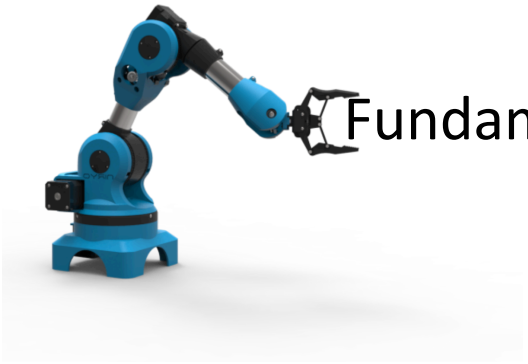
FACULTAD DE INGENIERIA

UNIVERSIDAD DE LA REPÚBLICA URUGUAY



Fundamentos de Robótica Industrial

Trayectorias

28 de Mayo del 2024

Como en cada uno de los laboratorios, éste contará con una primera parte donde se realizará un control de lectura escrito, con el fin de verificar que estudió el material facilitado por los docentes y conoce cómo se llevará a cabo el laboratorio.

Laboratorio 3

Objetivo

Este laboratorio tiene por objetivo resolver una trayectoria circular en el plano xy que será recorrida por el puntero del manipulador.

Fundamentos y Metodología

Antes de la realización del laboratorio es indispensable haber leído y tener presente la guía de “Buenas prácticas para trabajo de laboratorio”, disponible en el EVA del curso.

Materiales a utilizar

Se utilizarán:

- Manipulador de 4 grados de libertad (figura 1)
- USB 2D (figura 2)
- Fuente de alimentación 12V
- Computadora
- Soporte de marcador (figura 3)
- Marcador

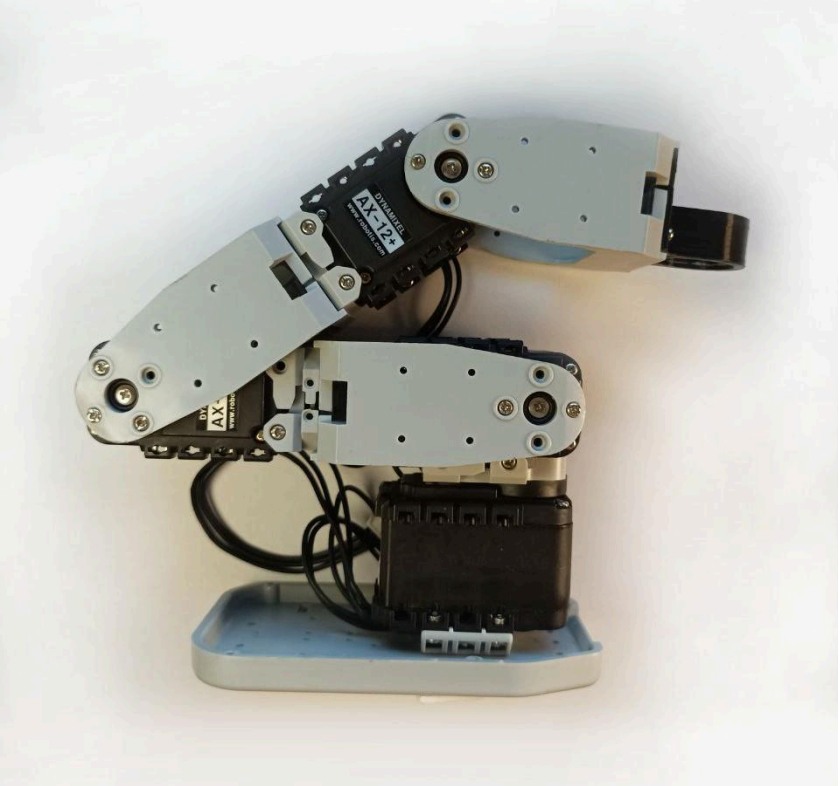


Figura 1: Referencia del manipulador a utilizar

Figura 2: Módulo USB U2D2 de Robotis

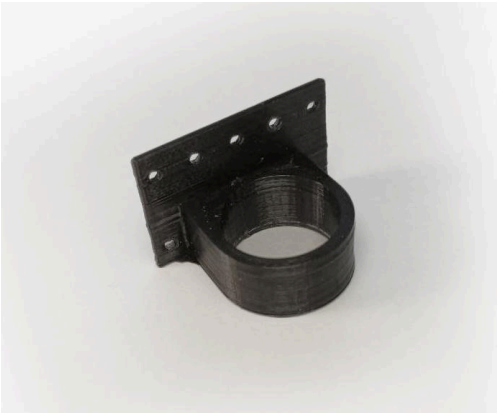


Figura 3: Soporte de marcador

Trayectorias

Teniendo en cuenta la resolución de la cinemática inversa para un punto cualquiera del espacio, resolver una trayectoria determinada para el manipulador no es más que resolver la cinemática inversa, uno a uno, de los puntos de la misma. Dado que en una trayectoria pueden encontrarse infinitos puntos, es necesario discretizar la misma en una cantidad razonable para que el manipulador mantenga un movimiento fluído y que la trayectoria real recorrida por el puntero del manipulador no difiera mucho de la trayectoria teórica. Un ejemplo podría ser discretizar una circunferencia en 360 puntos y resolver la cinemática inversa para cada uno de ellos.

Para que el extremo del manipulador de un punto A a un punto B dentro del espacio de trabajo, una solución para realizar esta tarea es mover cada articulación a su velocidad máxima, por lo que algunas articulaciones pueden alcanzar su posición final antes que otras. Otra forma sería mover todas las articulaciones de forma proporcional a su recorrido para que comiencen y terminen simultáneamente.

Cada trayectoria requerirá su análisis particular, ya que al discretizar la misma en pequeños tramos de igual longitud, puede que la velocidad en algunos de dichos tramos sobrepase los límites del actuador, para ello, una alternativa puede ser suavizar el comienzo y final de la trayectoria. Esto se logra discretizando en tramos de menor longitud al inicio y final de la misma.

Para la generación de trayectorias (en el espacio articular) se utilizarán los siguientes interpoladores:

- **Interpolador de tercer orden**

  Asumiendo que se conoce la **configuración inicial y la configuración final** a la que se desea llevar la terminal del robot, se puede conocer (a partir de cinemática inversa) la posición inicial y final de cada articulación.
  Además, se deberá imponer la **velocidad en dichos puntos**, que puede ser “reposo” o una velocidad de “paso”

  Considerando una articulación cuya posición y velocidad para el instante inicial y el final son:

  $$\theta(t_i) = \theta_i \qquad \dot{\theta}(t_i) = 0$$
  $$\theta(t_f) = \theta_f \qquad \dot{\theta}(t_f) = 0$$

  Y se desea conocer (con el objetivo de controlar) el valor de la posición de esa articulación para cada instante t: $[t_i < t < t_f]$. Esto puede lograrse imponiendo una ***trayectoria polinomial de tercer orden*** entre los puntos extremos, de forma que cumplan las condiciones iniciales y finales deseadas:

  $$\theta(t) = c_0 + c_1 t + c_2 t^2 + c_3 t^3$$

  De esta forma se obtienen los valores que debe tomar cada constante del polinomio para

cada articulación, de forma de alcanzar el punto deseado mediante el movimiento polinomial de cada articulación.

- **Interpolador de quinto orden:**

  En el caso donde las ***aceleraciones requeridas pueden no ser alcanzables*** por los actuadores que se poseen, o simplemente se desea controlar además, la aceleración durante la trayectoria, limitando sus mínimos, máximos y dándole continuidad, es posible utilizar ***interpoladores de quinto orden***, donde la posición, velocidad y aceleración quedan definidas como sigue:

$$\theta(t) = c_0 + c_1 t + c_2 t^2 + c_3 t^3 + c_4 t^4 + c_5 t^5$$

$$\dot{\theta}(t) = c_1 + 2c_2 t + 3c_3 t^2 + 4c_4 t^3 + 5c_5 t^4$$

$$\ddot{\theta}(t) = 2c_2 + 6c_3 t + 12c_4 t^2 + 20c_5 t^3$$

  Donde como incógnitas se tienen **6 constantes** para cada tramo de trayecto, y por lo tanto se necesitan 6 condiciones para determinar los **coeficientes c_0 a c_5.**

  Estas por lo general son: las posiciones, velocidades y aceleraciones al comienzo y al final:

$$\theta(t_i) = \theta_i \quad \dot{\theta}(t_i) = 0 \quad \ddot{\theta}(t_i) = \ddot{\theta}_i$$
$$\theta(t_f) = \theta_f \quad \dot{\theta}(t_f) = 0 \quad \ddot{\theta}(t_f) = \ddot{\theta}_f$$

- **Interpolador trapezoidal (lineal con extremos parabólicos):**

  Una alternativa a los polinomios de orden alto (3er o 5to) es usar una ***interpolación lineal*** (polinomio de primer orden) entre el punto de salida y el de llegada, pero ***no se recomienda*** este tipo de interpolación puesto que implica cambio bruscos de velocidad en los puntos, y altas aceleraciones, provocando movimientos espasmódicos.

  Como solución a estos cambios bruscos, se propone ***transformar el polinomio lineal en una parábola*** al comienzo de la trayectoria y próximo al final, creando una curva continua de posición y velocidad. De esta forma, se definirá un t_b que es el "tiempo de combinación" (blending time), luego el movimiento será lineal hasta t_f-t_b donde comenzará nuevamente la misma parábola, pero invertida (siempre que se considere simétrica que es lo usual).

  La trayectoria queda definida por una función partida:

$$\theta(t) = \begin{cases} c_{0p} + c_{1p} t + \frac{1}{2} c_{2p} t^2 & \text{si: } t \in \text{ parabola} \\ c_{0L} + c_{1L} t & \text{si: } t \in \text{ lineal} \end{cases}$$

De t=0 a t=t_b se tiene una parábola para la cual deberemos conocer ciertas condiciones para determinarla, por ejemplo: posición inicial, velocidad inicial y **se puede imponer una aceleración (que será constante) como la velocidad de rotación ω que se desea en el tramo constante dividido entre t_b.**

$$\begin{cases} \theta(t_i) = \theta_i \\ \dot{\theta}(t_i) = 0 \\ \ddot{\theta}(t_i) = \omega/t_b \end{cases}$$

De aquí se obtiene que para la zona parabólica (t<t_b):

$$\begin{gathered} \theta(t) = \theta_i + \frac{\omega}{2t_b}t^2 \\ \dot{\theta}(t) = (\omega/tb)t \\ \ddot{\theta}(t) = \omega/t_b \end{gathered}$$

Luego para determinar la trayectoria en la zona lineal se deberá igualar la expresión de la posición y de la velocidad de ambas zonas en A, para que sean continuas:

$$\dot{\theta}_p(t_b) = \dot{\theta}_L(t_b)$$

$$\frac{\omega}{t_b}t_b = \omega = c_{1L}$$

Obteniendo:

$$\begin{gathered} \theta(t) = \theta_i - \frac{\omega}{2}t_b + \omega t \\ \dot{\theta}(t) = w \\ \ddot{\theta}(t) = 0 \end{gathered}$$

La parábola final es simétrica a la primera, pero con una aceleración negativa. Entonces la parábola final (de t=t_f-t_b a t=t_f) puede expresarse de la siguiente forma:

$$\begin{gathered} \theta(t) = \theta_f - \frac{\omega}{2t_b}(t_f - t)^2 \\ \dot{\theta}(t) = \frac{\omega}{t_b}(t_f - t) \\ \ddot{\theta}(t) = -\frac{\omega}{t_b} \end{gathered}$$

- Existen también los Interpoladores lineales pero **no se recomiendan** por generar trayectorias espasmódicas.

Código

Se utilizará el código para la cinemática inversa del manipulador de 4GDL (resuelto en el laboratorio 2) para lograr el movimiento del puntero en la trayectoria deseada. Para ello se hará uso de nuevas herramientas de Python como la definición de una función. Una función es una sección de un programa que calcula un valor de manera independiente al resto del programa, en el anexo puede verse un ejemplo de trayectoria parabólica para el cual, modificando algunos detalles, pueden obtenerse movimientos para diversas trayectorias.

Para la resolución de la función "circunferencia", se hará uso del esquema mostrado en la figura 4, donde el sistema de coordenadas será el mismo que conforma el de la base del manipulador (S_0) utilizado en el laboratorio. Dicha función utilizará las coordenadas cartesianas (x, y, z) para cada punto de la circunferencia.

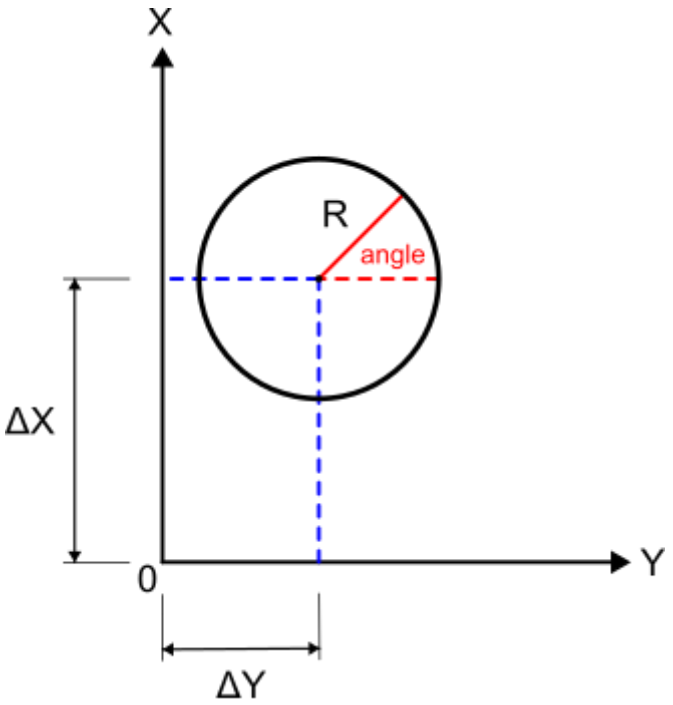


Figura 4: Esquema de circunferencia

Descripción de tareas:

1. Resolución de trayectoria circular
2. Desarrollo de dichas resoluciones en Python
3. Conexión del manipulador a la PC y verificación de su funcionamiento
4. Elección de parámetros correspondientes (ubicación en el espacio del centro de la cfa., radio, velocidad de trabajo, etc.)
5. Verificar el correcto funcionamiento del manipulador dibujando la trayectoria en la estación de trabajo con el marcador colocado en su soporte

Anexo

Ejemplo (parábola)

```
import math as m

speed = 150
step = 1

A=(1.0/50.0) #Parametros de la parabola
B=0
C=100

a1 =
a2 =
a3 =
a4 =

#Definicion de la funcion de nombre puntoPbla que recibe los parametros
recorrdoy y deltaZ
#Devuelve el array de coordenadas del punto (x,y,z)
def puntoPbla(recorridoy, deltaZ):
    y = recorridoy
    x = A*(y**2)+B*y+C
    z = deltaZ
    pto = [x, y, z]

    return pto


def cinematicaInversa(x, y, z):...#Insertar resolucion de cinematica
inversa para 4GDL


    #T en posicion del 0 al 1023
    T1 = (T1*(1023/300)+(1023/2))
```

```
    T2 = (T2*(1023/300)+(1023/2))
    T3 = (T3*(1023/300)+(1023/2))
    T4 = (T4*(1023/300)+(1023/2))

    return {1: T1, 2: T2, 3: T3, 4: T4}


array_ptos_pbla = [] #Inicializacion de un array vacio para cargarle
los datos

for punto_pbla in range(-50, 50, step):
    array_ptos_pbla.append(puntoPbla(punto_pbla, deltaZ)) #Se agrega un
valor por cada iteracion relativa al range


for i in array_ptos_pbla:
    final_pos = cinematicaInversa(i[0], i[1], i[2]) #Una vez cargados
los valores se le aplica la cinematica inversa a cada valor

    for j in range(4):
        chain.goto(j + 1, final_pos[j + 1], speed=speed,
blocking=False) #Ver apendice Lab 2
```

Código Cfa.:

```
import math as m

speed = 150
deltaX =     # Distancia en x desde el origen hasta el centro de la cfa
deltaY =     # Distancia en y desde el origen hasta el centro de la cfa
deltaZ =     # Distancia en z desde el origen hasta el centro de la cfa
step = 5      # Dimension de la discretizaccion
radio =   # Radio de la cfa
max_angulo_cfa = 360

# Pueden agregarse mas grados
```

```
a1 =
a2 =
a3 =
a4 =

# Funcion que devuelve las coordenadas de un punto
def puntoCfa(angle, radio, deltaX, deltaY, deltaZ):
    x =
    y =
    z =
    pto = [x, y, z]

    return pto


def cinematicaInversa(x, y, z):...#Insertar resolucion de cinematica
inversa para 4GDL



    # T1 en posicion del 0 al 1023 que trae el motor
    T1 = (T1*(1023/300)+(1023/2))
    T2 = (T2*(1023/300)+(1023/2))
    T3 = (T3*(1023/300)+(1023/2))
    T4 = (T4*(1023/300)+(1023/2))

    return {1: T1, 2: T2, 3: T3, 4: T4}


array_ptos_cfa = [] #Inicializacion de un array vacio para cargarle los
datos
# punto_cfa va de 0 a 360

for punto_cfa in range(0, max_angulo_cfa, step):
    array_ptos_cfa.append(puntoCfa(punto_cfa, radio, deltaX, deltaY,
deltaZ)) #Se agrega un valor por cada iteracion relativa al range

for i in array_ptos_cfa:
```

```
    final_pos = cinematicaInversa(i[0], i[1], i[2]) #Una vez cargados
los valores se le aplica la cinematica inversa a cada valor

    for j in range(4):
        chain.goto(j + 1, final_pos[j + 1], speed=speed,
blocking=False)
```