

Robótica Computacional

Samuel Martín Morales `alu0101359526@ull.edu.es`

22 de diciembre de 2023

Índice general

1. Introducción	2
2. Cinemática directa mediante Denavit Hartenberg	3
2.1. ¿Qué es la Cinemática Directa?	3
2.2. Explicación del código suministrado	3
2.3. Implementación de la Cinemática Directa	6
2.3.1. Representación gráfica del manipulador	6
2.3.2. Implementación de la cinemática directa del robot número 5	7
2.4. Ejecución del código	10
2.5. Parámetros modificados	11
2.6. Mejoras en la implementación	12
2.7. Conclusiones	12
3. Cinemática inversa	13
3.1. ¿Qué es la Cinemática Inversa?	13
3.2. Explicación del código suministrado	13
3.3. Implementación de la Cinemática Inversa	16
3.4. Ejecución del código	19
3.5. Parámetros modificados	20
3.6. Conclusiones	20
4. Localización	21
4.1. ¿Qué es la localización?	21
4.2. Explicación del código suministrado	21
4.3. Implementación de la localización	21
4.4. Ejecución del código	21
4.5. Parámetros modificados	21
4.6. Mejoras en la implementación	21
4.7. Conclusiones	21
5. Filtro de partículas	22
5.1. ¿Qué es un filtro de partículas?	22
6. Conclusiones	23

Capítulo 1 Introducción

Durante el desarrollo de la asignatura *Robótica Computacional* se ha realizado la implementación de tres prácticas relacionadas con el contenido teórico que se ha visto a lo largo de las semanas lectivas. Además de las tres prácticas mencionadas, se ha solicitado una cuarta práctica denominada como *Filtro de partículas*, la cual se ha determinado como práctica opcional. Por ello, en este documento se detallará el desarrollo de las tres primeras prácticas, dejando la cuarta práctica como trabajo futuro, debido a la falta de tiempo para su desarrollo.

Capítulo 2 Cinemática directa mediante Denavit Hartenberg

2.1. ¿Qué es la Cinemática Directa?

La cinemática de los manipuladores es la ciencia que estudia el movimiento de los componentes físicos del robot sin tener en cuenta o considerar las fuerzas que intervienen en el movimiento. Dentro de la cinemática se realiza el estudio de la velocidad, la posición, la aceleración con respecto al tiempo o a cualquier otra variable. Por tanto, la cinemática directa es el problema estático geométrico de calcular la posición y la orientación del efector terminal de manipulador. Para poder enfrentar este problema es necesario utilizar herramientas como las matrices de rotación y traslación, las cuales permiten representar el movimiento de un sistema de referencia a otro.

En cuanto al caso concreto de la cinemática directa mediante Denavit Hartenberg, se trata de un método para describir la cinemática de un manipulador articulado. Este método fue desarrollado por Jacques Denavit y Richard Hartenberg en 1955. El método de Denavit-Hartenberg es un método de notación para describir los sistemas de coordenadas de los eslabones de un robot manipulador. Este método se basa en la asignación de un sistema de coordenadas a cada uno de los eslabones del robot, de tal forma que se pueda describir la posición y orientación de cada uno de los eslabones con respecto a su eslabón anterior. Para ello, se utiliza una serie de parámetros que se asignan a cada uno de los eslabones del robot, los cuales son:

- **Distancia** d_i : Distancia entre los ejes Z_{i-1} y Z_i a lo largo del eje X_i .
- **Ángulo** θ_i : Ángulo entre los ejes X_{i-1} y X_i medido alrededor del eje Z_{i-1} .
- **Distancia** a_i : Distancia entre los ejes X_{i-1} y X_i a lo largo del eje Z_{i-1} .
- **Ángulo** α_i : Ángulo entre los ejes Z_{i-1} y Z_i medido alrededor del eje X_i .
- **Ángulo** θ_i : Ángulo entre los ejes X_{i-1} y X_i medido alrededor del eje Z_{i-1} .

2.2. Explicación del código suministrado

El script en python suministrado para la implementación de la práctica de cinemática directa mediante Denavit Hartenberg se puede observar a continuación:

Listing 2.1: Script de Cinemática Directa

```
1 # Ejemplo:
2 # ./cdDH.py 30 45
3
4 import sys
5 from math import *
6 import numpy as np
7 import matplotlib.pyplot as plt
8 from mpl_toolkits.mplot3d import Axes3D
9
10 # *****
```

```

11 # Declaración de funciones
12
13 def ramal(I,prev=[],base=0):
14     # Convierte el robot a una secuencia de puntos para representar
15     O = []
16     if I:
17         if isinstance(I[0][0],list):
18             for j in range(len(I[0])):
19                 O.extend(ramal(I[0][j], prev, base or j < len(I[0])-1))
20         else:
21             O = [I[0]]
22             O.extend(ramal(I[1:],I[0],base))
23             if base:
24                 O.append(prev)
25     return O
26
27 def muestra_robot(O,ef=[]):
28     # Pinta en 3D
29     OR = ramal(O)
30     OT = np.array(OR).T
31     fig = plt.figure()
32     ax = fig.add_subplot(111, projection='3d')
33     # Bounding box cúbico para simular el ratio de aspecto correcto
34     max_range = np.array([OT[0].max()-OT[0].min()
35                           ,OT[1].max()-OT[1].min()
36                           ,OT[2].max()-OT[2].min()
37                           ]).max()
38     Xb = (0.5*max_range*np.mgrid[-1:2:2,-1:2:2,-1:2:2][0].flatten()
39           + 0.5*(OT[0].max()+OT[0].min()))
40     Yb = (0.5*max_range*np.mgrid[-1:2:2,-1:2:2,-1:2:2][1].flatten()
41           + 0.5*(OT[1].max()+OT[1].min()))
42     Zb = (0.5*max_range*np.mgrid[-1:2:2,-1:2:2,-1:2:2][2].flatten()
43           + 0.5*(OT[2].max()+OT[2].min()))
44     for xb, yb, zb in zip(Xb, Yb, Zb):
45         ax.plot([xb], [yb], [zb], 'w')
46     ax.plot3D(OT[0],OT[1],OT[2],marker='s')
47     ax.plot3D([0],[0],[0],marker='o',color='k',ms=10)
48     if not ef:
49         ef = OR[-1]
50     ax.plot3D([ef[0]],[ef[1]],[ef[2]],marker='s',color='r')
51     ax.set_xlabel('X')
52     ax.set_ylabel('Y')
53     ax.set_zlabel('Z')
54     plt.show()
55     return
56
57 def arbol_origenes(O,base=0,sufijo=''):
58     # Da formato a los orígenes de coordenadas para mostrarlos por pantalla
59     if isinstance(O[0],list):
60         for i in range(len(O)):
61             if isinstance(O[i][0],list):
62                 for j in range(len(O[i])):
63                     arbol_origenes(O[i][j],i+base,sufijo+str(j+1))
64             else:
65                 print('(0'+str(i+base)+sufijo+')0\t= '+str([round(j,3) for j in O[i]]))
66     else:

```

```

67     print('(0'+str(base)+sufijo+')0\t= '+str([round(j,3) for j in 0]))
68
69 def muestra_origenes(0,final=0):
70     # Muestra los orígenes de coordenadas para cada articulación
71     print('Orígenes de coordenadas:')
72     arbol_origenes(0)
73     if final:
74         print('E.Final = '+str([round(j,3) for j in final]))
75
76 def matriz_T(d,theta,a,alpha):
77     # Calcula la matriz T (ángulos de entrada en grados)
78     th=theta*pi/180;
79     al=alpha*pi/180;
80     return [[cos(th), -sin(th)*cos(al), sin(th)*sin(al), a*cos(th)]
81             , [sin(th), cos(th)*cos(al), -sin(al)*cos(th), a*sin(th)]
82             , [0, sin(al), cos(al), d]
83             , [0, 0, 0, 1]
84             ]
85 # *****
86
87
88 # Introducción de los valores de las articulaciones
89 nvar=2 # Número de variables
90 if len(sys.argv) != nvar+1:
91     sys.exit('El número de articulaciones no es el correcto ('+str(nvar)+')')
92 p=[float(i) for i in sys.argv[1:nvar+1]]
93
94 # Parámetros D-H:
95 #     1     2
96 d = [ 0, 0]
97 th = [p[0],p[1]]
98 a = [ 10, 5]
99 al = [ 0, 0]
100
101 # Orígenes para cada articulación
102 o00=[0,0,0,1]
103 o11=[0,0,0,1]
104 o22=[0,0,0,1]
105
106 # Cálculo matrices transformación
107 T01=matriz_T(d[0],th[0],a[0],al[0])
108 T12=matriz_T(d[1],th[1],a[1],al[1])
109 T02=np.dot(T01,T12)
110
111 # Transformación de cada articulación
112 o10 =np.dot(T01, o11).tolist()
113 o20 =np.dot(T02, o22).tolist()
114
115 # Mostrar resultado de la cinemática directa
116 muestra_origenes([o00,o10,o20])
117 muestra_robot ([o00,o10,o20])
118 input()

```

Como se puede ver en el script adjunto anteriormente, realiza la cinemática directa de un robot con dos articulaciones rotativas (para el caso del robot del script inicial) . Utiliza la representación de Denavit-Hartenberg (D-H) para describir la geometría del robot y calcula las matrices de transformación

homogénea para cada articulación. Los ángulos de las articulaciones se introducen como argumentos de línea de comandos.

El script utiliza funciones para convertir la descripción del robot en una secuencia de puntos para su representación visual en 3D. La salida incluye los orígenes de coordenadas y la representación gráfica del robot utilizando la biblioteca *matplotlib*.

En términos generales, el script demuestra cómo determinar la posición y orientación final de un robot manipulador dados los ángulos de sus articulaciones. Los resultados de la cinemática directa se muestran tanto numéricamente como en una representación gráfica del robot en un entorno tridimensional.

2.3. Implementación de la Cinemática Directa

Tras la explicación del funcionamiento del script en *Python* adjunto en el apartado anterior, se procede a realizar la implementación de la cinemática directa mediante Denavit Hartenberg de una serie de robots manipuladores que se pueden observar en el pdf adjunto [5]. Pero, para no extender demasiado el documento, se va a tomar como ejemplo la implementación de la cinemática directa mediante Denavit Hartenberg del manipulador número 5 del pdf adjunto [5].

2.3.1. Representación gráfica del manipulador

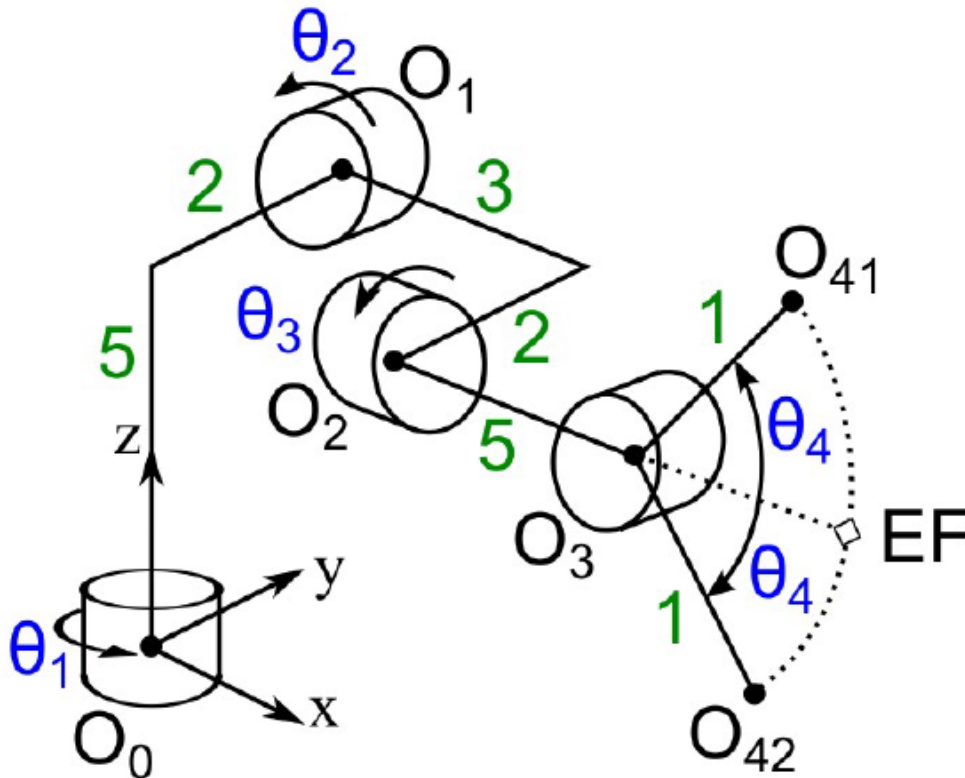


Figura 2.1: Representación gráfica del manipulador número 5

2.3.2. Implementación de la cinemática directa del robot número 5

Tras la visualización del robot número 5 en la figura 2.1, se procede a realizar la implementación de la cinemática directa del robot. Para ello, se va a utilizar el script en *Python* adjunto en el apartado anterior, modificando los parámetros de la tabla Denavit Hartenberg de la siguiente forma:

Cuadro 2.1: Parámetros D-H

Articulación	d_i	θ_i	a_i	α_i
05	5	$p[0]$	0	-90
1	2	0	0	0
15	0	$p[1]$	3	0
2	-2	-90	0	-90
3	5	$p[2]$	0	90
41	0	$-p[3] + 90$	1	0
42	0	$p[3] + 90$	1	0
ef	0	90	1	0

Una vez se tienen modificados los parámetros de la tabla, se procede a la modificación del número de manipuladores permitidos por el script, de tal manera que finalmente este toma el siguiente aspecto:

Listing 2.2: Script de Cinemática Directa del Manipulador 5

```
1 # Ejemplo:
2 # python3 cin_dir_5.py 10 20 30 30
3
4 # Cabe destacar que la ejecución de este script tiene la siguiente estructura de
   ejecución:
5 # python3 cin_dir_5.py <ángulo-o0> <ángulo-o2> <ángulo-o3> <ángulo-51-52>
6
7 import sys
8 from math import pi, cos, sin
9 import numpy as np
10 import matplotlib
11 # De esta manera se establece el motor gráfico que se debe de usar para MacOS
12 matplotlib.use('TkAgg')
13 import matplotlib.pyplot as plt
14 from mpl_toolkits.mplot3d import Axes3D
15
16 # *****
17 # Declaración de funciones
18
19 def ramal(I,prev=[],base=0):
20     # Convierte el robot a una secuencia de puntos para representar
21     O = []
22     if I:
23         if isinstance(I[0][0],list):
24             for j in range(len(I[0])):
25                 O.extend(ramal(I[0][j], prev, base or j < len(I[0])-1))
26         else:
27             O = [I[0]]
28             O.extend(ramal(I[1:],I[0],base))
29             if base:
30                 O.append(prev)
31     return O
```



```

32
33 def muestra_robot(0,ef=[]):
34     # Pinta en 3D
35     OR = ramal(0)
36     OT = np.array(OR).T
37     fig = plt.figure()
38     ax = fig.add_subplot(111, projection='3d')
39     # Bounding box cúbico para simular el ratio de aspecto correcto
40     max_range = np.array([OT[0].max()-OT[0].min()
41                           ,OT[1].max()-OT[1].min()
42                           ,OT[2].max()-OT[2].min()
43                           ]).max()
44     Xb = (0.5*max_range*np.mgrid[-1:2:2,-1:2:2,-1:2:2][0].flatten()
45           + 0.5*(OT[0].max()+OT[0].min()))
46     Yb = (0.5*max_range*np.mgrid[-1:2:2,-1:2:2,-1:2:2][1].flatten()
47           + 0.5*(OT[1].max()+OT[1].min()))
48     Zb = (0.5*max_range*np.mgrid[-1:2:2,-1:2:2,-1:2:2][2].flatten()
49           + 0.5*(OT[2].max()+OT[2].min()))
50     for xb, yb, zb in zip(Xb, Yb, Zb):
51         ax.plot([xb], [yb], [zb], 'w')
52     ax.plot3D(OT[0],OT[1],OT[2],marker='s')
53     ax.plot3D([0],[0],[0],marker='o',color='k',ms=10)
54     if not ef:
55         ef = OR[-1]
56     ax.plot3D([ef[0]],[ef[1]],[ef[2]],marker='s',color='r')
57     ax.set_xlabel('X')
58     ax.set_ylabel('Y')
59     ax.set_zlabel('Z')
60     plt.show()
61     return
62
63 def arbol_origenes(0,base=0,sufijo=''):
64     # Da formato a los orígenes de coordenadas para mostrarlos por pantalla
65     if isinstance(0[0],list):
66         for i in range(len(0)):
67             if isinstance(0[i][0],list):
68                 for j in range(len(0[i])):
69                     arbol_origenes(0[i][j],i+base,sufijo+str(j+1))
70             else:
71                 print(('('0'+str(i+base)+sufijo+')\t= '+str([round(j,3) for j in 0[i]])))
72     else:
73         print(('('0'+str(base)+sufijo+')\t= '+str([round(j,3) for j in 0])))
74
75 def muestra_origenes(0,final=0):
76     # Muestra los orígenes de coordenadas para cada articulación
77     print('Orígenes de coordenadas:')
78     arbol_origenes(0)
79     if final:
80         print(('E.Final = '+str([round(j,3) for j in final])))
81
82 def matriz_T(d,theta,a,alpha):
83     # Calcula la matriz T (ángulos de entrada en grados)
84     th=theta*pi/180;
85     al=alpha*pi/180;
86     return [[cos(th), -sin(th)*cos(al), sin(th)*sin(al), a*cos(th)]
87             , [sin(th), cos(th)*cos(al), -sin(al)*cos(th), a*sin(th)]

```

```

88         ,[      0,      sin(al),      cos(al),      d]
89         ,[      0,      0,      0,      1]
90     ]
91     # *****
92
93     plt.ion() # Modo interactivo
94     # Introducción de los valores de las articulaciones
95     nvar=4 # Número de variables
96     if len(sys.argv) != nvar+1:
97         sys.exit('El número de articulaciones no es el correcto ('+str(nvar)+')')
98     p=[float(i) for i in sys.argv[1:nvar+1]]
99
100    # Parámetros D-H:
101    #      05      1      15      2      3      41      42      ef
102    d = [ 5,      2,      0,      -2,      5,      0,      0,      0
103          ]
104    th = [ p[0],      0,      p[1],      -90,      p[2],      -p[3] + 90, p[3] + 90, 90
105           ]
106    a = [ 0,      0,      3,      0,      0,      1,      1,      1
107           ]
108    al = [ -90,      0,      0,      -90,      90,      0,      0,      0
109           ]
110
111    # Orígenes para cada articulación
112    o00=[0,0,0,1]
113    o05=[0,0,0,1]
114    o11=[0,0,0,1]
115    o15=[0,0,0,1]
116    o22=[0,0,0,1]
117    o33=[0,0,0,1]
118    o441=[0,0,0,1]
119    o442=[0,0,0,1]
120    oeff=[0,0,0,1]
121
122    # Cálculo matrices transformación
123    T005 = matriz_T(d[0],th[0],a[0],al[0])
124    T051 = matriz_T(d[1],th[1],a[1],al[1])
125    T115 = matriz_T(d[2],th[2],a[2],al[2])
126    T225 = matriz_T(d[3],th[3],a[3],al[3])
127    T033 = matriz_T(d[4],th[4],a[4],al[4])
128    T041 = matriz_T(d[5],th[5],a[5],al[5])
129    T042 = matriz_T(d[6],th[6],a[6],al[6])
130    T0ef = matriz_T(d[7],th[7],a[7],al[7])
131
132    T02 = np.dot(T005, T051)
133    T03 = np.dot(T02, T115)
134    T04 = np.dot(T03, T225)
135    T05 = np.dot(T04, T033)
136    T06 = np.dot(T05, T041)
137    T07 = np.dot(T06, T042)
138    T0ef = np.dot(T07, T0ef)
139
140    # Transformación de cada articulación
141    o10 = np.dot(T005, o05).tolist()
142    o20 = np.dot(T02, o11).tolist()

```

```

140 o30 = np.dot(T03, o15).tolist()
141 o40 = np.dot(T04, o22).tolist()
142 o50 = np.dot(T05, o33).tolist()
143 o60 = np.dot(T06, o441).tolist()
144 o70 = np.dot(T07, o442).tolist()
145 oeff = np.dot(T0ef, oeff).tolist()
146
147 # Mostrar resultado de la cinemática directa
148 muestra_origenes([o00,o10, o20, o30, o40, o50, [[o60], [o70]]], oeff)
149 muestra_robot    ([o00,o10, o20, o30, o40, o50, [[o60], [o70]]], oeff)
150 eval(input())
151
152 plt.show(block=True)

```

2.4. Ejecución del código

Para la ejecución del código, se debe ejecutar el siguiente comando en la terminal:

Listing 2.3: Ejecución del script de Cinemática Directa del Manipulador 5

```
1 $ python3 Manipulador-5.py 10 20 30 30
```

Tras esto, se obtiene el siguiente resultado:

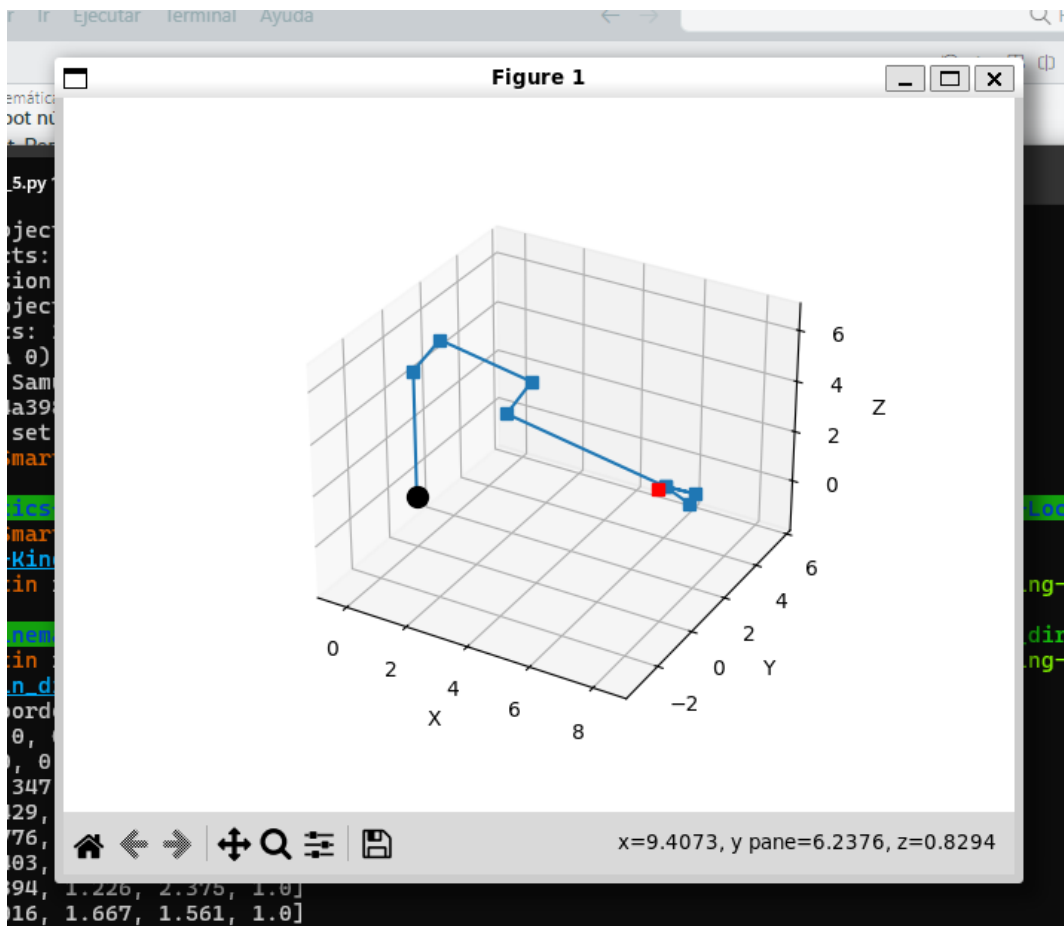


Figura 2.2: Resultado de la ejecución del script de Cinemática Directa del Manipulador 5

2.5. Parámetros modificados

Los parámetros modificados en la implementación de la cinemática directa del robot número 5 han sido los siguientes:

- **Distancia** d_i : Se ha modificado el valor de la distancia d_i de la articulación 05, de tal manera que se ha cambiado de 0 a 5.
- **Ángulo** θ_i : Se ha modificado el valor del ángulo θ_i de la articulación 05, de tal manera que se ha cambiado de 0 a $p[0]$.
- **Ángulo** α_i : Se ha modificado el valor del ángulo α_i de la articulación 05, de tal manera que se ha cambiado de 0 a -90.
- **Distancia** d_i : Se ha modificado el valor de la distancia d_i de la articulación 1, de tal manera que se ha cambiado de 0 a 2.
- **Ángulo** θ_i : Se ha modificado el valor del ángulo θ_i de la articulación 1, de tal manera que se ha cambiado de 0 a 0.
- **Distancia** a_i : Se ha modificado el valor de la distancia a_i de la articulación 15, de tal manera que se ha cambiado de 0 a 3.
- **Ángulo** α_i : Se ha modificado el valor del ángulo α_i de la articulación 15, de tal manera que se ha cambiado de 0 a 0.
- **Distancia** d_i : Se ha modificado el valor de la distancia d_i de la articulación 2, de tal manera que se ha cambiado de 0 a -2.
- **Ángulo** θ_i : Se ha modificado el valor del ángulo θ_i de la articulación 2, de tal manera que se ha cambiado de 0 a -90.
- **Ángulo** α_i : Se ha modificado el valor del ángulo α_i de la articulación 2, de tal manera que se ha cambiado de 0 a -90.
- **Distancia** d_i : Se ha modificado el valor de la distancia d_i de la articulación 3, de tal manera que se ha cambiado de 0 a 5.
- **Ángulo** θ_i : Se ha modificado el valor del ángulo θ_i de la articulación 3, de tal manera que se ha cambiado de 0 a $p[2]$.
- **Ángulo** α_i : Se ha modificado el valor del ángulo α_i de la articulación 3, de tal manera que se ha cambiado de 0 a 90.
- **Ángulo** θ_i : Se ha modificado el valor del ángulo θ_i de la articulación 41, de tal manera que se ha cambiado de 0 a $-p[3] + 90$.
- **Ángulo** θ_i : Se ha modificado el valor del ángulo θ_i de la articulación 42, de tal manera que se ha cambiado de 0 a $p[3] + 90$.
- **Ángulo** θ_i : Se ha modificado el valor del ángulo θ_i de la articulación ef, de tal manera que se ha cambiado de 0 a 90.
- **Ángulo** α_i : Se ha modificado el valor del ángulo α_i de la articulación ef, de tal manera que se ha cambiado de 0 a 0.
- **Distancia** a_i : Se ha modificado el valor de la distancia a_i de la articulación ef, de tal manera que se ha cambiado de 0 a 1.

Como se puede observar, dependiendo del manipulador que quiera ser implementado, se deben modificar los parámetros de la tabla Denavit Hartenberg de una forma u otra. Es decir, de manera previa, se deben conocer los parámetros de la tabla Denavit Hartenberg del manipulador que se quiere implementar, o en su defecto, como para el caso de la práctica implementada y su modificación, estos valores deben de ser calculados previamente.

2.6. Mejoras en la implementación

Tras la explicación del script de cinemática directa suministrado, se pueden tomar como mejoras de esta práctica pues el hecho de la visualización final del manipulador, es decir, se podría realizar la implementación de la visualización del manipulador en otros tipos de entornos gráficos, como por ejemplo, *Blender*. Es decir, se podría realizar la implementación de la visualización del manipulador en un entorno gráfico más realista, en el cual se podría observar el manipulador en un entorno tridimensional más realista, de tal manera que el alumnado pueda comprender de manera mucho más sencilla y visual el funcionamiento de la cinemática directa mediante Denavit Hartenberg.

Con todo esto comentado, se podría realizar la implementación de la práctica en python de la misma manera que hasta el momento, pero tras la explicación de la matriz de Denavit Hartenberg, se podría trasladar la representación gráfica de python mediante el empleo de la biblioteca *matplotlib* a un entorno gráfico más realista como *Blender*, haciendo uso de librerías como *bpy* [6].

2.7. Conclusiones

Para concluir con la práctica de Cinemática Directa mediante Denavit Hartenberg, se puede decir que se ha conseguido realizar la implementación de la cinemática directa de un robot manipulador mediante el empleo de la representación de Denavit Hartenberg. Además, se ha conseguido realizar la implementación de la visualización del robot manipulador en un entorno gráfico mediante el empleo de la biblioteca *matplotlib*. Pero, como persona a la que le gusta el mundo de la robótica, me hubiera gustado realizar la implementación de la visualización del robot manipulador en un entorno gráfico más realista como *Blender*, o poder ver su funcionamiento dentro de un manipulador real, ya sea como explicación de la práctica por parte del profesorado, pero, me faltó como esa emoción de poder ver el resultado de la práctica en la realidad.

Capítulo 3 Cinemática inversa

3.1. ¿Qué es la Cinemática Inversa?

Mientras que la cinemática directa se ocupa de calcular la posición y orientación del extremo del robot dado un conjunto de parámetros de articulación, la *cinemática inversa* implica encontrar las configuraciones de las articulaciones necesarias para alcanzar una posición y orientación del extremo. La cinemática inversa es un problema más difícil que la cinemática directa, ya que puede haber más de una solución posible, o ninguna. Además, la cinemática inversa puede ser más difícil de calcular que la cinemática directa, ya que puede requerir la resolución de una ecuación no lineal.

La cinemática inversa es esencial en la programación de robots ya que permite determinar cómo deben moverse las articulaciones para lograr una tarea específica.

3.2. Explicación del código suministrado

El script en python suministrado para la implementación de la práctica de cinemática inversa se puede observar a continuación:

Listing 3.1: Script de Cinemática Inversa

```
1 import sys
2 from math import *
3 import numpy as np
4 import matplotlib.pyplot as plt
5 import colorsys as cs
6
7 # *****
8 # Declaración de funciones
9
10 def muestra_origenes(0,final=0):
11     # Muestra los orígenes de coordenadas para cada articulación
12     print('Origenes de coordenadas:')
13     for i in range(len(0)):
14         print('(0'+str(i)+' )0\t= '+str([round(j,3) for j in 0[i]]))
15     if final:
16         print('E.Final = '+str([round(j,3) for j in final]))
17
18 def muestra_robot(0,obj):
19     # Muestra el robot graficamente
20     plt.figure()
21     plt.xlim(-L,L)
22     plt.ylim(-L,L)
23     T = [np.array(o).T.tolist() for o in 0]
24     for i in range(len(T)):
25         plt.plot(T[i][0], T[i][1], '-o', color=cs.hsv_to_rgb(i/float(len(T)),1,1))
26     plt.plot(obj[0], obj[1], '*')
27     plt.pause(0.0001)
28     plt.show()
29
```

```

30 # input()
31 plt.close()
32
33 def matriz_T(d,th,a,al):
34
35     return [[cos(th), -sin(th)*cos(al), sin(th)*sin(al), a*cos(th)]
36             , [sin(th), cos(th)*cos(al), -sin(al)*cos(th), a*sin(th)]
37             , [0, sin(al), cos(al), d]
38             , [0, 0, 0, 1]
39             ]
40
41 def cin_dir(th,a):
42     #Sea 'th' el vector de thetas
43     #Sea 'a' el vector de longitudes
44     T = np.identity(4)
45     o = [[0,0]]
46     for i in range(len(th)):
47         T = np.dot(T,matriz_T(0,th[i],a[i],0))
48         tmp=np.dot(T,[0,0,0,1])
49         o.append([tmp[0],tmp[1]])
50     return o
51
52 # *****
53 # Cálculo de la cinemática inversa de forma iterativa por el método CCD
54
55 # valores articulares arbitrarios para la cinemática directa inicial
56 th=[0.,0.,0.]
57 a =[5.,5.,5.]
58 L = sum(a) # variable para representación gráfica
59 EPSILON = .01
60
61 #plt.ion() # modo interactivo
62
63 # introducción del punto para la cinemática inversa
64 if len(sys.argv) != 3:
65     sys.exit("python " + sys.argv[0] + " x y")
66 objetivo=[float(i) for i in sys.argv[1:]]
67 O=cin_dir(th,a)
68 #O=zeros(len(th)+1) # Reservamos estructura en memoria
69 # Calculamos la posicion inicial
70 print ("- Posicion inicial:")
71 muestra_origenes(O)
72
73 dist = float("inf")
74 prev = 0.
75 iteracion = 1
76 while (dist > EPSILON and abs(prev-dist) > EPSILON/100.):
77     prev = dist
78     O=[cin_dir(th,a)]
79     # Para cada combinación de articulaciones:
80     for i in range(len(th)):
81         # cálculo de la cinemática inversa:
82
83         O.append(cin_dir(th,a))
84
85     dist = np.linalg.norm(np.subtract(objetivo,O[-1][-1]))

```

```

86 print ("\n- Iteracion " + str(iteracion) + ':')
87 muestra_origenes(0[-1])
88 muestra_robot(0,objetivo)
89 print ("Distancia al objetivo = " + str(round(dist,5)))
90 iteracion+=1
91 O[0]=O[-1]
92
93 if dist <= EPSILON:
94     print ("\n" + str(iteracion) + " iteraciones para converger.")
95 else:
96     print ("\nNo hay convergencia tras " + str(iteracion) + " iteraciones.")
97 print ("- Umbral de convergencia epsilon: " + str(EPSILON))
98 print ("- Distancia al objetivo: " + str(round(dist,5)))
99 print ("- Valores finales de las articulaciones:")
100 for i in range(len(th)):
101     print ("  theta" + str(i+1) + " = " + str(round(th[i],3)))
102 for i in range(len(th)):
103     print ("  L" + str(i+1) + " = " + str(round(a[i],3)))

```

Como se puede ver en el script adjunto anteriormente, se realiza la implementación de un script genérico que permite realizar el cálculo de la cinemática inversa de un robot mediante el método de Cierre de Cadena Cinemática (CCD). El método CCD es un algoritmo iterativo que comienza con una estimación inicial de la configuración de las articulaciones y luego mejora la estimación en cada iteración. El algoritmo CCD se puede resumir en los siguientes pasos:

1. Seleccionar un punto de destino para el extremo del robot.
2. Para cada articulación, comenzando con la articulación más alejada del extremo del robot, calcule el vector desde la articulación hasta el punto de destino.
3. Calcule el vector desde la articulación hasta el extremo del robot.
4. Calcule el ángulo entre estos dos vectores.
5. Gire la articulación por este ángulo.
6. Repita los pasos 2 a 5 hasta que el extremo del robot esté lo suficientemente cerca del punto de destino.
7. Repita los pasos 1 a 6 para cada punto de destino.
8. Repita los pasos 1 a 7 hasta que el robot alcance su posición final.
9. Repita los pasos 1 a 8 para cada posición final.
10. Repita los pasos 1 a 9 hasta que el robot complete su trayectoria.

Después de definir el algoritmo CCD, se determinan las funciones para mostrar orígenes de coordenadas y representar gráficamente el robot, el programa establece valores iniciales arbitrarios para las articulaciones y longitudes de eslabones. El usuario proporciona las coordenadas del punto objetivo, y utilizando la cinemática directa se calcula la posición inicial y se inicia un bucle iterativo para ajustar las articulaciones de manera que el extremo del robot se acerque al objetivo. En cada iteración, se muestra la posición actual, los orígenes de coordenadas y la distancia al objetivo. Las iteraciones continúan hasta que se alcanza un umbral de convergencia o se supera un límite predefinido de iteraciones.

Finalmente, se presentan los resultados, incluyendo la cantidad de iteraciones necesarias, el umbral de convergencia, la distancia al objetivo y los valores finales de las articulaciones.

3.3. Implementación de la Cinemática Inversa

Tras la explicación del funcionamiento del script de *cinemática inversa* proporcionado, se realiza la implementación de esta de la siguiente manera:

Listing 3.2: Script de Cinemática Inversa implementado de manera correcta

```
1 import colorsys as cs
2 import matplotlib.pyplot as plt
3 import sys
4 from math import *
5 import numpy as np
6 import matplotlib
7 matplotlib.use('TkAgg')
8
9 # *****
10 # Declaración de funciones
11
12 def muestra_origenes(0, final=0):
13     # Muestra los orígenes de coordenadas para cada articulación
14     print('Orígenes de coordenadas:')
15     for i in range(len(0)):
16         print('(0'+str(i)+'')0\t= '+str([round(j, 3) for j in 0[i]]))
17     if final:
18         print('E.Final = '+str([round(j, 3) for j in final]))
19
20
21 def muestra_robot(0, obj):
22     # Muestra el robot graficamente
23     plt.figure(1)
24     plt.xlim(-L, L)
25     plt.ylim(-L, L)
26     T = [np.array(o).T.tolist() for o in 0]
27     for i in range(len(T)):
28         plt.plot(T[i][0], T[i][1], '-o',
29                 color=cs.hsv_to_rgb(i/float(len(T)), 1, 1))
30     plt.plot(obj[0], obj[1], '*')
31     plt.show()
32     input("Continuar...")
33     plt.close()
34
35 def matriz_T(d, th, a, al):
36     # Calcula la matriz T (ángulos de entrada en grados)
37
38     return [[cos(th), -sin(th)*cos(al), sin(th)*sin(al), a*cos(th)], [sin(th),
39         cos(th)*cos(al), -sin(al)*cos(th), a*sin(th)], [0, sin(al),
40         cos(al), d], [0, 0, 0, 1]
41
42 ]
43
44 def cin_dir(th, a):
45     # Sea 'th' el vector de thetas
46     # Sea 'a' el vector de longitudes
47     T = np.identity(4)
48     o = [[0, 0]]
49     for i in range(len(th)):
50         T = np.dot(T, matriz_T(0, th[i], a[i], 0))
```

```

48     tmp = np.dot(T, [0, 0, 0, 1])
49     o.append([tmp[0], tmp[1]])
50     return o
51
52 def read_input_file(file_name):
53     with open(file_name, 'r') as f:
54         lines = f.readlines()
55         ## Comprobación de errores en la estructura del fichero introducido
56         if len(lines) != 5:
57             sys.exit("El fichero de entrada debe de tener 5 líneas.")
58         type_arm = [int(i) for i in lines[0].split()]
59         th = [float(i) for i in lines[1].split()]
60         a = [float(i) for i in lines[2].split()]
61         upper_limit = [float(i) for i in lines[3].split()]
62         lower_limit = [float(i) for i in lines[4].split()]
63         return type_arm, th, a, upper_limit, lower_limit
64
65 # *****
66 # Cálculo de la cinemática inversa de forma iterativa por el método CCD
67 plt.ion() # modo interactivo
68
69 # introducción del punto para la cinemática inversa
70 if sys.argv[1] == "-h" or sys.argv[1] == "--help":
71     print("<< Bienvenido al programa de ayuda de cinemática inversa >>")
72     print("Para ejecutar el programa, debe de seguir la siguiente estructura de ejecución:")
73     print("python ccdp3.py << FileName >> << x >> << y >>")
74     print("Donde FileName es el nombre del fichero de entrada")
75     print("y x e y son las coordenadas del punto al que se quiere llegar")
76     print("El fichero de entrada debe de tener la siguiente estructura (CADA VALOR SEPARADO POR ESPACIOS):")
77     print("<< Tipo de articulación >> (0 para rotacional, 1 para prismática)")
78     print("<< Theta >>")
79     print("<< a, Longitud >>")
80     print("<< Límite superior >>")
81     print("<< Límite inferior >>")
82     print("Ejemplo:")
83     print("=====")
84     print("0 0 1 1")
85     print("10. 20. 0. 0.")
86     print("1. 1. 1. 1.")
87     print("90 90 0 0")
88     print("-90 -90 0 0")
89     print("=====")
90     sys.exit()
91
92 if len(sys.argv) != 4:
93     sys.exit("python " + sys.argv[0] + " FileName " + " x y")
94 objetivo = [float(i) for i in sys.argv[2:]]
95
96 # Lectura del fichero de entrada
97 file_name = sys.argv[1]
98 type_arm, th, a, Upper_limit, Lower_limit = read_input_file(file_name)
99 L = sum(a)
100 EPSILON = .01 # umbral de convergencia
101

```

```

102 0 = list(range(len(th)+1)) # Reservamos estructura en memoria
103 0[0] = cin_dir(th, a) # Calculamos la posicion inicial
104 print("- Posicion inicial:")
105 muestra_origenes(0[0])
106
107 dist = float("inf")
108 prev = 0.
109 iteracion = 1
110 while (dist > EPSILON and abs(prev-dist) > EPSILON/100.):
111     prev = dist
112     # Para cada combinación de articulaciones:
113     for i in range(len(th)):
114         # cálculo de la cinemática inversa:
115         current = len(th)-i-1
116         if (type_arm[current] == 1): # Si es prismática
117             w = np.sum(th[:current+1]) # Suma acumulativa de los ángulos anteriores
118             u = [np.cos(w), np.sin(w)] # Vector unitario u
119             d = np.dot(u, np.subtract(objetivo, 0[i][-1])) # Distancia a recorrer
120             a[current] += d # Se suma la distancia a recorrer
121             # Se realiza una normalización de la longitud
122             if (a[current] > Upper_limit[current]):
123                 a[current] = Upper_limit[current]
124             if (a[current] < Lower_limit[current]):
125                 a[current] = Lower_limit[current]
126         else: # Si es rotacional
127             v1 = np.subtract(objetivo, 0[i][current]) # Vector objetivo
128             v2 = np.subtract(0[i][-1], 0[i][current]) # Vector actual
129             c1 = atan2(v1[1], v1[0]) # Ángulo asociado al vector v1
130             c2 = atan2(v2[1], v2[0]) # Ángulo asociado al vector v2
131             th[current] += c1 - c2 # Ajuste de ángulo para acercarse al objetivo
132             # Se realiza una normalización del ángulo
133             while th[current] > pi:
134                 th[current] -= 2*pi
135             while th[current] < -pi:
136                 th[current] += 2*pi
137             # Se realiza una normalización del ángulo
138             if (th[current] > Upper_limit[current]):
139                 th[current] = Upper_limit[current]
140             if (th[current] < Lower_limit[current]):
141                 th[current] = Lower_limit[current]
142             0[i+1] = cin_dir(th, a)
143
144     dist = np.linalg.norm(np.subtract(objetivo, 0[-1][-1]))
145     print("\n- Iteracion " + str(iteracion) + ':')
146     muestra_origenes(0[-1])
147     muestra_robot(0, objetivo)
148     print("Distancia al objetivo = " + str(round(dist, 5)))
149     iteracion += 1
150     0[0] = 0[-1]
151
152 if dist <= EPSILON:
153     print("\n" + str(iteracion) + " iteraciones para converger.")
154 else:
155     print("\nNo hay convergencia tras " + str(iteracion) + " iteraciones.")
156 print("- Umbral de convergencia epsilon: " + str(EPSILON))
157 print("- Distancia al objetivo: " + str(round(dist, 5)))

```

```

157 print("- Valores finales de las articulaciones:")
158 for i in range(len(th)):
159     print("  theta" + str(i+1) + " = " + str(round(th[i], 3)))
160 for i in range(len(th)):
161     print("  L" + str(i+1) + " = " + str(round(a[i], 3)))
162
163 plt.show(block=True)

```

Para comenzar con los cambios realizados dentro del script que se puede observar anteriormente, se realiza la implementación de la cinemática inversa donde se presenta un bucle iterativo basado en el método de Cierre de Cadena Cinemática (CCD) para calcular la cinemática inversa. El bucle continúa hasta que la distancia entre la posición actual del extremo del robot y el objetivo sea menor que un umbral de convergencia (EPSILON). Durante cada iteración, se ajustan las articulaciones del robot para acercar el extremo al objetivo.

El proceso iterativo incluye las siguientes etapas:

1. Se almacena la distancia actual como referencia (prev) para evaluar la convergencia.
2. Para cada articulación, se calcula la cinemática inversa, tomando en cuenta si la articulación es prismática o rotacional.
 - a) En el caso de una articulación prismática, se ajusta la longitud del eslabón y se normaliza si excede los límites.
 - b) Para articulaciones rotacionales, se determina el ángulo de ajuste necesario y se normaliza si es necesario.
3. Se actualiza la posición del robot después de cada ajuste.

Tras la implementación de la cinemática inversa, se procede a la inclusión de una función nueva dentro del script, denominada como *read_input_file*, la cual permite la lectura de un fichero de texto, en el cual se encuentran los valores de los distintos parámetros como el tipo de brazo, sus ángulos, los límites superiores, los límites inferiores. Un ejemplo de este fichero de texto se puede observar a continuación:

Listing 3.3: Ejemplo de fichero de entrada

```

1 0 1 0
2 50. 0. 20.
3 5. 5. 5.
4 90 0 90
5 -90 0 -90

```

Finalmente como último cambio realizado al script de cinemática inversa, se realiza la implementación de un *menú de ayuda* que permite al usuario conocer los parámetros que se deben introducir para la ejecución del script, junto con la forma de ejecutar el script.

3.4. Ejecución del código

Tras la explicación de la implementación de la cinemática inversa, se procede a la ejecución del script para la comprobación del funcionamiento del mismo. Para ello, se ejecuta el siguiente comando en la terminal:

Listing 3.4: Ejecución del script de Cinemática Inversa

```

1 $ python3 Inverse-Kinematics-result.py input-file.txt 15 -10

```

Tras esto, se obtiene el siguiente resultado:

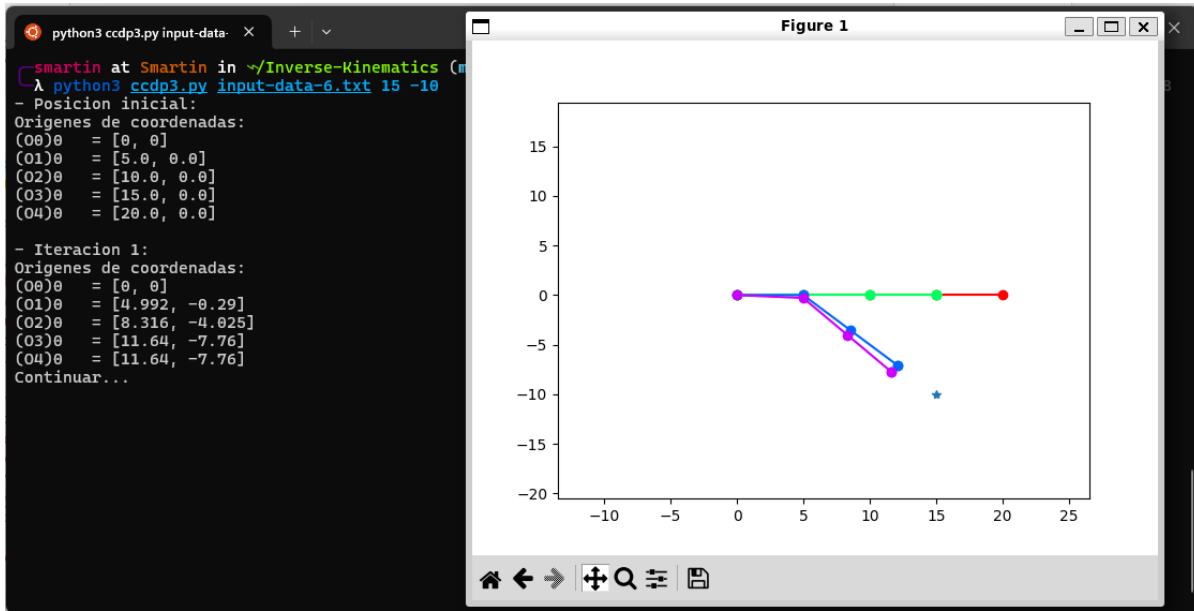


Figura 3.1: Resultado de la ejecución del script de Cinemática Inversa

3.5. Parámetros modificados

Para la implementación de cinemática inversa, se han realizado numerosas pruebas con los parámetros que existen dentro de los ficheros de entrada del programa, es por ello que se han realizado numerosas pruebas con los cambios que se producirían si por ejemplo si se realizan cambios en el tipo de articulación que existe en el brazo, es decir, 0 si se trata de una articulación rotacional y 1 si se trata de una articulación prismática.

Además, se han realizado pruebas con los cambios que se producirían si se modifican los valores en los *límites superiores* e *inferiores* de las articulaciones, es decir, si se cambian los valores de los límites superiores e inferiores de las articulaciones, se producen cambios en el resultado final de la cinemática inversa. Por último, se han realizado pruebas con los ángulos de las articulaciones, es decir, con los valores de *theta* o finalmente con la *longitud* de los distintos eslabones que componen el brazo.

3.6. Conclusiones

Para concluir, se puede decir que se ha conseguido realizar la implementación de la cinemática inversa de un robot manipulador mediante el empleo del método de Cierre de Cadena Cinemática (CCD). Esta práctica me ha resultado personalmente más interesante en cuanto a su desarrollo y en cuanto a su ejecución, ya que el hecho de poder ver como dependiendo de los valores que se le introduzcan a los distintos eslabones del brazo, se pueda alcanzar el objetivo final de manera más rápida o más lenta dependiendo de los valores que se le introduzcan, me ha resultado muy interesante. Además, cabe destacar que en cuanto a esta práctica no se me ha ocurrido ninguna mejora, ya que la forma de ver como se desplaza el brazo mediante la librería de *matplotlib*, me parece que de esta manera es mucho más cómoda y sencilla de entender y comprender el funcionamiento de la cinemática inversa.

Capítulo 4 Localización

- 4.1. ¿Qué es la localización?
- 4.2. Explicación del código suministrado
- 4.3. Implementación de la localización
- 4.4. Ejecución del código
- 4.5. Parámetros modificados
- 4.6. Mejoras en la implementación
- 4.7. Conclusiones

Capítulo 5 Filtro de partículas

5.1. ¿Qué es un filtro de partículas?

Capítulo 6 Conclusiones

Bibliografía

- [1] UdeSantiagoVirtual. [2023]. Herramientas matemáticas. Universidad de Santiago de Chile. Recuperado de <http://www.udesantiagoovirtual.cl/moodle2/mod/book/view.php?id=24916>
- [2] Samuel Martín Morales. [2023]. Direct-Kinematics-using-Denavit-Hartenberg. GitHub. Recuperado de <https://github.com/Samuelmm15/Direct-Kinematics-using-Denavit-Hartenberg.git>
- [3] Samuel Martín Morales. [2023]. Inverse-Kinematics. GitHub. Recuperado de <https://github.com/Samuelmm15/Inverse-Kinematics.git>.
- [4] Samuel Martín Morales. [2023]. Robot-Localization. GitHub. Recuperado de <https://github.com/Samuelmm15/Robot-Localization.git>.
- [5] Universidad de La Laguna. [2023]. Cinemática Directa mediante Denavit-Hartenberg. Universidad de La Laguna. https://drive.google.com/file/d/1639F9ebw14zquNaZcDf6o_u-_Tpaksh4/view?usp=sharing
- [6] Comunidad Python. [2023]. bpy - Blender Python. Comunidad Python. <https://pypi.org/project/bpy/>
- [7] Escamilla Losoyo José de Jesús. [2020]. Cinemática inversa. Studocu. <https://www.studocu.com/es-mx/document/instituto-tecnologico-de-leon/robotica/cinematica-inversa/12281135>