jacobiana de cada punto de una secuencia de gran tamaño.

```
#############################################
###                    Jacobiana
#############################################

      #Escribimos la expresion explicita alcanzada derivando
      def jacobiana(self, theta):
            return np.array([[-
a[2]*m.sin(theta[0])*m.cos(theta[1]+theta[2]) -
a[1]*m.sin(theta[0])*m.cos(theta[1]),     -
a[1]*m.cos(theta[0])*m.sin(theta[1]) -
a[2]*m.cos(theta[0])*m.sin(theta[1]+theta[2]), -
a[2]*m.cos(theta[0])*m.sin(theta[1]+theta[2])],\
                        [a[1]*m.cos(theta[0])*m.sin(theta[1])+
a[2]*m.cos(theta[0])*m.cos(theta[1]+theta[2]),
a[1]*m.sin(theta[0])*m.cos(theta[1])
+a[2]*m.sin(theta[0])*m.sin(theta[1]+theta[2]),-
a[2]*m.sin(theta[0])*m.sin(theta[1]+theta[2])],\
                        [0,
      a[1]*m.cos(theta[1])+a[2]*m.cos(theta[1]+theta[2]),
      a[2]*m.cos(theta[1]+theta[2])],\
                        [0,   0, 0],\
                        [1, 1, 1]])

      def jacobiana_por(self, theta, q):
            return np.dot(self.jacobiana(theta),q)
```

Figura M4.2.1 Código para calcular la jacobiana y las coordenadas lineales a partir de un vector de coordenadas articulares

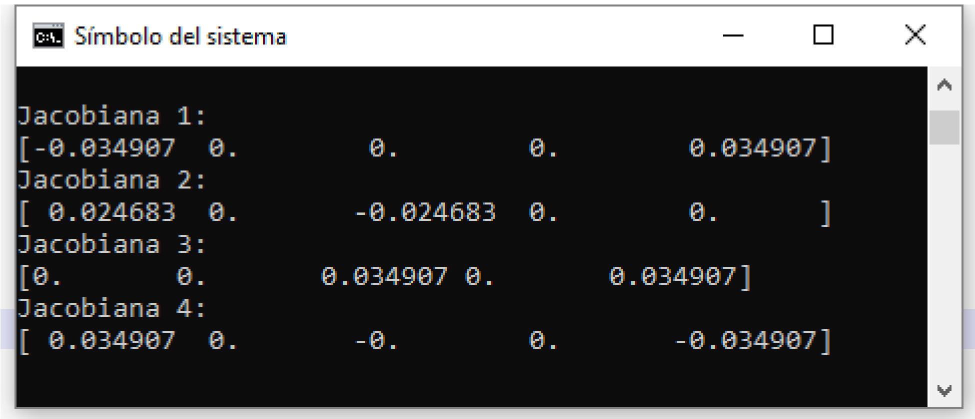


Figura M4.2.2 Ejecución de la multiplicación del vector de angulares por la jacobiana de los ejemplos proporcionados en el enunciado.

M5.1. Singularidades

Determinar los casos en los que el manipulador se encuentra en configuraciones singulares mediante demostración analítica. Comprobar si los casos de los apartados M2 y M4 corresponden con singularidades.

Con el fin de encontrar las configuraciones en los que el manipulador Puma se encuentra en una configuración singular lo primero que debemos realizar es calcular el determinante de la matriz jacobiana J_1:

$$J_1 = \begin{bmatrix} -a_3S_1C_{23} - a_2S_1C_2 & -a_2C_1S_2 - a_3C_1S_{23} & -a_3C_1S_{23} \\ a_2C_1S_2 + a_3C_1C_{23} & a_2S_1C_2 - a_3S_1S_{23} & -a_3S_1S_{23} \\ 0 & a_2C_2 + a_3C_{23} & a_3C_{23} \end{bmatrix}$$

$$\det J_1 = (-a_2S_1C_2 - a_3S_1C_{23})[(a_3C_{23})(-a_2S_1S_2 - a_3S_1S_{23}) + a_3S_1S_{23}(a_2C_2 + a_3C_{23})]$$

$$-(a_2C_1C_2 + a_3C_1C_{23})[(a_3C_{23})(-a_2S_2C_1 - a_3S_{23}C_1) + a_3C_1S_{23}(a_2C_2 + a_3C_{23})]$$

$$= a_2^2a_3(S_2C_2C_{23} - S_{23}C_2^2) + a_2a_3^2(S_2C_{23}^2 - S_{23}C_2C_{23})$$

$$= -a_2^2a_3C_2S_3 - a_2a_3^2C_{23}S_3$$

$$= a_2a_3S_3(a_2C_2 + a_3C_{23})$$

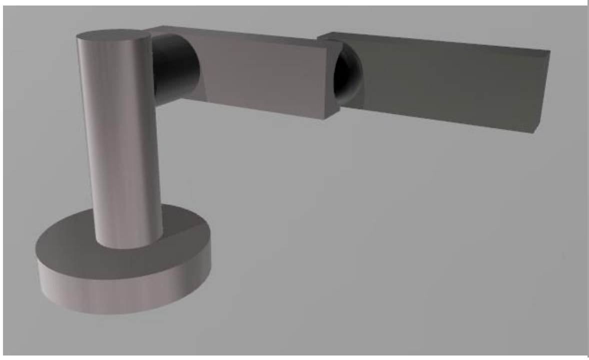
Analizando el resultado obtenido podemos saber que el codo del manipulador estará en una configuración singular cuando:

$$S_3 = 0 \rightarrow Es\ decir, cuando\ \theta_3 = 0, \pi$$

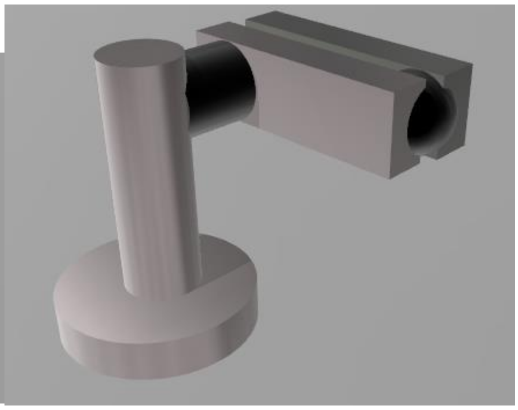
Es cuando en nuestro manipulador la variable S_3tome el valor 0 nos encontraremos ante una configuración singular, y esto ocurre cuando toma los valores $\theta_3\ de\ 0, \pi$.

Y además cuando:

$$a_2C_2 + a_3C_{23} = 0$$



Manipulador con $\theta_3 = 0$



Manipulador con $\theta_3 = \pi$

Las situaciones explicadas anteriormente surgen cuando el codo está completamente extendido o retraído como se muestra. La segunda situación de la ecuación cuando $a_2C_2 + a_3C_{23} = 0$. Esta configuración ocurre:

Singularidad del codo del manipulador sin offset.

El centro de la muñeca cruza el eje de la rotación de la base Z_0, es decir, cuando el centro de la muñeca está a lo largo de este eje. Para un manipulador de codo con un offset, como se muestra en la siguiente figura, el centro de la muñeca no puede intersecar con Z_0, lo que corrobora nuestra afirmación anterior de que los puntos alcanzables a configuraciones singulares pueden no ser accesibles bajo perturbaciones arbitrariamente pequeñas de los parámetros del manipulador, en este caso ya sea en el codo o en el hombro.

Manipulador con un offset en el codo para evitar la singularidad.

Y efectivamente muchos de los casos de uso probados en los apartados anteriores coincidían con una singularidad del manipulador. Siguiendo las ecuaciones anteriormente calculadas:

$$S_3 = 0 \rightarrow Es\ decir, cuando\ \theta_3 = 0, \pi$$

$$a_2 C_2 + a_3 C_{23} = 0$$

Podemos saber que casos de los apartados M2 y M4 son singularidades:

- Para M2 cuyos casos de estudio son:

  $$\vec{q} = (0{,}0{,}0), \vec{q} = (0, \pi/2{,}0), \vec{q} = (-\pi/2, \pi/2{,}0) \text{ y } \vec{q}(\pi, 0, \pi/2$$

  Se correspondes con configuraciones singulares los casos 2.1, 2.2, y 2.3 los cuales cumplen la primera ecuación en la que el $S_3 = 0$. Y el caso de uso 2.4 no se correspondería con una configuración singular.

- Para M4 es singularidad el caso 4.4.

  $\vec{q}$=(0,0,π/2) y $\vec{q}$=(0,0,π/90); $\vec{q}$=(0,π/4,−π/4) y $\vec{q}$=(0,−π/90,π/90); $\vec{q}$=(π /2,0,−π/2) y $\vec{q}$=(0,π/90,0); y $\vec{q}$=(0,π/2,0) y $\vec{q}$=(0,0,−π/90)

  Se corresponden con configuraciones singulares los casos de estudio 4.4 que cumple la primera ecuación en la que $S_3 = 0$.

M5.2. Generación de trayectorias

Se ha programado, utilizando las funciones jacobiana y cinematica_directa, de los apartados anteriores una nueva función que devuelve una lista de coordenadas que aplicadas en secuencia hacen que la *end-effector* describa una determinada trayectoria descrita por una función o composición de funciones matemáticas.

En las siguientes tres figuras se pueden ver capturas del objeto programado siguiendo distintos tipos de trayectorias.

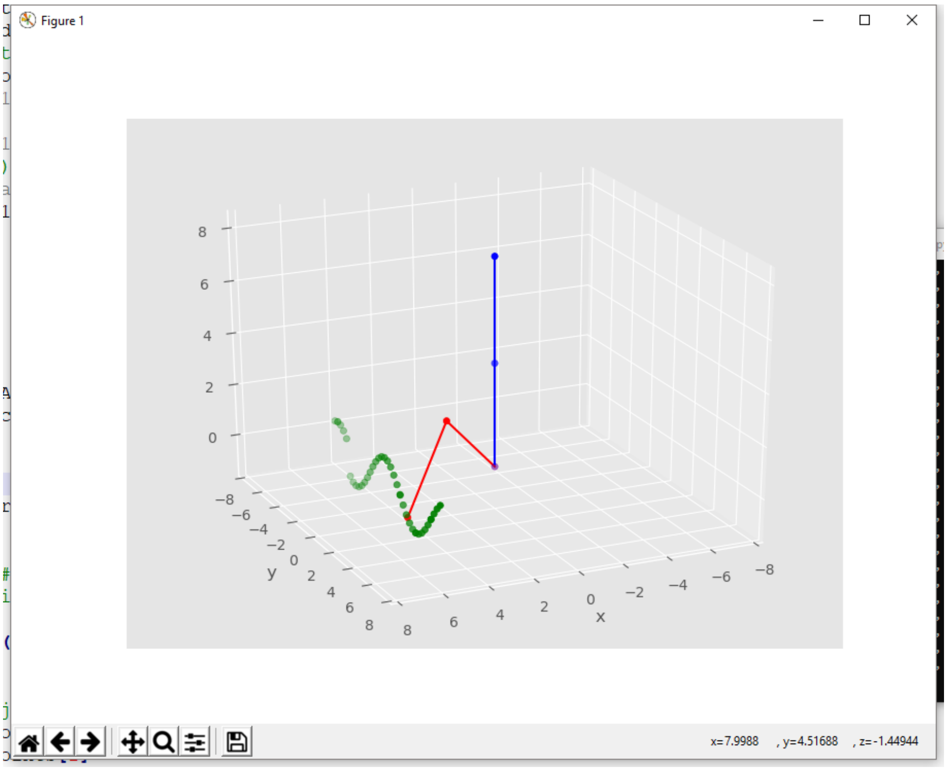


Figura M5.2.1 El brazo describiendo la trayectoria de z=cos(y)

Se ha subido un vídeo del movimiento a https://www.youtube.com/watch?v=6UaRuxUMwwQ

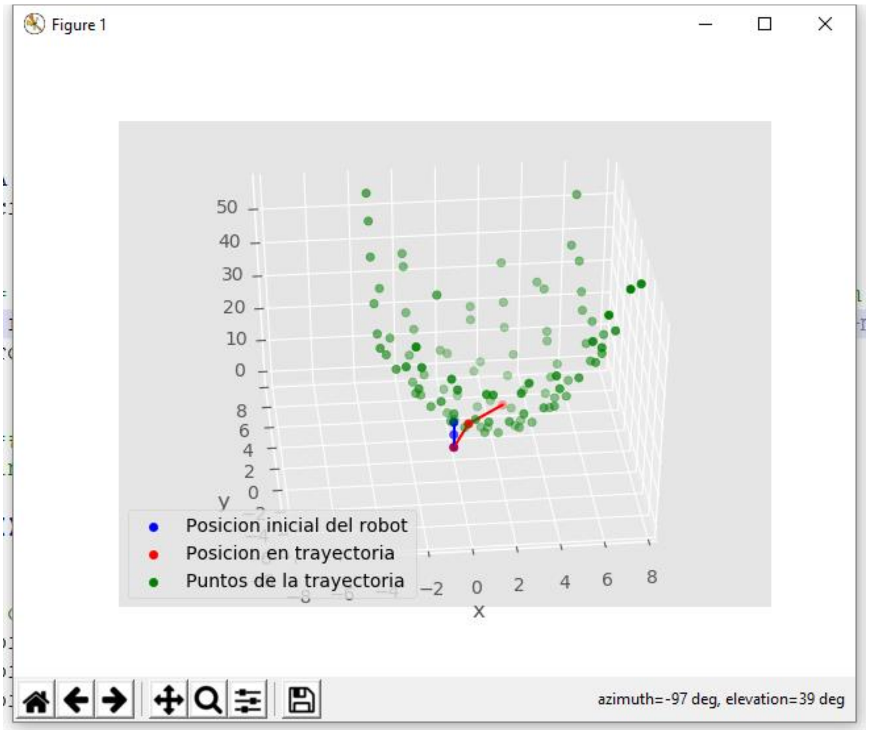


Figura M5.2.2 El brazo describiendo la trayectoria de z=x^2+y^2

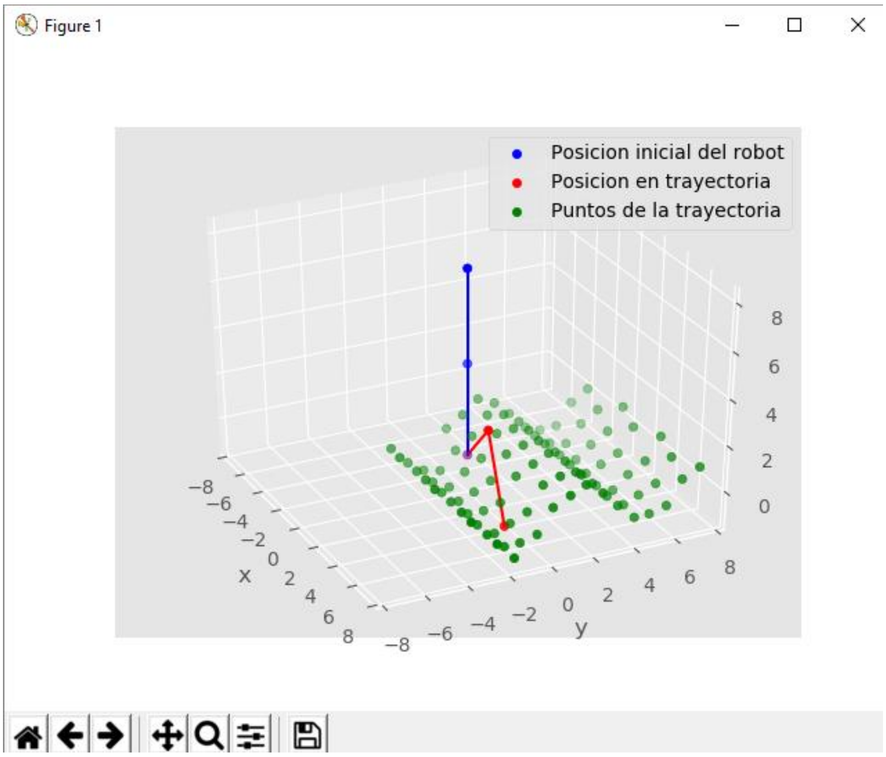


Figura M5.2.3 El brazo describiendo la trayectoria de z=sen(y)+x

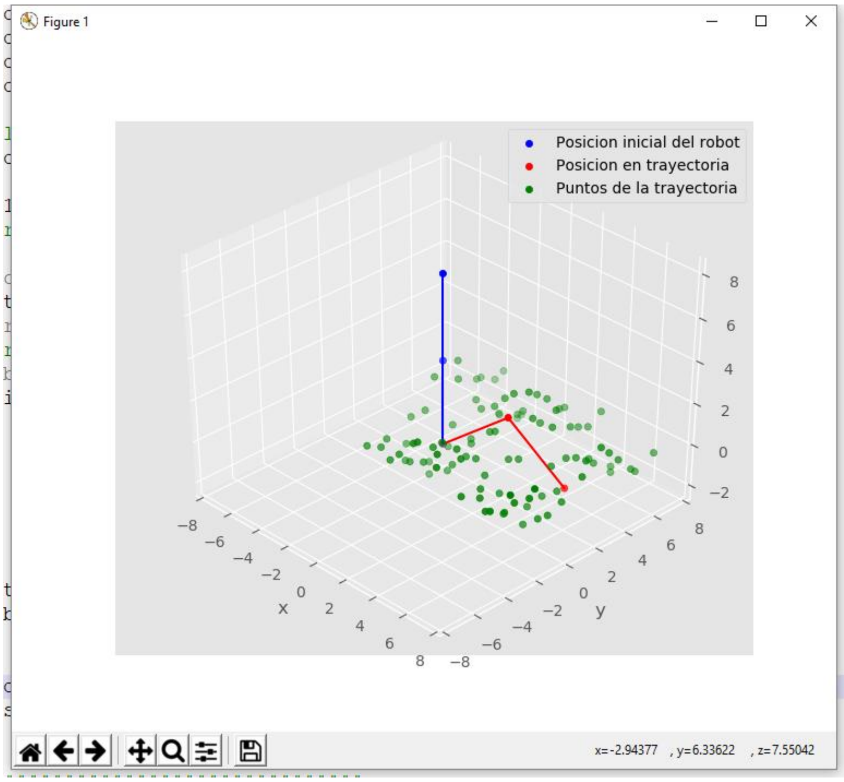


Figura M5.2.4 El brazo describiendo la trayectoria de z=sen(y)+cos(x)

El código utiliza la inversa, pseudoinversa y la transpuesta de la jacobiana, en función de las singularidades que se alcanzan, para describir el vector de velocidad lineal.

Se trata de una función que recibe una secuencia de puntos generada por otra función *generar_puntos_para_funcion*. Esta recibe una función o composición de funciones como string y devuelve una lista de puntos de esa función, generados con un paso establecido.

La función generar_trayectoria_hacia recibe una lista de puntos y devuelve la trayectoria que el robot ha de seguir para alcanzar uno a uno secuencialmente. Se devuelven los puntos de cada referencial para poder graficar el movimiento.

Dada las dimensiones de la función no se copia y pega aquí sino que se remite a página 27 del presente documento donde se encuentra debidamente comentada.

Apéndice 1. Código completo del manipulador con representación gráfica del movimiento

```
from matplotlib import pyplot as plt
from mpl_toolkits.mplot3d import axes3d
from matplotlib import style
import numpy as np
import math as m

class RobotPUMA():

      #El constructor recibe tres vectores correspondientes con las columnas de la tabla de DH
      def __init__(self, d, a, alfa, theta):

            if len(d) != len(a) != len(alfa) != len(theta):
                  print("Parametros DH no estan bien establecidos")
                  exit()

            self.d = d;
            self.a = a;
            self.alfa = alfa;
            self.theta = theta;

      #############################################
      ###               Cinematica directa                 ###
      #############################################

      def cinematica_directa(self, theta):
            self.theta = theta;

            #Matriz A de cada eje respecto al anterior
            A10 = self.matrizA(self.theta[0] , self.d[0], self.a[0], self.alfa[0])
            A21 = self.matrizA(self.theta[1], self.d[1], self.a[1], self.alfa[1])
            A32 = self.matrizA(self.theta[2], self.d[2], self.a[2], self.alfa[2])
```

```
        #Multiplicamos respecto a la trama adjunta a la base
        A20 = np.dot(A10, A21)
        A30 = np.dot(A20, A32)

        #Devolvemos las coordenadas de cada trama adjunta a cada articulacion
        #para poder graficar el brazo entero
        return np.transpose([A10[0:3, 3], A20[0:3, 3], A30[0:3, 3]])

    def matrizA(self, theta_i, d_i, a_i, alfa_i):
        return np.array([[m.cos(theta_i),  -m.sin(theta_i) * m.cos(alfa_i),   m.sin(theta_i) * m.sin(alfa_i),
    a_i * m.cos(theta_i)], \
                        [m.sin(theta_i),            m.cos(theta_i) * m.cos(alfa_i),     -
m.sin(alfa_i)*m.cos(theta_i),            a_i * m.sin(theta_i)], \
                        [0.0,                                 m.sin(alfa_i),
    m.cos(alfa_i),                                  d_i], \
                        [0.0,                                 0.0,
    0.0,                                            1.0]])


    ############################################
    ###              Cinematica inversa              ###
    ############################################

    def cinematica_inversa(self, posicion):

        theta = [0 for i in range(3)]
        px = round(posicion[0])
        py = round(posicion[1])
        pz = round(posicion[2])

        #Conocemos analiticamente los valores que producen una singularidad


        #Theta1
        if(px != 0):
            theta[0] = m.atan2(py,px)
```

```
            else:
                theta[0] = "Infinitas soluciones para theta1"
            #Primero calculamos theta3, ya que theta2 se resuelve en funcion de esta
            cos_theta3 = (m.pow(px,2) + m.pow(py,2) + m.pow(pz,2) - m.pow(a[1],2) - m.pow(a[2],2))/(2*a[1]*a[2])
            cos_theta3_cuadrado = m.pow(cos_theta3, 2)

            #Posible situacion en que el coseno calculado geometricamene con la posicion del extremo
            #sea mayor que uno (no se puede alcanzar el punto por la logitud del brazo)
            #o el coseno, que se encuentra en el denominador de theta3
            #es cero
            if(abs(cos_theta3)>1 or 0==cos_theta3):
                theta[2] = "Singularidad theta3"
            else:
                theta[2] = m.atan2(m.sqrt(1-cos_theta3_cuadrado), cos_theta3)

            #Finalmente calculamos theta2

            sen_theta3 = m.sqrt(1-m.pow(cos_theta3,2))

            beta = m.atan2(m.sqrt(m.pow(px,2)+m.pow(px,2)), pz)
            #gamma = m.atan2(a[1] + a[2]*m.cos(theta[2]), a[2]*m.sin(theta[2]))
            gamma = m.atan2(a[1] + a[2]*cos_theta3, a[2]*sen_theta3)
            theta[1] = gamma - beta

            return theta

      #############################################
      ###                     Jacobiana                      ###
      #############################################

      #Escribimos la expresion explicita alcanzada derivando
      def jacobiana(self, theta):
            return np.array([[-a[2]*m.sin(theta[0])*m.cos(theta[1]+theta[2]) -
a[1]*m.sin(theta[0])*m.cos(theta[1]),    -a[1]*m.cos(theta[0])*m.sin(theta[1]) -
a[2]*m.cos(theta[0])*m.sin(theta[1]+theta[2]), -a[2]*m.cos(theta[0])*m.sin(theta[1]+theta[2])],\
```

```
                        [a[1]*m.cos(theta[0])*m.sin(theta[1])+ a[2]*m.cos(theta[0])*m.cos(theta[1]+theta[2]),
a[1]*m.sin(theta[0])*m.cos(theta[1])  +a[2]*m.sin(theta[0])*m.sin(theta[1]+theta[2]),    -
a[2]*m.sin(theta[0])*m.sin(theta[1]+theta[2])],\
                        [0,   a[1]*m.cos(theta[1])+a[2]*m.cos(theta[1]+theta[2]),  a[2]*m.cos(theta[1]+theta[2])],\
                        [0,   0, 0],\
                        [1, 1, 1]])

      def jacobiana_por(self, theta, q):
            return np.dot(self.jacobiana(theta),q)

      #Esta funcion recibe una lista de puntos y devuelve la trayectoria que el robot
      #ha de seguir para alcanzar uno detras de otro
      def trayectoria_jacobiana_hacia(self, puntos_funcion):
            #Devolveremos la lista de las tres tramas de cada articulacion para poder dibujar el brazo
            lista_articulares = list()

            for i in range(len(puntos_funcion)):

                  punto = puntos_funcion[i]
                  print("Calculando punto", punto)

                  #Obtenemos las posiciones articulares para ese punto usando la c.directa
                  posiciones_articulares = self.cinematica_directa(self.theta)
                  lista_articulares.append(posiciones_articulares)
                  punto_actual = [row[2] for row in posiciones_articulares]

                  #Queremos alcanzar ese punto, calculamos lo cerca que estamos de el
                  error = np.subtract(punto, punto_actual)

                  #Si los puntos estan muy pegados seguimos
                  if(error[0] < 0.05 and error[1] < 0.05 and error[2] < 0.05):
                        continue

                  #Establecemos el factor de velocidad
                  incremento_posicion = 0.1
                  qd = [0 for x in range(3)]
```

```
            #Establecemos un limite de iteraciones por si se produce el caso de que un punto no es alcanzable
(nunca se cumpliria la condicion del while ya que el error seria grande siempre)
            iteraciones = 0
            lim_iteraciones = 1e3

            while (error[0] > 0.05 or error[1] > 0.05 or error[2] > 0.05)  :

                if(iteraciones >= lim_iteraciones):
                    print("     Punto no alcanzable ", punto)
                    break

                #Vector de velocidad lineal (modulo, direccion y sentido de la velocidad del manipulador)
                v = ( error / len(error) ) * incremento_posicion

                #Obtenemos la jacobiana
                J = self.jacobiana(self.theta)
                J = J[0:3][0:3]

                #Si el determinante es cero calculamos la jacobiana
                if abs(np.linalg.det(J)) < 0.0005:
                    #Puede ser que el determinante del producto tambien sea cero, en ese caso usamos la
traspuesta para calcular la trayectoria
                    JJi = np.dot(J,np.transpose(J))
                    if(abs(np.linalg.det(JJi)) < 0.0005):
                        Ji = np.transpose(J)
                    else:
                        Ji = np.dot(np.transpose(J),np.linalg.inv(JJi))
                else:
                    Ji = np.linalg.inv(J)

                qd = np.dot(Ji, v)

                # Incrementamos las coordenadas articulares
                self.theta[0] += qd[0];
                self.theta[1] += qd[1];
```

```
                        self.theta[2] += qd[2];

                        #Calamos el error respecto al nuevo punto
                        posiciones_articulares = self.cinematica_directa(self.theta);
                        punto_actual = [row[2] for row in posiciones_articulares]
                        lista_articulares.append(posiciones_articulares)
                        error = np.subtract(punto, punto_actual)
                        iteraciones += 1


            return lista_articulares

    #Funcion que recibe una funcion o composicion de funciones como string y devuelve una lista
    #de puntos de esa funcion, generados con con un paso
    def generar_puntos_para_funcion(self, cantidad, paso, expresion, xi, yi):
            hayX, hayY = "x" in expresion, "y" in expresion
            if(not hayX and not hayY):
                    print ("La expresion debe contener x o y")
                    return []

            puntos = []
            #Usamos de limite la hipotenusa de la longitud del brazo
            hipotenusa = m.sqrt(m.pow(self.a[1],2)+m.pow(self.a[2],2))
            x, y = xi, yi
            z = eval(expresion)

            for i in range(cantidad):

                    if(z > hipotenusa):
                            z = z % hipotenusa
                    if(hayX and hayY):
                            if(i < cantidad/2):
                                    x += paso
                                    y = y
                            elif(i >= cantidad/2):
                                    x = x
```

```
                        y += paso
                elif(hayX):
                    x += paso
                elif(hayY):
                    y += paso

                if(x > hipotenusa):
                    x = x % (-hipotenusa)
                    if(hayX and hayY):
                        y += paso
                if(y > hipotenusa):
                    y = y % (-hipotenusa)
                    if(hayX and hayY):
                        x += paso

                z = eval(expresion)
                puntos.append([x+xi, y+yi, z])


            return puntos



#############################################
###                     EJECUCION                     ###
#############################################

d = [0,0,0]
a = [0, 4, 4]
alfa = [m.pi/2, 0, 0]
theta = [0, m.pi/2, 0 ]

#############################################
###         EJEMPLOS PROPORCIONADOS                ###
#############################################
```

```
theta1 = [0, 0, 0 ]
theta2 = [0, m.pi/2, 0 ]
theta3 = [-m.pi/2, m.pi/2, 0 ]
theta4 = [m.pi, 0, m.pi/2 ]

theta1_jaco = [0, 0, m.pi/2]
theta2_jaco = [0, m.pi/4, -m.pi/4]
theta3_jaco = [m.pi/2, 0, -m.pi/2]
theta4_jaco = [0, m.pi/2, 0]
q1 = [0, 0, m.pi/90]
q2 = [0, -m.pi/90, m.pi/90]
q3 = [0, m.pi/90, 0]
q4 = [0, 0, -m.pi/90]

robot_ejemplos = RobotPUMA(d, a, alfa, theta1)

#Imprimimos los ejemplos

#EJEMPLO 1
directa1 = robot_ejemplos.cinematica_directa(theta1)
directa1_t = np.transpose(directa1)
end_effector1 = directa1_t[2]
inversa1 = robot_ejemplos.cinematica_inversa(end_effector1)

#EJEMPLO 2
directa2 = robot_ejemplos.cinematica_directa(theta2)
directa2_t = np.transpose(directa2)
end_effector2 = directa2_t[2]
inversa2 = robot_ejemplos.cinematica_inversa(end_effector2)

#EJEMPLO 3
directa3 = robot_ejemplos.cinematica_directa(theta3)
directa3_t = np.transpose(directa3)
end_effector3 = directa3_t[2]
inversa3 = robot_ejemplos.cinematica_inversa(end_effector3)
```

```
#EJEMPLO 4
directa4 = robot_ejemplos.cinematica_directa(theta4)
directa4_t = np.transpose(directa4)
end_effector4 = directa4_t[2]
inversa4 = robot_ejemplos.cinematica_inversa(end_effector4)

#Jacobianas
jacobiana1 = robot_ejemplos.jacobiana_por(theta1_jaco, q1)
jacobiana2 = robot_ejemplos.jacobiana_por(theta2_jaco, q2)
jacobiana3 = robot_ejemplos.jacobiana_por(theta3_jaco, q3)
jacobiana4 = robot_ejemplos.jacobiana_por(theta4_jaco, q4)

print("##############################################")
print("###        EJEMPLOS PROPORCIONADOS               ###")
print("##############################################")

print("Theta1:", theta1)
print("Theta2:", theta2)
print("Theta3:", theta3)
print("Theta4:", theta4)
print("")
print ("Directa 1:")
print (np.round(directa1, 2))
print ("Directa 2:")
print (np.round(directa2, 2))
print ("Directa 3:")
print (np.round(directa3, 2))
print ("Directa 4:")
print (np.round(directa4, 2))
print("")
print ("Inversa 1:")
print (inversa1)
print ("Inversa 2:")
print (inversa2)
print ("Inversa 3:")
print (inversa3)
```

```
print ("Inversa 4:")
print (inversa4)
print("")
print ("Jacobiana 1:")
print (np.round(jacobiana1,6))
print ("Jacobiana 2:")
print (np.round(jacobiana2,6))
print ("Jacobiana 3:")
print (np.round(jacobiana3,6))
print ("Jacobiana 4:")
print (np.round(jacobiana4,6))
print("")

#############################################
###         GRAFICANDO EL MANIPULADOR            ###
#############################################

robot = RobotPUMA(d, a, alfa, theta)
p_articulacion= robot.cinematica_directa(theta)

#Distintos tipos de funciones
#puntos_funcion = robot.generar_puntos_para_funcion(100, m.pi/2, "m.sin(y)+m.cos(x)", 0, 2)
#puntos_funcion = robot.generar_puntos_para_funcion(100, m.pi/2, "m.pow(y,2)+m.pow(x,2)", 0, 2)
puntos_funcion = robot.generar_puntos_para_funcion(100, m.pi/2, "m.cos(y)", 2, 0)
#Obtenemos la secuencia de posiciones de las coordenadas articulares que vamos a graficar
sec_coord_articulares = robot.trayectoria_jacobiana_hacia(puntos_funcion)

plt.close('all')
fig = plt.figure()

#Fila de la matriz de las posiciones x,y,z de las articulaciones de la posicion inicial del brazo
articulacion_x_o = p_articulacion[0]
articulacion_y_o = p_articulacion[1]
articulacion_z_o = p_articulacion[2]

style.use('ggplot')
```

```
ax1 = fig.add_subplot(111, projection='3d')

#Graficamos los brazos
puntos_funcion = np.transpose(puntos_funcion)
for i in range(len(sec_coord_articulares)):

      #Nueva posicion para graficar
      x_IK = sec_coord_articulares[i][0]
      y_IK = sec_coord_articulares[i][1]
      z_IK = sec_coord_articulares[i][2]

      #Establecemos los limites del grafico
      ax1.set_xlim([-(a[1]+a[2]),a[1]+a[2]])
      ax1.set_ylim([-(a[1]+a[2]),a[1]+a[2]])

      # Graficamos la posicion inicial
      ax1.plot(articulacion_x_o,articulacion_y_o,articulacion_z_o,color='blue')
      start_joints=ax1.scatter(articulacion_x_o,articulacion_y_o,articulacion_z_o,label='start',color='blue')

      #Graficamos los puntos de la trayectoria
      puntos_trayectoria = ax1.scatter(puntos_funcion[0],puntos_funcion[1],puntos_funcion[2],color='green')

      #Graficamos la
      nueva_posicion=ax1.scatter(x_IK,y_IK,z_IK,color='red')
      ax1.plot(x_IK,y_IK,z_IK,label='IK position',color='red')

      #Colcamos las leyendas
      plt.legend([start_joints,nueva_posicion, puntos_trayectoria], ['Posicion inicial del robot','Posicion en 
trayectoria', 'Puntos de la trayectoria'])
      plt.xlabel("x")
      plt.ylabel("y")
      plt.pause(0.001)
      #Para que no se borre el tapiz al llegar la ultima posicion
      if i != len(sec_coord_articulares)-1:
            ax1.clear()
plt.show()
```