# Robótica Computacional

Samuel Martín Morales alu0101359526@ull.edu.es

22 de diciembre de 2023

# Índice general

# 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** $\theta_i$: 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** $\alpha_i$: Ángulo entre los ejes $Z_{i-1}$ y $Z_i$ medido alrededor del eje $X_i$.
- **Ángulo** $\theta_i$: 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

```
# Ejemplo:
# ./cdDH.py 30 45

import sys
from math import *
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# ***************************************************************************
```

```
# Declaración de funciones

def ramal(I,prev=[],base=0):
  # Convierte el robot a una secuencia de puntos para representar
  O = []
  if I:
    if isinstance(I[0][0],list):
      for j in range(len(I[0])):
        O.extend(ramal(I[0][j], prev, base or j < len(I[0])-1))
    else:
      O = [I[0]]
      O.extend(ramal(I[1:],I[0],base))
      if base:
        O.append(prev)
  return O

def muestra_robot(O,ef=[]):
  # Pinta en 3D
  OR = ramal(O)
  OT = np.array(OR).T
  fig = plt.figure()
  ax = fig.add_subplot(111, projection='3d')
  # Bounding box cúbico para simular el ratio de aspecto correcto
  max_range = np.array([OT[0].max()-OT[0].min()
                        ,OT[1].max()-OT[1].min()
                        ,OT[2].max()-OT[2].min()
                        ]).max()
  Xb = (0.5*max_range*np.mgrid[-1:2:2,-1:2:2,-1:2:2][0].flatten()
     + 0.5*(OT[0].max()+OT[0].min()))
  Yb = (0.5*max_range*np.mgrid[-1:2:2,-1:2:2,-1:2:2][1].flatten()
     + 0.5*(OT[1].max()+OT[1].min()))
  Zb = (0.5*max_range*np.mgrid[-1:2:2,-1:2:2,-1:2:2][2].flatten()
     + 0.5*(OT[2].max()+OT[2].min()))
  for xb, yb, zb in zip(Xb, Yb, Zb):
     ax.plot([xb], [yb], [zb], 'w')
  ax.plot3D(OT[0],OT[1],OT[2],marker='s')
  ax.plot3D([0],[0],[0],marker='o',color='k',ms=10)
  if not ef:
    ef = OR[-1]
  ax.plot3D([ef[0]],[ef[1]],[ef[2]],marker='s',color='r')
  ax.set_xlabel('X')
  ax.set_ylabel('Y')
  ax.set_zlabel('Z')
  plt.show()
  return

def arbol_origenes(O,base=0,sufijo=''):
  # Da formato a los origenes de coordenadas para mostrarlos por pantalla
  if isinstance(O[0],list):
    for i in range(len(O)):
      if isinstance(O[i][0],list):
        for j in range(len(O[i])):
          arbol_origenes(O[i][j],i+base,sufijo+str(j+1))
      else:
        print('(O'+str(i+base)+sufijo+')0\t= '+str([round(j,3) for j in O[i]]))
  else:
```

```
    print('(O'+str(base)+sufijo+')0\t= '+str([round(j,3) for j in O]))

def muestra_origenes(O,final=0):
  # Muestra los orígenes de coordenadas para cada articulación
  print('Orígenes de coordenadas:')
  arbol_origenes(O)
  if final:
    print('E.Final = '+str([round(j,3) for j in final]))

def matriz_T(d,theta,a,alpha):
  # Calcula la matriz T (ángulos de entrada en grados)
  th=theta*pi/180;
  al=alpha*pi/180;
  return [[cos(th), -sin(th)*cos(al),  sin(th)*sin(al), a*cos(th)]
         ,[sin(th),  cos(th)*cos(al), -sin(al)*cos(th), a*sin(th)]
         ,[      0,          sin(al),          cos(al),         d]
         ,[      0,                0,                0,         1]
         ]
# ***************************************************************************


# Introducción de los valores de las articulaciones
nvar=2 # Número de variables
if len(sys.argv) != nvar+1:
  sys.exit('El número de articulaciones no es el correcto ('+str(nvar)+')')
p=[float(i) for i in sys.argv[1:nvar+1]]

# Parámetros D-H:
#        1    2
d  = [   0,   0]
th = [p[0],p[1]]
a  = [  10,   5]
al = [   0,   0]

# Orígenes para cada articulación
o00=[0,0,0,1]
o11=[0,0,0,1]
o22=[0,0,0,1]

# Cálculo matrices transformación
T01=matriz_T(d[0],th[0],a[0],al[0])
T12=matriz_T(d[1],th[1],a[1],al[1])
T02=np.dot(T01,T12)

# Transformación de cada articulación
o10 =np.dot(T01, o11).tolist()
o20 =np.dot(T02, o22).tolist()

# Mostrar resultado de la cinemática directa
muestra_origenes([o00,o10,o20])
muestra_robot   ([o00,o10,o20])
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

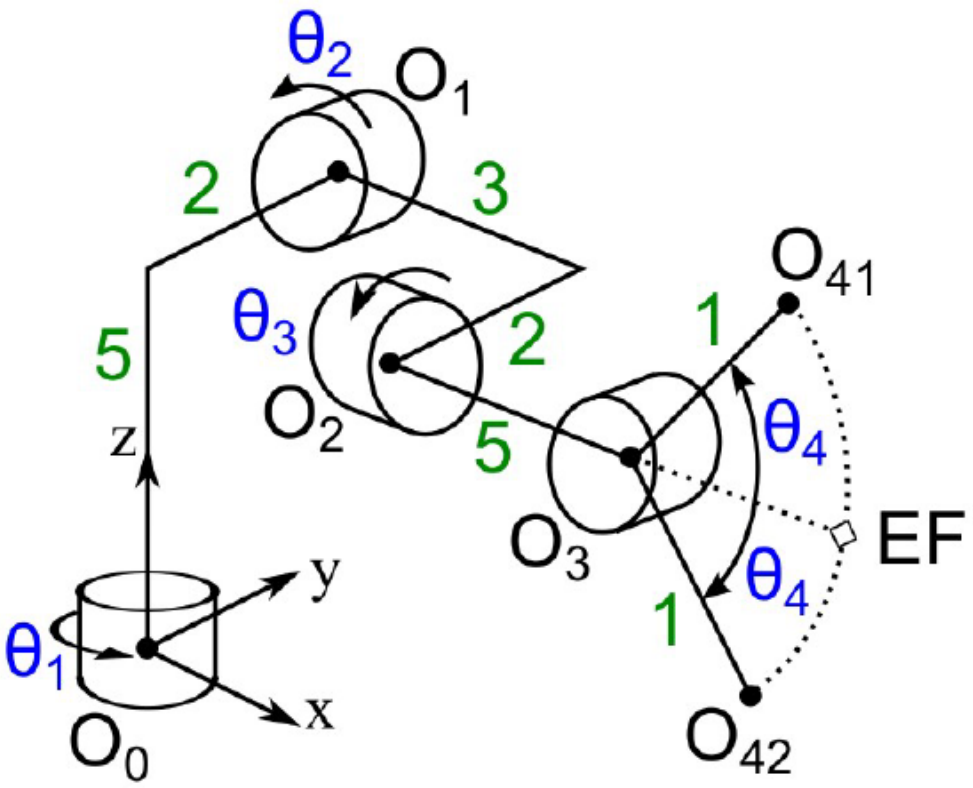


Figura 2.1: Representación gráfica del manipulador número 5

### 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** | $\mathbf{d_i}$ | $\theta_\mathbf{i}$ | $\mathbf{a_i}$ | $\alpha_\mathbf{i}$ |
|---|---|---|---|---|
| 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

```
# Ejemplo:
# python3 cin_dir_5.py 10 20 30 30

# Cabe destacar que la ejecución de este script tiene la siguiente estructura de
    ejecución:
# python3 cin_dir_5.py <ángulo-o0> <ángulo-o2> <ángulo-o3> <ángulo-51-52>

import sys
from math import pi, cos, sin
import numpy as np
import matplotlib
# De esta manera se establece el motor gráfico que se debe de usar para MacOS
matplotlib.use('TkAgg')
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# ******************************************************************************
# Declaración de funciones

def ramal(I,prev=[],base=0):
  # Convierte el robot a una secuencia de puntos para representar
  O = []
  if I:
    if isinstance(I[0][0],list):
      for j in range(len(I[0])):
        O.extend(ramal(I[0][j], prev, base or j < len(I[0])-1))
    else:
      O = [I[0]]
      O.extend(ramal(I[1:],I[0],base))
      if base:
        O.append(prev)
  return O
```

```
32
33 def muestra_robot(O,ef=[]):
34   # Pinta en 3D
35   OR = ramal(O)
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(O,base=0,sufijo=''):
64   # Da formato a los origenes de coordenadas para mostrarlos por pantalla
65   if isinstance(O[0],list):
66     for i in range(len(O)):
67       if isinstance(O[i][0],list):
68         for j in range(len(O[i])):
69           arbol_origenes(O[i][j],i+base,sufijo+str(j+1))
70       else:
71         print(('(O'+str(i+base)+sufijo+')0\t= '+str([round(j,3) for j in O[i]])))
72   else:
73     print(('(O'+str(base)+sufijo+')0\t= '+str([round(j,3) for j in O])))
74
75 def muestra_origenes(O,final=0):
76   # Muestra los orígenes de coordenadas para cada articulación
77   print('Orígenes de coordenadas:')
78   arbol_origenes(O)
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)]
```

```
          ,[      0,           sin(al),           cos(al),          d]
          ,[      0,                 0,                 0,          1]
          ]
# ***************************************************************************

plt.ion() # Modo interactivo
# Introducción de los valores de las articulaciones
nvar=4 # Número de variables
if len(sys.argv) != nvar+1:
  sys.exit('El número de articulaciones no es el correcto ('+str(nvar)+')')
p=[float(i) for i in sys.argv[1:nvar+1]]

# Parámetros D-H:
#        05      1      15          2       3          41          42          ef
d  = [   5,      2,     0,          -2,     5,         0,          0,          0
     ]
th = [   p[0],   0,     p[1],       -90,    p[2],      -p[3] + 90, p[3] + 90,  90
    ]
a  = [   0,      0,     3,          0,      0,         1,          1,          1
     ]
al = [   -90,    0,     0,          -90,    90,        0,          0,          0
     ]

# Orígenes para cada articulación
o00=[0,0,0,1]
o05=[0,0,0,1]
o11=[0,0,0,1]
o15=[0,0,0,1]
o22=[0,0,0,1]
o33=[0,0,0,1]
o441=[0,0,0,1]
o442=[0,0,0,1]
oeff=[0,0,0,1]

# Cálculo matrices transformación
T005 = matriz_T(d[0],th[0],a[0],al[0])
T051 = matriz_T(d[1],th[1],a[1],al[1])
T115 = matriz_T(d[2],th[2],a[2],al[2])
T225 = matriz_T(d[3],th[3],a[3],al[3])
T033 = matriz_T(d[4],th[4],a[4],al[4])
T041 = matriz_T(d[5],th[5],a[5],al[5])
T042 = matriz_T(d[6],th[6],a[6],al[6])
T0ef = matriz_T(d[7],th[7],a[7],al[7])


T02 = np.dot(T005, T051)
T03 = np.dot(T02, T115)
T04 = np.dot(T03, T225)
T05 = np.dot(T04, T033)
T06 = np.dot(T05, T041)
T07 = np.dot(T06, T042)
T0ef = np.dot(T07, T0ef)

# Transformación de cada articulación
o10 = np.dot(T005, o05).tolist()
o20 = np.dot(T02, o11).tolist()
```

```
o30 = np.dot(T03, o15).tolist()
o40 = np.dot(T04, o22).tolist()
o50 = np.dot(T05, o33).tolist()
o60 = np.dot(T06, o441).tolist()
o70 = np.dot(T07, o442).tolist()
oeff = np.dot(T0ef, oeff).tolist()

# Mostrar resultado de la cinemática directa
muestra_origenes([o00,o10, o20, o30, o40, o50, [[o60], [o70]]], oeff)
muestra_robot   ([o00,o10, o20, o30, o40, o50, [[o60], [o70]]], oeff)
eval(input())

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

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

Tras esto, se obtiene el siguiente resultado:

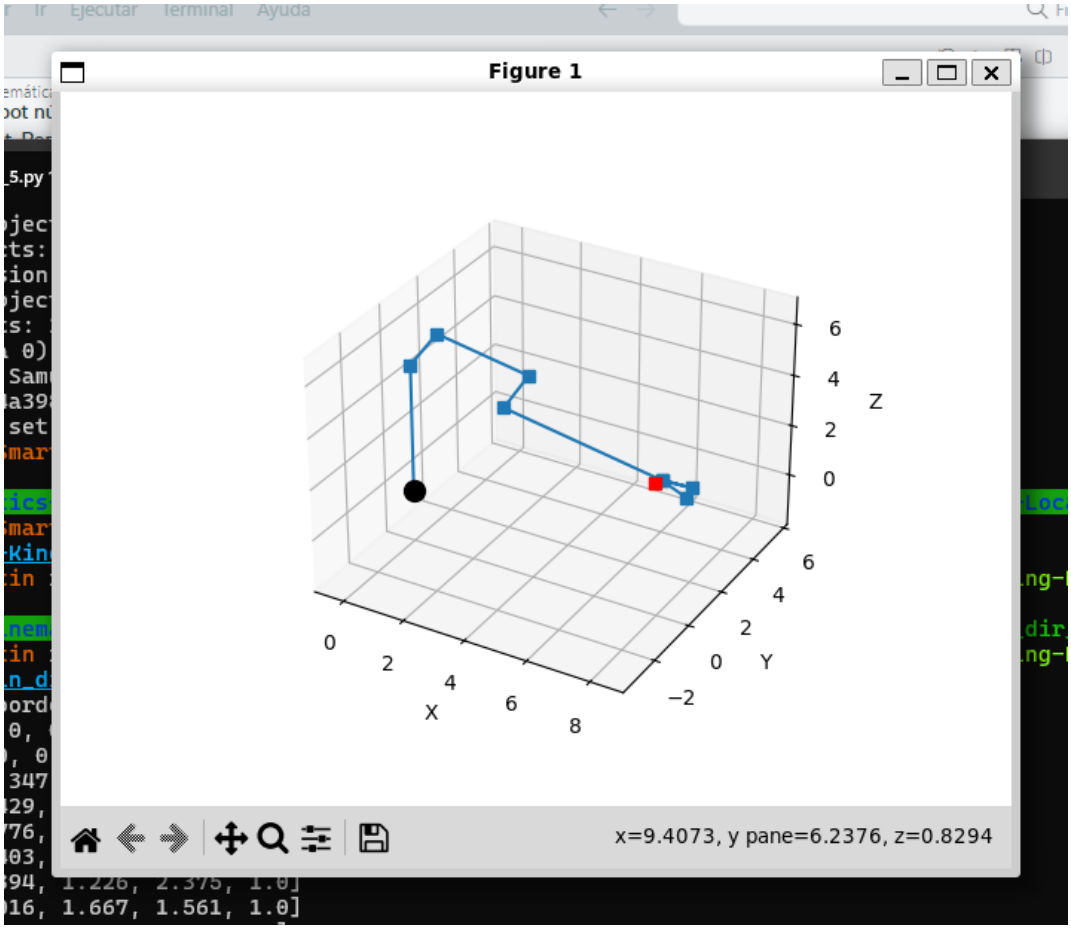


Figura 2.2: Resultado de la ejecución del script de Cinemática Directa del Manipulador 5

## 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** $\theta_i$: Se ha modificado el valor del ángulo $\theta_i$ de la articulación 05, de tal manera que se ha cambiado de 0 a $p[0]$.
- **Ángulo** $\alpha_i$: Se ha modificado el valor del ángulo $\alpha_i$ 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** $\theta_i$: Se ha modificado el valor del ángulo $\theta_i$ 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** $\alpha_i$: Se ha modificado el valor del ángulo $\alpha_i$ 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** $\theta_i$: Se ha modificado el valor del ángulo $\theta_i$ de la articulación 2, de tal manera que se ha cambiado de 0 a -90.
- **Ángulo** $\alpha_i$: Se ha modificado el valor del ángulo $\alpha_i$ 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** $\theta_i$: Se ha modificado el valor del ángulo $\theta_i$ de la articulación 3, de tal manera que se ha cambiado de 0 a $p[2]$.
- **Ángulo** $\alpha_i$: Se ha modificado el valor del ángulo $\alpha_i$ de la articulación 3, de tal manera que se ha cambiado de 0 a 90.
- **Ángulo** $\theta_i$: Se ha modificado el valor del ángulo $\theta_i$ de la articulación 41, de tal manera que se ha cambiado de 0 a $-p[3] + 90$.
- **Ángulo** $\theta_i$: Se ha modificado el valor del ángulo $\theta_i$ de la articulación 42, de tal manera que se ha cambiado de 0 a $p[3] + 90$.
- **Ángulo** $\theta_i$: Se ha modificado el valor del ángulo $\theta_i$ de la articulación ef, de tal manera que se ha cambiado de 0 a 90.
- **Ángulo** $\alpha_i$: Se ha modificado el valor del ángulo $\alpha_i$ 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?**

**3.2. Explicación del código suministrado**

**3.3. Implementación de la Cinemática Inversa**

**3.4. Ejecución del código**

**3.5. Parámetros modificados**

**3.6. Mejoras en la implementación**

**3.7. Conclusiones**

# Capítulo 4 Localización

# Capítulo 5 Filtro de partículas

## 5.1. ¿Qué es un filtro de partículas?

# Capítulo 6 Conclusiones

# Bibliografía


[1] UdeSantigoVirtual. [2023]. Herramientas matemáticas. Universidad de Santiago de Chile. Recuperado de http://www.udesantiagovirtual.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/