Práctica 3

Javier Gálvez Obispo

21 de mayo de 2021

Ejercicio 1

Definimos la función **trayectoria434** que calcula los coeficientes de los polinomios que definen la trayectoria 4-3-4 para una articulación dados los valores de las articulaciones en los puntos de inicio (p_I), despegue (p_D), asentamiento (p_A) y fin (p_F) junto a la duración de los segmentos t_1, t_2 y t_3 en segundos.

Utilizamos las ecuaciones dadas para el cálculo de 7 de los 14 coeficientes necesarios teniendo en cuenta, que tanto la velocidad como la aceleración son 0 en las posiciones inicial y final. Utilizamos la función **linalg.inv** de *numpy* para calcular la inversa de la matriz del sistema de ecuaciones que hay que resolver para obtener el valor de los 7 coeficientes restantes.

```
def trayectoria434(qI, qD, qA, qF, t1, t2, t3):
    c10 = qI
    c11 = 0
    c12 = 0
    c20 = qD
    c30 = qF
    c31 = 0
    c32 = 0

    matriz = np.array([[1, 1, 0, 0, 0, 0, 0],
                       [3/t1, 4/t1, -1/t2, 0, 0, 0, 0],
                       [6/t1**2, 12/t1**2, 0, -2/t2**2, 0, 0, 0],
                       [0, 0, 1, 1, 1, 0, 0],
                       [0, 0, 0, 0, 0, -1, 1],
                       [0, 0, 1/t2, 2/t2, 3/t2, -3/t3, 4/t3],
                       [0, 0, 0, 2/t2**2, 6/t2**2, 6/t3**2, -12/t3**2]
                      ])
    vector = np.array([qD-qI, 0, 0, qA-qD, qA-qF, 0, 0])
    coefs = np.dot(np.linalg.inv(matriz), vector)

    f1 = [coefs[1], coefs[0], c12, c11, c10]
    f2 = [coefs[4], coefs[3], coefs[2], c20]
    f3 = [coefs[6], coefs[5], c32, c31, c30]
    return (f1, f2, f3)
```

Ejercicio 2

Utilizamos la función **trayectoria434** definida para mostrar la trayectoria 4-3-4 seguida por un robot RR que pasa por los puntos:

- Punto de inicio → [1, 0]
- Punto de despegue → [1, 0,1]
- Punto de asentamiento → [1,5, 0,1]
- Punto de fin → [1,5, 0]

y que tarda 1 segundo en recorrer cada segmento.

Calculamos los valores que toman las articulaciones del robot en los cuatro puntos dados con la función **pci** de la práctica 2 y se los pasamos a la función del ejercicio anterior.

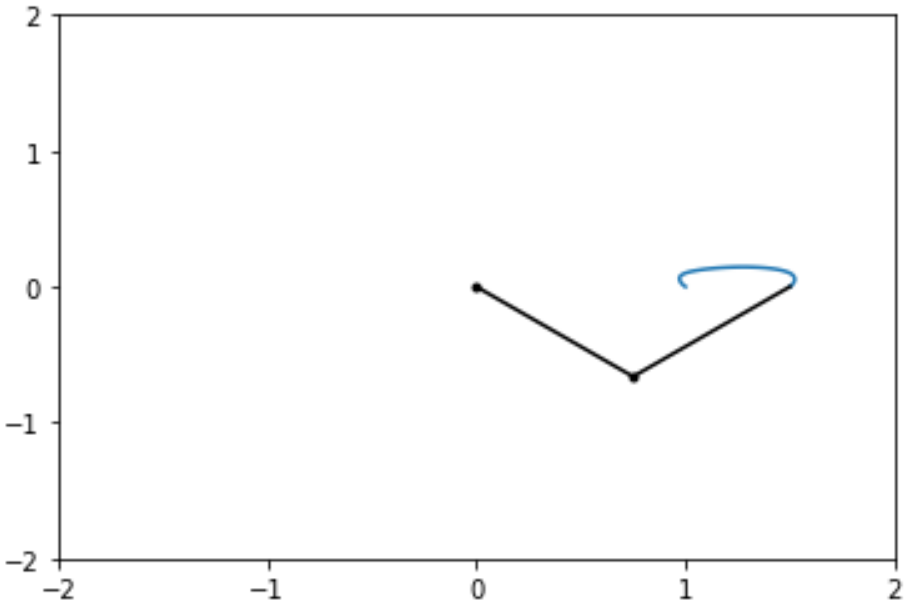
Las funciones que definen la trayectoria para la primera articulación son las siguientes:

$$
\begin{aligned}
{}^{q_1}f_1(t) &= -0,0167t^4 + 0,1192t^3 - 1,0472 \\
{}^{q_1}f_2(t) &= -0,2575t^3 + 0,2576t^2 + 0,2910t - 0,9446 \\
{}^{q_1}f_3(t) &= -0,2408t^4 - 0,3096t^3 - 0,7227
\end{aligned}
$$

Y para la segunda articulación se tiene:

$$
\begin{aligned}
{}^{q_2}f_1(t) &= -0,3153t^4 + 0,3095t^3 + 2,0944 \\
{}^{q_2}f_2(t) &= 0,6475t^3 - 0,9631t^2 - 0,3326t + 2,0886 \\
{}^{q_2}f_3(t) &= 0,3315t^4 + 0,3365t^3 + 1,4455
\end{aligned}
$$

Entonces, la trayectoria seguida por el robot RR utilizando un periodo de muestreo de 0,05 segundos es la siguiente:



Vemos que la trayectoria no es la esperable. Lo lógico sería que el robot fuese en línea recta a los distintos puntos, pero vemos que no es el caso. Esto se debe al método utilizado para el cálculo de la trayectoria, que resulta en trayectorias menos lógicas para el usuario, pero que cumplen con todas las condiciones que se imponen.

Para comprobrar que el robot pasa por todos los puntos indicados en la trayectoria utilizamos la función **pcd** de la práctica 2 y obtenemos los siguientes resultados:

- Posición inicial $\rightarrow$ (0,9999999999999998, $2,220446049250313 * 10^{-16}$)
- Posición al principio del segundo segmento $\rightarrow$ (1,0, 0,10000000000000009)
- Posición al principio del tercer segmento $\rightarrow$ (1,5, 0,10000000000000009)
- Posición final $\rightarrow$ (1,5, 0,0)

y vemos que efectivamente, despreciando una pequeña variación en algunos casos, el robot pasa por los 4 puntos.

Ejercicio 3

Calculamos la velocidad articular derivando las funciones anteriores que modelan los valores de la variable de articulación. Para ello, hacemos uso

de la función **polyder** que proporciona *numpy*. Para la primera articulación obtenemos:

$$
\begin{aligned}
{}^{q_1}f_1'(t) &= -0,0668t^3 + 0,3577t^2 \\
{}^{q_1}f_2'(t) &= -0,7726t^2 + 0,5152t + 0,2910 \\
{}^{q_1}f_3'(t) &= -0,9631t^3 - 0,9296t^2
\end{aligned}
$$

Para comprobar que la velocidad es continua para la articulación q_1, como hemos impuesto al definir los polinomios de la trayectoria, evaluamos las funciones de la velocidad en los extremos de los segmentos:

$$
\begin{aligned}
{}^{q_1}f_1'(1) &= -0,0668 + 0,3577 = 0,2910 = {}^{q_1}f_2'(0) \\
{}^{q_1}f_2'(1) &= -0,7726 + 0,5152 + 0,2910 = 0,0335 \\
{}^{q_1}f_3'(-1) &= 0,9631 - 0,9296 = 0,0335 = {}^{q_1}f_2'(1)
\end{aligned}
$$

Derivados las funciones para la segunda articulación:

$$
\begin{aligned}
{}^{q_2}f_1'(t) &= -1,2611^3 + 0,9285t^2 \\
{}^{q_2}f_2'(t) &= 1,9424^2 - 1,9262t - 0,3326 \\
{}^{q_2}f_3'(t) &= 1,3260t^3 + 1,0096t^2
\end{aligned}
$$

Es fácil ver que la velocidad para la segunda articulación también es continua.

Volvemos a derivar para obtener la aceleración articular y para la primera articulación obtenemos:

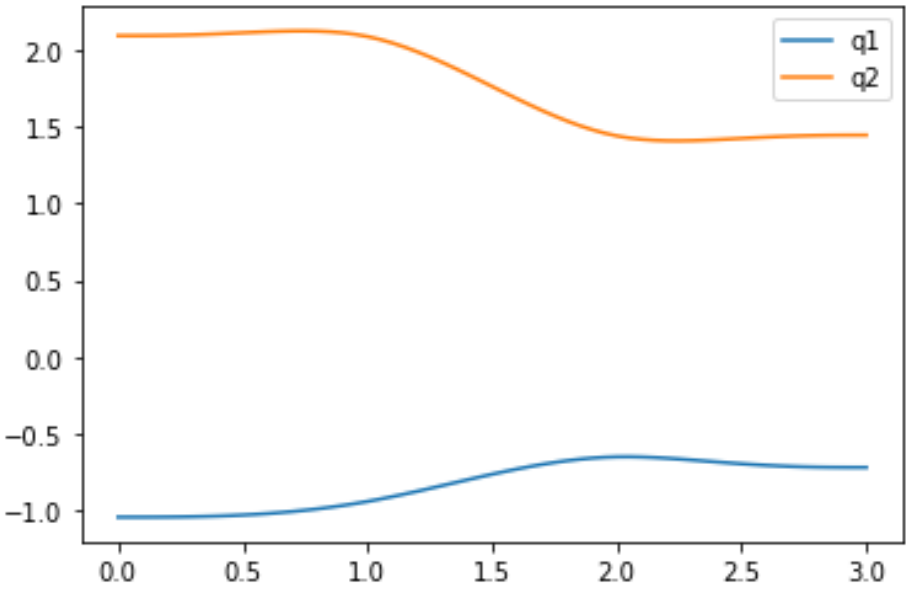
$$
\begin{aligned}
{}^{q_1}f_1''(t) &= -0,2003t^2 + 0,7154t \\
{}^{q_1}f_2''(t) &= -1,5452t + 0,5152 \\
{}^{q_1}f_3''(t) &= -2,8892t^2 - 1,8591t
\end{aligned}
$$

y para la segunda articulación:

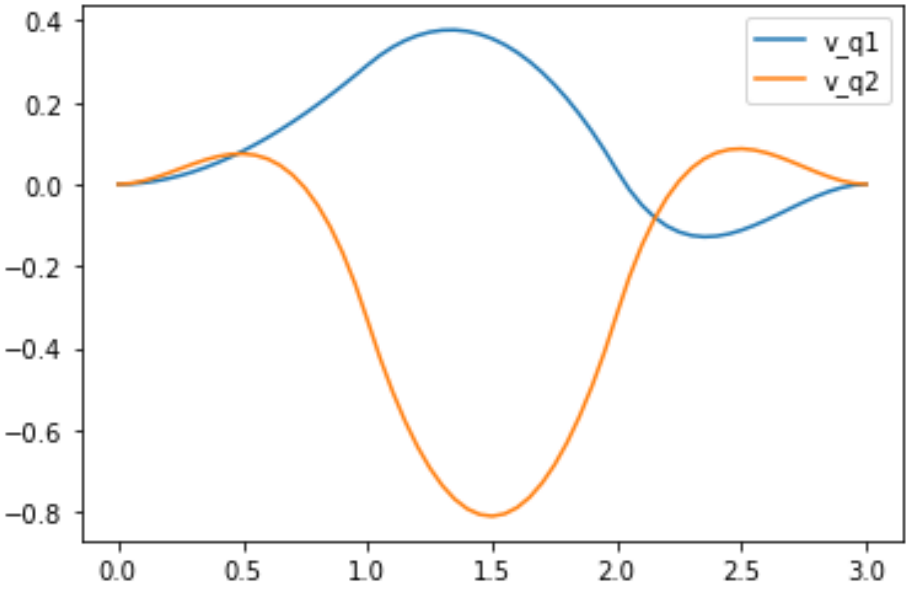
$$
\begin{aligned}
{}^{q_2}f_1''(t) &= -3,7833t^2 + 1,8571t \\
{}^{q_2}f_2''(t) &= 3,8849t - 1,9262 \\
{}^{q_2}f_3''(t) &= 3,9778t^2 + 2,0191t
\end{aligned}
$$

Al igual que antes, es muy sencillo comprobar que la aceleración es continua para ambas articulaciones.

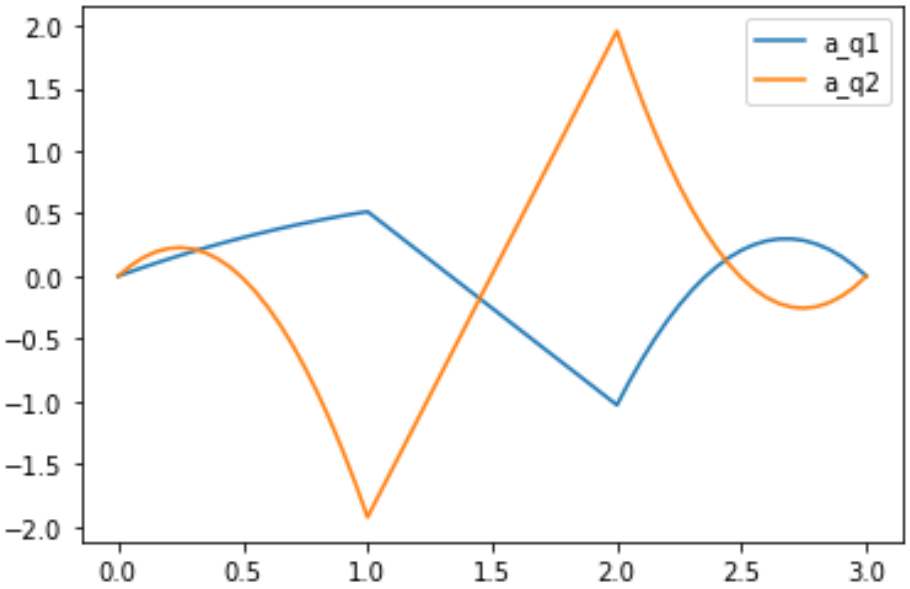
Mostramos ahora el valor de las articulaciones q_1 y q_2 a lo largo del tiempo:



hacemos lo mismo para la velocidad de ambas articulaciones:



y para la aceleración:



Con la gráficas también podemos ver que tanto la velocidad como la aceleración son continuas para ambas articulaciones.

Archivos de código

practica3.py

```
# —*— coding: utf—8 —*—
# Javier Gálvez Obispo

import numpy as np
import matplotlib.pyplot as plt

from cinematico_inverso import pci
from cinematico_directo import animacion_trayectoria_pcd, pcd

def trayectoria434(qI, qD, qA, qF, t1, t2, t3):
    c10 = qI
    c11 = 0
    c12 = 0
    c20 = qD
    c30 = qF
    c31 = 0
    c32 = 0

    matriz = np.array([[1, 1, 0, 0, 0, 0, 0],
                       [3/t1, 4/t1, —1/t2, 0, 0, 0, 0],
                       [6/t1**2, 12/t1**2, 0, —2/t2**2, 0, 0, 0],
                       [0, 0, 1, 1, 1, 0, 0],
                       [0, 0, 0, 0, 0, —1, 1],
                       [0, 0, 1/t2, 2/t2, 3/t2, —3/t3, 4/t3],
                       [0, 0, 0, 2/t2**2, 6/t2**2, 6/t3**2, —12/t3**2]
                      ])
    vector = np.array([qD—qI, 0, 0, qA—qD, qA—qF, 0, 0])
    coefs = np.dot(np.linalg.inv(matriz), vector)

    f1 = [coefs[1], coefs[0], c12, c11, c10]
    f2 = [coefs[4], coefs[3], coefs[2], c20]
    f3 = [coefs[6], coefs[5], c32, c31, c30]
    return (f1, f2, f3)

def evaluar_polinomio(coefs, x):
    f = 0
    l = len(coefs)—1
    for i, c in enumerate(coefs):
```

```
        f += c * x**(l-i)
    return f

def evaluar_trayectoria(tiempos, periodo, f1s, f2s):
    q1s, q2s = [], []


    for k in range(len(f1s)):
        pasos = int(tiempos[k] / periodo)
        for i in range(pasos):
            # Cambio de variable para normalizar el tiempo
            t_hasta_seg = sum(tiempos[:k])
            tau = i*periodo + t_hasta_seg
            t = (tau - t_hasta_seg) / tiempos[k]

            # Segundo cambio de variable para el último segmento
            if k == len(trayectoria_q1)-1:
                t = t - 1

            q1s.append(evaluar_polinomio(f1s[k], t))
            q2s.append(evaluar_polinomio(f2s[k], t))

    # Último punto
    q1s.append(evaluar_polinomio(f1s[2], 0))
    q2s.append(evaluar_polinomio(f2s[2], 0))

    return q1s, q2s


l1 = l2 = 1
qI = pci(1, 0, l1, l2)
qD = pci(1, 0.1, l1, l2)
qA = pci(1.5, 0.1, l1, l2)
qF = pci(1.5, 0, l1, l2)

ti = [1, 1, 1]
trayectoria_q1 = trayectoria434(qI[0], qD[0], qA[0], qF[0], *ti)
trayectoria_q2 = trayectoria434(qI[1], qD[1], qA[1], qF[1], *ti)

periodo = 0.05
```

```
q1s, q2s = evaluar_trayectoria(ti, periodo,
                               trayectoria_q1, trayectoria_q2)
animacion_trayectoria_pcd(q1s, q2s, l1, l2)

# Comprobación que pasa por por los puntos dados
print("Posición inicial:", pcd(q1s[0], q2s[0], l1, l2))
print("Posición al principio del segundo segmento:",
      pcd(q1s[20], q2s[20], l1, l2))
print("Posición al principio del tercer segmento:",
      pcd(q1s[40], q2s[40], l1, l2))
print("Posición final:", pcd(q1s[-1], q2s[-1], l1, l2))

# Cálculo de la velocidad
v_q1 = [np.polyder(f) for f in trayectoria_q1]
v_q2 = [np.polyder(f) for f in trayectoria_q2]
v_q1s, v_q2s = evaluar_trayectoria(ti, periodo, v_q1, v_q2)

# Cálculo de la aceleración
a_q1 = [np.polyder(f) for f in v_q1]
a_q2 = [np.polyder(f) for f in v_q2]
a_q1s, a_q2s = evaluar_trayectoria(ti, periodo, a_q1, a_q2)

# Pintar gráficas
tiempo = [i*0.05 for i in range(int(sum(ti)/periodo)+1)]
plt.plot(tiempo, q1s, label="q1")
plt.plot(tiempo, q2s, label="q2")
plt.legend()
plt.show()

plt.plot(tiempo, v_q1s, label="v_q1")
plt.plot(tiempo, v_q2s, label="v_q2")
plt.legend()
plt.show()

plt.plot(tiempo, a_q1s, label="a_q1")
plt.plot(tiempo, a_q2s, label="a_q2")
plt.legend()
plt.show()
```

cinematico_directo.py

```python
# Javier Gálvez Obispo

import numpy as np
import matplotlib.pyplot as plt

def pcd(q1, q2, l1, l2):
    x = l1*np.cos(q1) + l2*np.cos(q1+q2)
    y = l1*np.sin(q1) + l2*np.sin(q1+q2)
    return (x, y)

def dibujar_trayectoria_pcd(q1s, q2s, l1, l2):
    # Calculamos todos los puntos y los pintamos
    puntos = [pcd(q1, q2, l1, l2) for q1, q2 in zip(q1s, q2s)]
    plt.plot(*zip(*puntos))

    dibujar_robot(q1s[-1], q2s[-1], l1, l2)

    # Centrar el origen de coordenadas
    rango = np.arange(-l1-l2, l1+l2+1)
    plt.xticks(rango)
    plt.yticks(rango)

    plt.show()

def dibujar_robot(q1, q2, l1, l2):
    x0, y0 = 0, 0                       # Posición de la articulación 1
    x1, y1 = pcd(q1, 0, l1, 0)          # Posición de la articulación 2
    x2, y2 = pcd(q1, q2, l1, l2)        # Posición del extremo del robot

    x, y = [x0, x1, x2], [y0, y1, y2]   # Coordenadas de la trayectoria
    plt.plot(x, y, 'k')                 # Traza la trayectoria
    plt.plot(x0, y0, 'k.')              # Dibuja la articulación 1
    plt.plot(x1, y1, 'k.')              # Dibuja la articulación 2

def animacion_trayectoria_pcd(q1s, q2s, l1, l2):
    n = min(len(q1s), len(q2s))
    for i in range(1, n):
        plt.clf()
```

```
        dibujar_trayectoria_pcd(q1s[0:i], q2s[0:i], l1, l2)
        plt.pause(0.001)

if __name__ == "__main__":
    n = 100
    l1 = l2 = 2

    q1s = [np.pi / 4] * n
    q2s = np.linspace(0, np.pi/2, n)
    dibujar_trayectoria_pcd(q1s, q2s, l1, l2)

    q1s = np.linspace(np.pi/4, np.pi/2, n)
    q2s = np.linspace(0, np.pi/2, n)
    dibujar_trayectoria_pcd(q1s, q2s, l1, l2)

    q1s = [i*np.pi / n for i in range(n+1)]
    q2s = q1s[:n//2+1] + q1s[n//2-1::-1]
    animacion_trayectoria_pcd(q1s, q2s, l1, l2)
```

cinematico_inverso.py

```
# Javier Gálvez Obispo

import numpy as np
import matplotlib.pyplot as plt

from cinematico_directo import dibujar_robot

def pci(x, y, l1, l2):
    cosq2 = (x**2 + y**2 - l1**2 - l2**2) / (2*l1*l2)
    sinq2 = np.sqrt(1-cosq2**2)

    # q1 = np.arctan(y/x) - np.arctan(l2*sinq2 / (l1 + l2*cosq2))
    # q2 = np.arctan(sinq2/cosq2)

    q1 = np.arctan2(y, x) - np.arctan2(l2*sinq2, (l1 + l2*cosq2))
    q2 = np.arctan2(sinq2, cosq2)

    q1_neg = np.arctan2(y, x) - np.arctan2(l2*(-sinq2), (l1 + l2*cosq2))
```

```
    q2_neg = np.arctan2(-sinq2, cosq2)

    return (q1, q2, q1_neg, q2_neg)

def dibujar_trayectoria_pci(xs, ys, l1, l2, qi_anteriores):
    # Pintamos la trayectoria
    plt.plot(xs, ys)

    # Pintamos el robot
    q1, q2, q1_neg, q2_neg = pci(xs[-1], ys[-1], l1, l2)

    if qi_anteriores is not None:
        variacion_q1 = abs(qi_anteriores[0]-q1)
        variacion_q1_neg = abs(qi_anteriores[0]-q1_neg)

        # Los dos primeros casos son para el cambio del tercer cuadrante
        # al cuarto que se pasa de 3pi/2 a -pi/2 o el caso contrario.
        # Entonces, la diferencia de los ángulos es muy grande aunque
        # en realidad la variación en el espacio es pequeña
        if variacion_q1 > np.pi + np.pi/2:
            qi_usados = (q1, q2)
        elif variacion_q1_neg > np.pi + np.pi/2:
            qi_usados = (q1_neg, q2_neg)

        # Se compara cuál de los dos ángulos es más próximo
        elif variacion_q1 < variacion_q1_neg:
            qi_usados = (q1, q2)
        else:
            qi_usados = (q1_neg, q2_neg)
    else:
        qi_usados = (q1, q2)

    dibujar_robot(*qi_usados, l1, l2)

    # Centrar el origen de coordenadas
    rango = np.arange(-l1-l2-1, l1+l2+2)
    plt.xticks(rango)
    plt.yticks(rango)

    plt.show()
```

```
    return qi_usados

def animacion_trayectoria_pci(xs, ys, l1, l2):
    n = min(len(xs), len(ys))
    qi_anteriores = None
    for i in range(1, n):
        plt.clf()
        qi_anteriores = dibujar_trayectoria_pci(xs[0:i], ys[0:i],
                                                l1, l2, qi_anteriores)
        plt.pause(0.001)

if __name__ == "__main__":
    n = 100
    l1 = l2 = 2

    xs = np.linspace(2, 0, n)
    ys = np.linspace(0, 2, n)
    animacion_trayectoria_pci(xs, ys, l1, l2)

    xs = np.linspace(1, -2, n)
    ys = [1] * n
    animacion_trayectoria_pci(xs, ys, l1, l2)

    xs = np.linspace(1, -1, n)
    ys = [0] * n
    animacion_trayectoria_pci(xs, ys, l1, l2)

    xs = np.linspace(-2, 2, n)
    ys = np.linspace(0, 1, n)
    animacion_trayectoria_pci(xs, ys, l1, l2)
```