Práctica 2

Raúl Soria González

6 de mayo de 2021

Ejercicio 1

Para el primer ejercicio se nos pedía crear una función `pcd` que resuelva el problema cinemático directo para un manipulador RR.

Yo he decidido realizarlo mediante le método geométrico ya que es un robot muy sencillo con únicamente 2 grados de libertad.

$x1 = l1 * np.cos(q1) + l2 * np.cos(q1 + q2)$

$y1 = l1 * np.sin(q1) + l2 * np.sin(q1 + q2)$

Ejercicio 2

Para este ejercicio se nos pedía crear otra función `dibujar_trayectoria_pcd` que calculase y dibujase las coordenadas finales de los manipuladores para un conjunto de ángulos de giro de las articulaciones. El resultado es el mismo que para el del ejercicio 3 pero sin dibujar el robot (Figura 1).

Ejercicio 3

Ahora teníamos que crear una función que dibujase un robot dados los ángulos de sus articulaciones y la longitud de sus eslabones `dibujar_robot`. Teníamos que llamar a esta función dentro de la función `dibujar_trayectoria_pcd` con los últimos valores de los vectores de ángulos de las articulaciones.

Ejercicio 4

En el cuarto ejercicio se nos pedía dibujar la trayectoria de un robot formada por 100 puntos. Para ello había que pasarle a la función `dibujar_trayectoria_pcd` dos vectores de 100 puntos cada uno. El primero de los vectores (para la articulación q1) tenía que tener siempre el mismo valor: $\pi/4$, mientras que la articulación q2 variaría entre 0 y $\pi/2$.

Para dibujar la trayectoria como una linea que pasase por todos los puntos que calcula la función `pcd` he creado dos vectores `vx` y `vt` que van a almacenar estos puntos y dentro de un `for` que itera los vectores de q1s y q2s, voy calculando el pcd y añadiendo los resultados a estos vectores para luego mostrarlos con `plot`.

El resultado podemos verlo en la Figura 2.

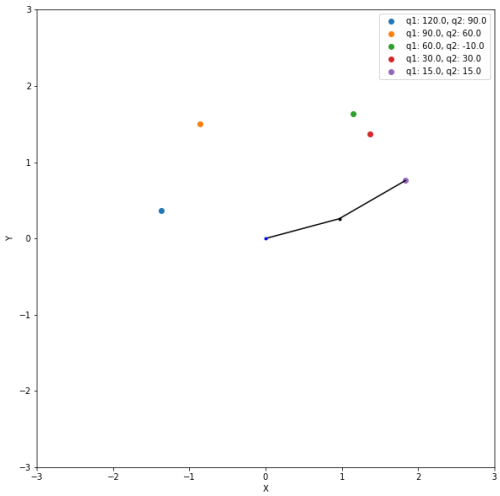

Figura 1: Ejercicio 3

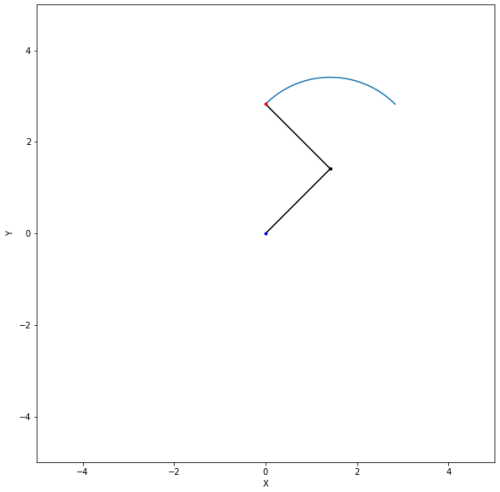

Figura 2: Ejercicio 4

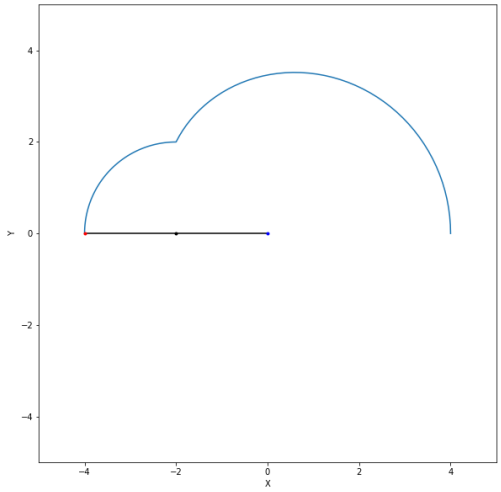

Figura 3: Ejercicio 6

Ejercicio 5

Ahora se nos pedía que se viese una animación del movimiento del robot siguiendo una trayectoria. Para ello llamamos a una función `animacion_trayectoria_pcd`, una función que iría llamando a la función `dibujar_trayectoria_pcd` tantas veces como ángulos tengan los vectores q1s y q2s. En la primera iteración llama a la función con sólo los 2 primeros ángulos y por cada iteración va aumentando en 1 el número de ángulos que le pasa a `dibujar_trayectoria_pcd`, tras cada iteración borra lo que había en la figura plot. Así se consigue un efecto de animación.

Ejercicio 6

Para este ejercicio tan sólo se nos pedía probar la función creada en el ejercicio 5. Esta vez había que pasar los siguientes conjunto de ángulos:

$q1 \rightarrow 0, \pi/100, 2\pi/100, ..., \pi$

$q2 \rightarrow 0, \pi/100, 2\pi/100, ..., 50\pi/100, 49\pi/100, 48\pi/100, ..., 0$

Por motivos obvios no puedo mostrar la animación en este PDF pero el resultado final es le de la Figura 3.

Ejercicio 7

En este ejercicio se nos pedía resolver el problema cinemático inverso, de un modo similar a como hicimos en el ejercicio 1 con el directo.

Para ello he decidido emplear nuevamente el método geométrico debido a la simplicidad del manipulador.

A la función `pci` se le pasa una cordenada (x, y) además de las longitudes de los eslabones. Como resultado devuelve los ángulos q1 y q2 del robot. Los cálculos de los ángulos se realizan de la siguiente forma:

$cosq2 = (xf * *2 + yf * *2 - l1 * *2 - l2 * *2)/(2 * l1 * l2)$
$sinq2 = np.sqrt(1 - cosq2 * *2)$
$q2 = np.arctan2(sinq2, cosq2)$
$alpha = np.arctan((l2 * sinq2)/(l1 + l2 * cosq2))$
$beta = np.arctan2(yf, xf)$
$q1 = beta - alpha$

Ejercicio 8

Ahora se nos pide implementar las funciones `dibujar_trayectoria_pci` y `animacion_trayectoria_pci`, equivalentes a las implementadas para el modelo cinemático directo.

Para la función `dibujar_trayectoria_pci` tan sólo tenemos que calcular los ángulos de las articulaciones en las últimas coordenadas que se le pasa. Con estos ángulos dibujaremos la posición final del robot. La trayectoria simplemente se puede dibujar con los vectores `xs` y `ys` que se nos pasan como parámetros.

La función de `animacion_trayectoria_pci` es igual a la del problema cinemático directo pero esta vez trabajamos con vectores de coordenadas en lugar de ángulos.

Ejercicio 9

Para este ejercicio se nos pide analizar si las funciones implementadas en el ejercicio anterior funcionan correctamente.

La animación del robot sigue la trayectoria correcta pero el robot realiza algunos movimientos antinaturales debido a que hay 2 soluciones para le pci y no se elige la opción correcta. Esto lo solucionamos en el ejercicio 10.

Desde un principio he utilizado la función `arctan2` para resolver el problema cinemático inverso ya que si utilizaba `arctan`, al intentar resolver alunos pci me daba un fallo dado que dividía por 0. Esto se solventa con el uso de

$$\operatorname{atan2}(y,x) = \begin{cases} \arctan(\frac{y}{x}) & \text{if } x > 0, \\ \arctan(\frac{y}{x}) + \pi & \text{if } x < 0 \text{ and } y \geq 0, \\ \arctan(\frac{y}{x}) - \pi & \text{if } x < 0 \text{ and } y < 0, \\ +\frac{\pi}{2} & \text{if } x = 0 \text{ and } y > 0, \\ -\frac{\pi}{2} & \text{if } x = 0 \text{ and } y < 0, \\ \text{undefined} & \text{if } x = 0 \text{ and } y = 0. \end{cases}$$

Figura 4: arctan2

`arctan2` ya que esta función recibe dos parámetros y realiza la arcotangente del primero partido del segundo. Además tiene en cuenta que los parámetros puedan ser 0 y solventa el problema de dividir entre 0. La función `arctan2` sigue el esquema de la figura 4.

Ejercicio 10

Para solventar el problema de los movimientos antinaturales que nos encontramos al animar los robots en el pci es necesario que la función `pci` nos devuelva ambos posibles resultados. Encontramos dos posibles resultados ya que el seno de q2 se calcula de la siguiente forma: $\pm\sqrt{1 - \cos^2 q2}$ por lo que tenemos dos posibles valores.

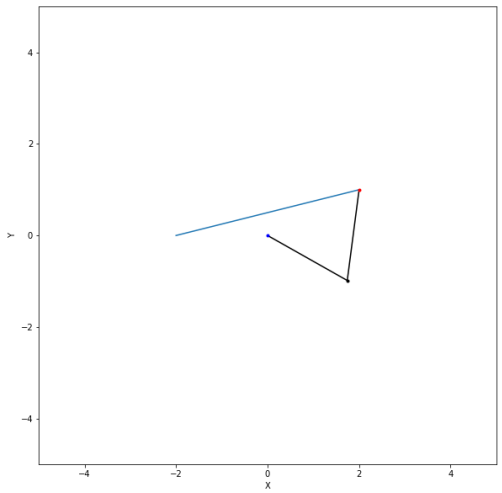

Figura 5: Ejercicio 10

Para elegir qué valor es el que debemos elegir utilizamos el criterio de que el valor absoluto de la diferencia entre el seno de q1 con el seno del anterior q1 de la animación, más el valor absoluto de la diferencia de los cosenos de ambos q1s más la diferencia de senos y cosenos de q2 con le anterior q2, sea la menor de entre los dos posibles valores que pueden tomar los ángulos al resolver el pci. De esta forma cada movimiento que realice el robot será para llegar a una posición lo más parecida posible a la anterior.

Ahora sí, todos las trayectorias que se proponen en el ejercicio 9 se realizan correctamente y sin ningún movimiento extraño del robot. Podemos ver el resultado en la Figura 5.

Archivos de código

ejer1.py

```
import numpy as np


def pcd(q1, q2, l1, l2):
    x1 = l1 * np.cos(q1) + l2 * np.cos(q1 + q2)
    y1 = l1 * np.sin(q1) + l2 * np.sin(q1 + q2)

    return x1.round(5), y1.round(5)


x, y = pcd(np.deg2rad(90), np.deg2rad(—90), 2, 1)

print('x:', x, 'y:', y)
```

ejer2.py

```
import numpy as np
import matplotlib.pyplot as plt


def pcd(q1, q2, l1, l2):
    x1 = l1 * np.cos(q1) + l2 * np.cos(q1 + q2)
    y1 = l1 * np.sin(q1) + l2 * np.sin(q1 + q2)

    return x1.round(5), y1.round(5)


def dibujar_trayectoria_pcd(q1s, q2s, l1, l2):
    plt.plot(0, 0, 'ko', label='Origen')
    for i in range(q1s.size):
        x, y = pcd(q1s[i], q2s[i], l1, l2)
        label = 'q1: ' + str(np.rad2deg(q1s[i]).round(3)) + \
                ', q2: ' + str(np.rad2deg(q2s[i]).round(3))
        plt.plot(x, y, 'o', label=label)

    ax = plt.gca()
    fig = plt.gcf()
```

```
    ax.set_aspect('equal')
    fig.set_size_inches(10, 10)
    plt.xlabel("X")
    plt.ylabel("Y")
    plt.ylim([- (l1 + l2 + 1), l1 + l2 + 1])
    plt.xlim([- (l1 + l2 + 1), l1 + l2 + 1])
    plt.legend()
    plt.show()


Vq1 = np.array([np.deg2rad(120), np.deg2rad(90),
                np.deg2rad(60), np.deg2rad(30), np.deg2rad(15)])
Vq2 = np.array([np.deg2rad(90), np.deg2rad(60),
                np.deg2rad(-10), np.deg2rad(30), np.deg2rad(15)])

dibujar_trayectoria_pcd(Vq1, Vq2, 1, 1)
```

ejer3.py

```
import numpy as np
import matplotlib.pyplot as plt


def pcd(q1, q2, l1, l2):
    x1 = l1 * np.cos(q1) + l2 * np.cos(q1 + q2)
    y1 = l1 * np.sin(q1) + l2 * np.sin(q1 + q2)

    return x1.round(5), y1.round(5)


def dibujar_trayectoria_pcd(q1s, q2s, l1, l2):
    for i in range(q1s.size):
        x, y = pcd(q1s[i], q2s[i], l1, l2)
        label = 'q1: ' + str(np.rad2deg(q1s[i]).round(3)) + \
                ', q2: ' + str(np.rad2deg(q2s[i]).round(3))
        plt.plot(x, y, 'o', label=label)

    ax = plt.gca()
    fig = plt.gcf()
```

```
    ax.set_aspect('equal')
    fig.set_size_inches(10, 10)
    plt.xlabel("X")
    plt.ylabel("Y")
    plt.ylim([- (l1 + l2 + 1), l1 + l2 + 1])
    plt.xlim([- (l1 + l2 + 1), l1 + l2 + 1])
    plt.legend()
    dibujar_robot(q1s[q1s.size - 1], q2s[q2s.size - 1], l1, l2)
    plt.show()


def dibujar_robot(q1, q2, l1, l2):
    x0, y0 = 0, 0
    x1, y1 = pcd(q1, 0, l1, 0)
    x2, y2 = pcd(q1, q2, l1, l2)
    x, y = [x0, x1, x2], [y0, y1, y2]
    plt.plot(x, y, 'k')
    plt.plot(x1, y1, 'k.')
    plt.plot(x0, y0, 'b.')


Vq1 = np.array([np.deg2rad(120), np.deg2rad(90),
                np.deg2rad(60), np.deg2rad(30), np.deg2rad(15)])
Vq2 = np.array([np.deg2rad(90), np.deg2rad(60),
                np.deg2rad(-10), np.deg2rad(30), np.deg2rad(15)])

dibujar_trayectoria_pcd(Vq1, Vq2, 1, 1)
```

ejer4.py

```
import numpy as np
import matplotlib.pyplot as plt


def pcd(q1, q2, l1, l2):
    x1 = l1 * np.cos(q1) + l2 * np.cos(q1 + q2)
    y1 = l1 * np.sin(q1) + l2 * np.sin(q1 + q2)

    return x1.round(5), y1.round(5)
```

```
def dibujar_trayectoria_pcd(q1s, q2s, l1, l2):
    vx, vy = [], []
    for i in range(q1s.size):
        x, y = pcd(q1s[i], q2s[i], l1, l2)
        vx.append([x])
        vy.append([y])

    plt.plot(vx, vy)
    ax = plt.gca()
    fig = plt.gcf()
    ax.set_aspect('equal')
    fig.set_size_inches(10, 10)
    plt.xlabel("X")
    plt.ylabel("Y")
    plt.ylim([- (l1 + l2 + 1), l1 + l2 + 1])
    plt.xlim([- (l1 + l2 + 1), l1 + l2 + 1])
    dibujar_robot(q1s[q1s.size - 1], q2s[q2s.size - 1], l1, l2)
    plt.show()


def dibujar_robot(q1, q2, l1, l2):
    x0, y0 = 0, 0
    x1, y1 = pcd(q1, 0, l1, 0)
    x2, y2 = pcd(q1, q2, l1, l2)
    x, y = [x0, x1, x2], [y0, y1, y2]
    plt.plot(x, y, 'k')
    plt.plot(x1, y1, 'k.')
    plt.plot(x2, y2, 'r.')
    plt.plot(x0, y0, 'b.')


Vq1 = np.full(100, np.pi / 4)
Vq2 = np.linspace(0, np.pi / 2, 100)

dibujar_trayectoria_pcd(Vq1, Vq2, 2, 2)
```

ejer5.py

```
import numpy as np
import matplotlib.pyplot as plt


def pcd(q1, q2, l1, l2):
    x1 = l1 * np.cos(q1) + l2 * np.cos(q1 + q2)
    y1 = l1 * np.sin(q1) + l2 * np.sin(q1 + q2)

    return x1.round(5), y1.round(5)


def dibujar_trayectoria_pcd(q1s, q2s, l1, l2):
    vx, vy = [], []
    for i in range(q1s.size):
        x, y = pcd(q1s[i], q2s[i], l1, l2)
        vx.append([x])
        vy.append([y])

    plt.plot(vx, vy)
    ax = plt.gca()
    fig = plt.gcf()
    ax.set_aspect('equal')
    fig.set_size_inches(10, 10)
    plt.xlabel("X")
    plt.ylabel("Y")
    plt.ylim([- (l1 + l2 + 1), l1 + l2 + 1])
    plt.xlim([- (l1 + l2 + 1), l1 + l2 + 1])
    dibujar_robot(q1s[q1s.size - 1], q2s[q2s.size - 1], l1, l2)
    plt.show()


def dibujar_robot(q1, q2, l1, l2):
    x0, y0 = 0, 0
    x1, y1 = pcd(q1, 0, l1, 0)
    x2, y2 = pcd(q1, q2, l1, l2)
    x, y = [x0, x1, x2], [y0, y1, y2]
    plt.plot(x, y, 'k')
    plt.plot(x1, y1, 'k.')
```

```
    plt.plot(x2, y2, 'r.')
    plt.plot(x0, y0, 'b.')


def animacion_trayectoria_pcd(q1s, q2s, l1, l2):
    n = min(len(q1s), len(q2s))
    for i in range(1, n+1):
        plt.clf()
        dibujar_trayectoria_pcd(q1s[0:i], q2s[0:i], l1, l2)
        plt.pause(0.001)


Vq1 = np.full(100, np.pi / 4)
Vq2 = np.linspace(0, np.pi / 2, 100)

animacion_trayectoria_pcd(Vq1, Vq2, 2, 2)
```

ejer6.py

```
import numpy as np
import matplotlib.pyplot as plt


def pcd(q1, q2, l1, l2):
    x1 = l1 * np.cos(q1) + l2 * np.cos(q1 + q2)
    y1 = l1 * np.sin(q1) + l2 * np.sin(q1 + q2)

    return x1.round(5), y1.round(5)


def dibujar_trayectoria_pcd(q1s, q2s, l1, l2):
    vx, vy = [], []
    for i in range(q1s.size):
        x, y = pcd(q1s[i], q2s[i], l1, l2)
        vx.append([x])
        vy.append([y])

    plt.plot(vx, vy)
    ax = plt.gca()
```

```
    fig = plt.gcf()
    ax.set_aspect('equal')
    fig.set_size_inches(10, 10)
    plt.xlabel("X")
    plt.ylabel("Y")
    plt.ylim([- (l1 + l2 + 1), l1 + l2 + 1])
    plt.xlim([- (l1 + l2 + 1), l1 + l2 + 1])
    dibujar_robot(q1s[q1s.size - 1], q2s[q2s.size - 1], l1, l2)
    plt.show()


def dibujar_robot(q1, q2, l1, l2):
    x0, y0 = 0, 0
    x1, y1 = pcd(q1, 0, l1, 0)
    x2, y2 = pcd(q1, q2, l1, l2)
    x, y = [x0, x1, x2], [y0, y1, y2]
    plt.plot(x, y, 'k')
    plt.plot(x1, y1, 'k.')
    plt.plot(x2, y2, 'r.')
    plt.plot(x0, y0, 'b.')


def animacion_trayectoria_pcd(q1s, q2s, l1, l2):
    n = min(len(q1s), len(q2s))
    for i in range(1, n+1):
        plt.clf()
        dibujar_trayectoria_pcd(q1s[0:i], q2s[0:i], l1, l2)
        plt.pause(0.001)


Vq1 = np.linspace(0, np.pi, 101)
primera_mitad = np.linspace(0, 50 * np.pi / 100, 51)
segunda_mitad = primera_mitad[49::-1]
Vq2 = np.append(primera_mitad, segunda_mitad)

animacion_trayectoria_pcd(Vq1, Vq2, 2, 2)
```

ejer7.py

```
import numpy as np
import matplotlib.pyplot as plt


def pci(xf, yf, l1, l2):
    cosq2 = (xf ** 2 + yf ** 2 — l1 ** 2 — l2 ** 2) / (2 * l1 * l2)
    sinq2 = np.sqrt(1 — cosq2 ** 2)
    q2 = np.arctan2(sinq2, cosq2)

    alpha = np.arctan((l2 * sinq2) / (l1 + l2 * cosq2))
    beta = np.arctan2(yf, xf)
    q1 = beta — alpha

    return q1, q2


q1f, q2f = pci(1, 0.5, 1, 1)

print('q1:', np.rad2deg(q1f), 'q2:', np.rad2deg(q2f))
```

ejer8.py

```
import numpy as np
import matplotlib.pyplot as plt


def pcd(q1, q2, l1, l2):
    x1 = l1 * np.cos(q1) + l2 * np.cos(q1 + q2)
    y1 = l1 * np.sin(q1) + l2 * np.sin(q1 + q2)

    return x1.round(5), y1.round(5)


def pci(xf, yf, l1, l2):
    cosq2 = (xf ** 2 + yf ** 2 — l1 ** 2 — l2 ** 2) / (2 * l1 * l2)
    sinq2 = np.sqrt(1 — cosq2 ** 2)
    q2 = np.arctan2(sinq2, cosq2)
```

```
    alpha = np.arctan2((l2 * sinq2), (l1 + l2 * cosq2))
    beta = np.arctan2(yf, xf)
    q1 = beta - alpha

    return q1, q2


def dibujar_trayectoria_pci(xs, ys, l1, l2):
    q1, q2 = pci(xs[len(xs) - 1], ys[len(ys) - 1], l1, l2)

    plt.plot(xs, ys)
    ax = plt.gca()
    fig = plt.gcf()
    ax.set_aspect('equal')
    fig.set_size_inches(10, 10)
    plt.xlabel("X")
    plt.ylabel("Y")
    plt.ylim([- (l1 + l2 + 1), l1 + l2 + 1])
    plt.xlim([- (l1 + l2 + 1), l1 + l2 + 1])
    dibujar_robot(q1, q2, l1, l2)
    plt.show()


def dibujar_robot(q1, q2, l1, l2):
    x0, y0 = 0, 0
    x1, y1 = pcd(q1, 0, l1, 0)
    x2, y2 = pcd(q1, q2, l1, l2)
    x, y = [x0, x1, x2], [y0, y1, y2]
    plt.plot(x, y, 'k')
    plt.plot(x1, y1, 'k.')
    plt.plot(x2, y2, 'r.')
    plt.plot(x0, y0, 'b.')


def animacion_trayectoria_pci(xs, ys, l1, l2):
    n = min(len(xs), len(ys))
    for i in range(1, n+1):
        plt.clf()
        dibujar_trayectoria_pci(xs[0:i], ys[0:i], l1, l2)
        plt.pause(0.001)
```

```
Vx = np.linspace(1, —1, 100)
Vy = np.linspace(0, 0, 100)

animacion_trayectoria_pci(Vx, Vy, 2, 2)
```

ejer10.py

```
import numpy as np
import matplotlib.pyplot as plt


def pcd(q1, q2, l1, l2):
    x1 = l1 * np.cos(q1) + l2 * np.cos(q1 + q2)
    y1 = l1 * np.sin(q1) + l2 * np.sin(q1 + q2)

    return x1.round(5), y1.round(5)


def pci(xf, yf, l1, l2):
    cosq2 = (xf ** 2 + yf ** 2 — l1 ** 2 — l2 ** 2) / (2 * l1 * l2)
    sinq2 = np.sqrt(1 — cosq2 ** 2)
    q2 = np.arctan2(sinq2, cosq2)

    alpha = np.arctan2((l2 * sinq2), (l1 + l2 * cosq2))
    beta = np.arctan2(yf, xf)
    q1 = beta — alpha

    sinq2 = —np.sqrt(1 — cosq2 ** 2)
    q2_2 = np.arctan2(sinq2, cosq2)

    alpha = np.arctan2((l2 * sinq2), (l1 + l2 * cosq2))
    q1_2 = beta — alpha

    return [q1, q1_2], [q2, q2_2]


def dibujar_trayectoria_pci(xs, ys, l1, l2, q1_anterior, q2_anterior):
    q1s, q2s = pci(xs[len(xs) — 1], ys[len(ys) — 1], l1, l2)
```

```
    if q1_anterior == -1:
        q1_anterior = q1s[0]
        q2_anterior = q2s[0]

    if (abs(np.sin(q1s[0]) - np.sin(q1_anterior)) +
            abs(np.cos(q1s[0]) - np.cos(q1_anterior)) +
            abs(np.sin(q2s[0]) - np.sin(q2_anterior)) +
            abs(np.cos(q2s[0]) - np.cos(q2_anterior)) <=
            abs(np.sin(q1s[1]) - np.sin(q1_anterior)) +
            abs(np.cos(q1s[1]) - np.cos(q1_anterior)) +
            abs(np.sin(q2s[1]) - np.sin(q2_anterior)) +
            abs(np.cos(q2s[1]) - np.cos(q2_anterior))):
        q1 = q1s[0]
        q2 = q2s[0]
    else:
        q1 = q1s[1]
        q2 = q2s[1]

    plt.plot(xs, ys)
    ax = plt.gca()
    fig = plt.gcf()
    ax.set_aspect('equal')
    fig.set_size_inches(8, 8)
    plt.xlabel("X")
    plt.ylabel("Y")
    plt.ylim([-(l1 + l2 + 1), l1 + l2 + 1])
    plt.xlim([-(l1 + l2 + 1), l1 + l2 + 1])
    dibujar_robot(q1, q2, l1, l2)
    plt.show()

    return q1


def dibujar_robot(q1, q2, l1, l2):
    x0, y0 = 0, 0
    x1, y1 = pcd(q1, 0, l1, 0)
    x2, y2 = pcd(q1, q2, l1, l2)
    x, y = [x0, x1, x2], [y0, y1, y2]
    plt.plot(x, y, 'k')
    plt.plot(x1, y1, 'k.')
```

```
    plt.plot(x2, y2, 'r.')
    plt.plot(x0, y0, 'b.')


def animacion_trayectoria_pci(xs, ys, l1, l2):
    n = min(len(xs), len(ys))
    q1_a = —1
    q2_a = —1
    for i in range(1, n + 1):
        plt.clf()
        q1_a = dibujar_trayectoria_pci(xs[0:i], ys[0:i],
                                       l1, l2, q1_a, q2_a)
        plt.pause(0.001)


Vx = np.linspace(1, —1, 100)
Vy = np.linspace(0, 0, 100)

animacion_trayectoria_pci(Vx, Vy, 2, 2)
```