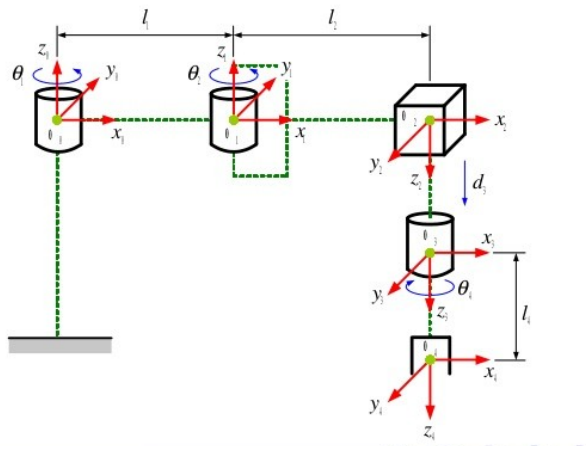
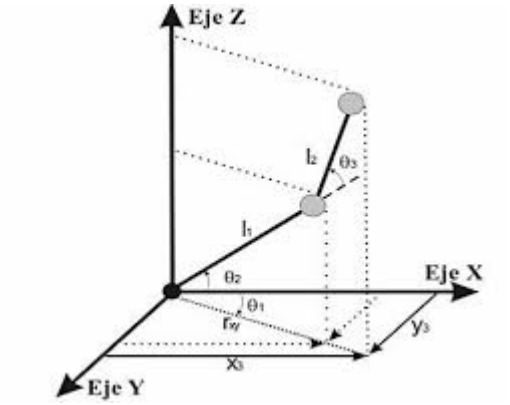
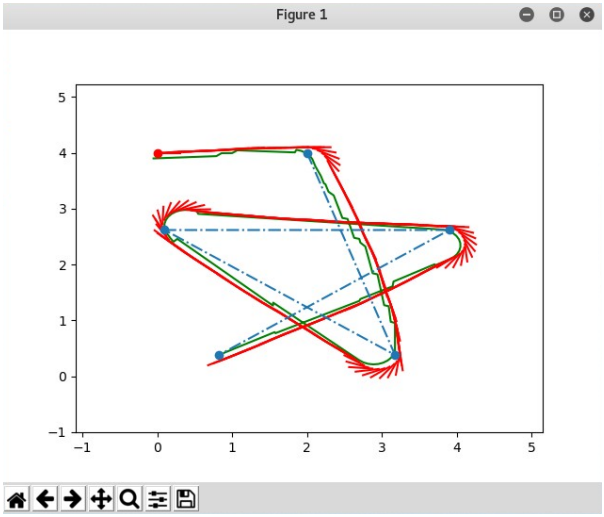
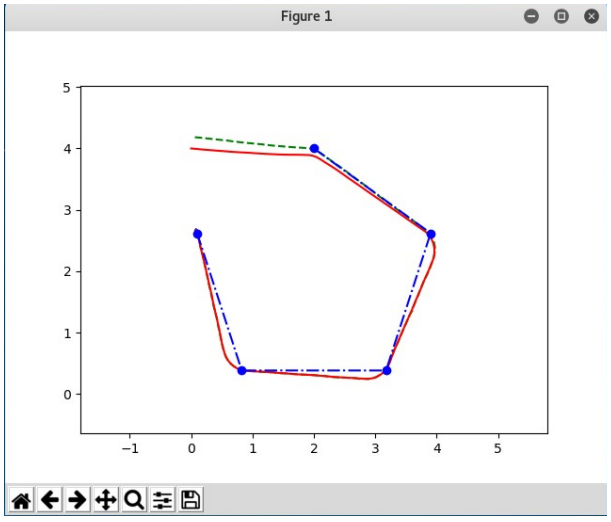
INFORME DE PRÁCTICAS

Robótica Computacional









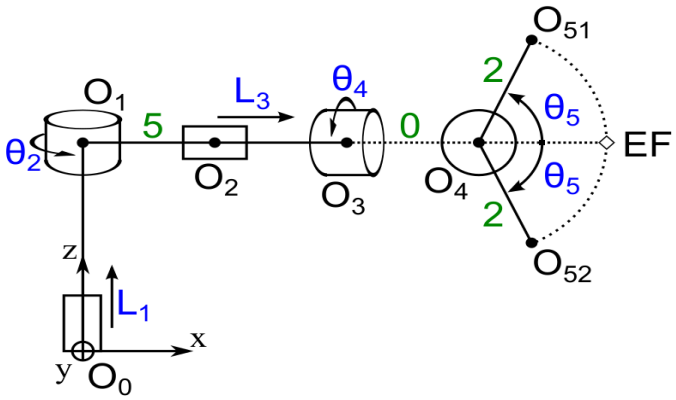
Kapil Ashok Melwani Chugani
alu0100883473@ull.edu.es
Universidad de La Laguna (ETSII)

Cinemática Directa

Trabajaremos con Sistemas Compatibles D – H donde los parámetros D – H vendrán dados según la siguiente tabla:

$$ {}^{i-1}T_i = \begin{bmatrix} \cos\theta_i & -\cos\alpha_i \cdot \sin\theta_i & \sin\alpha_i \cdot \sin\theta_i & a_i \cdot \cos\theta_i \\ \sin\theta_i & \cos\alpha_i \cdot \cos\theta_i & -\sin\alpha_i \cdot \cos\theta_i & a_i \cdot \sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

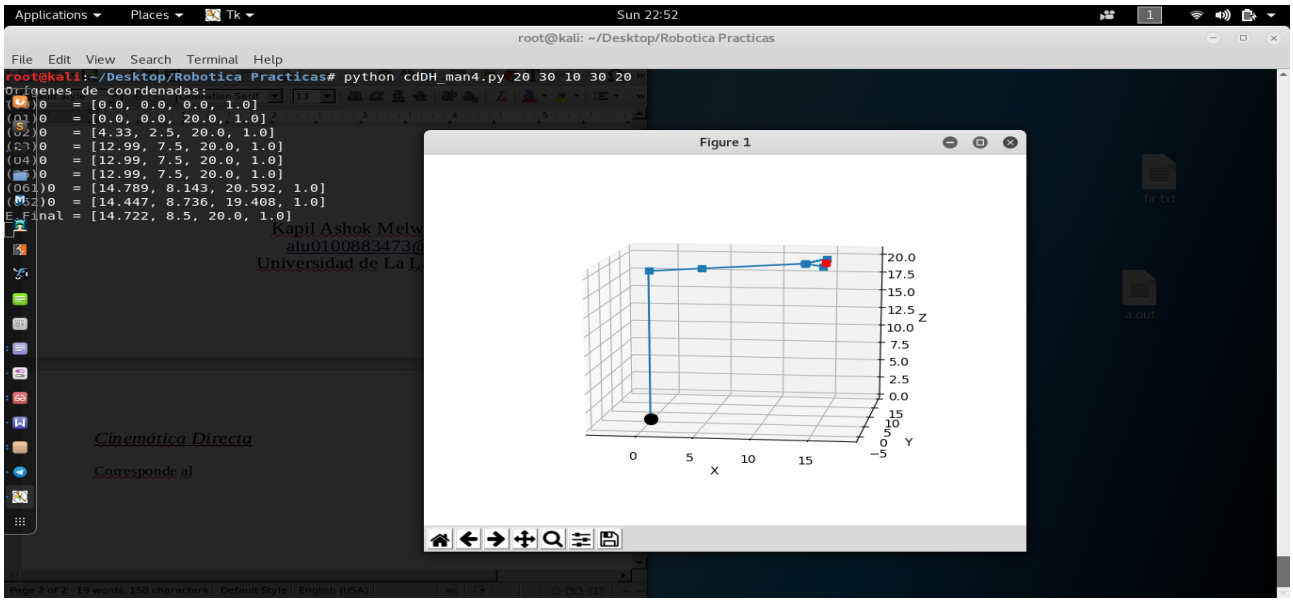
El robot con el que vamos a trabajar va a ser el siguiente:



Como se puede contemplar en la imagen anterior, tenemos el caso de un robot con 5 articulaciones cuyos parámetros D – H son los siguientes:

```
# Parâmetros D-H:
#        1     2       3       4   5   6   7   8
d  = [   p[0],  0,      0,      0,  0,  0,  0,  0]
th = [   0,     p[1],   0,      90, 90+p[3],    -90+p[4], -90-p[4], -90]
a  = [   0,     5,      p[2],   0,  0,  2,  2,  2]
al = [   0,     0,      0,      90, -90,0,  0,  0]
```

Como ya sabemos, encontramos las distancias (d, a) y los ángulos (th, al) correspondientes al siguiente robot:



Donde en este caso, los valores variables de la tabla D – H son:

p[0] = 20
p[1] = 30
p[2] = 10
p[3] = 30
p[4] = 20

Los cuales fueron especificados en la terminal a la hora de ejecutar el archivo.py:

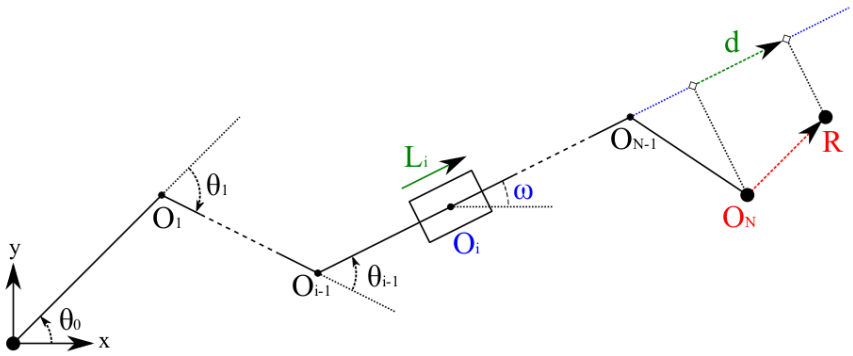
python cdDH_man4.py 20 30 10 30 20

Además, en el codigo deberemos especificar datos tales como:

```
# C@lculo matrices transformaci@n
T01=matriz_T(d[0],th[0],a[0],al[0])
T12=matriz_T(d[1],th[1],a[1],al[1])
T23=matriz_T(d[2],th[2],a[2],al[2])
T34=matriz_T(d[3],th[3],a[3],al[3])
T45=matriz_T(d[4],th[4],a[4],al[4])
T56=matriz_T(d[5],th[5],a[5],al[5])
T67=matriz_T(d[6],th[6],a[6],al[6])
T78=matriz_T(d[7],th[7],a[7],al[7])

T02=np.dot(T01,T12)
T03=np.dot(T02,T23)
T04=np.dot(T03,T34)
T05=np.dot(T04,T45)
T06=np.dot(T05,T56)
T07=np.dot(T05,T67)
T08=np.dot(T05,T78)

# Transformaci@n de cada articulaci@n
o10 =np.dot(T01, o11).tolist()
o20 =np.dot(T02, o22).tolist()
o30 =np.dot(T03, o33).tolist()
o40 =np.dot(T04, o44).tolist()
o50 =np.dot(T05, o55).tolist()
o60 =np.dot(T06, o66).tolist()
o70 =np.dot(T07, o77).tolist()
o80 =np.dot(T08, o88).tolist()

# Mostrar resultado de la cinem@tica directa
muestra_origenes([o00,o10,o20,o30,o40,o50,[[o60],[o70]]],o80)
muestra_robot   ([o00,o10,o20,o30,o40,o50,[[o60],[o70]]],o80)
input()
```

- Cálculo de las matrices de transformación.
- Transformación de cada articulación (especificamos los datos de la tabla D – H)
- Resultado de la cinemática Directa:
    - Hemos puesto juntos o60 y o70 ya que se tratan de las pinzas del robot, y por último o80 es el Efector Final

Cinemática Inversa



Queremos calcular la distancia que debe extenderse una articulación prismática situada en el punto Oi, de tal forma que el puto final del robot On se acrque tanto como sea posible a la posición objetivo R.

El acercamiento solo puede hacerse en la direccion de extensión Li, que podemos calcular como un ángulo w que define su rotacion respecto al eje x absoluto. Dicho angulo puede calcularse como:

$$\omega = \sum_{j=0}^{i} \theta_j$$

Usando el producto escalar podemos proyectar el vector que va de On hasta R sobre la dirección de extensión de la articulación, obteniendo así la distancia d:

$$\mathbf{d} = \begin{bmatrix} \cos(\boldsymbol{\omega}) \\ \sin(\boldsymbol{\omega}) \end{bmatrix} \cdot (\mathbf{R} - \mathbf{O_N})$$

Por tanto, el valor de Li tras cada iteración pasa a ser:

$$\mathbf{L_i} + \begin{bmatrix} \cos(\boldsymbol{\omega}) \\ \sin(\boldsymbol{\omega}) \end{bmatrix} \cdot (\mathbf{R} - \mathbf{O_N}), \qquad \text{con } \boldsymbol{\omega} = \sum_{j=0}^{i} \theta_j$$

En el código, para empezar crearemos un Limite Superior y un Limite Inferior para que nuestro robot no sobrepase dichos limites. Ademas, deberemos diferenciar entre articulaciones de rotación o prismática.

```
EPSILON = .01
TIPO =          [0,     1]
th =            [90,  -90]
a =             [3,     5]
LIMITE_SUP =    [90,   10]
LIMITE_INF =    [-90,   0]
L = sum(a) # variable para representaci□n gr□fica
```

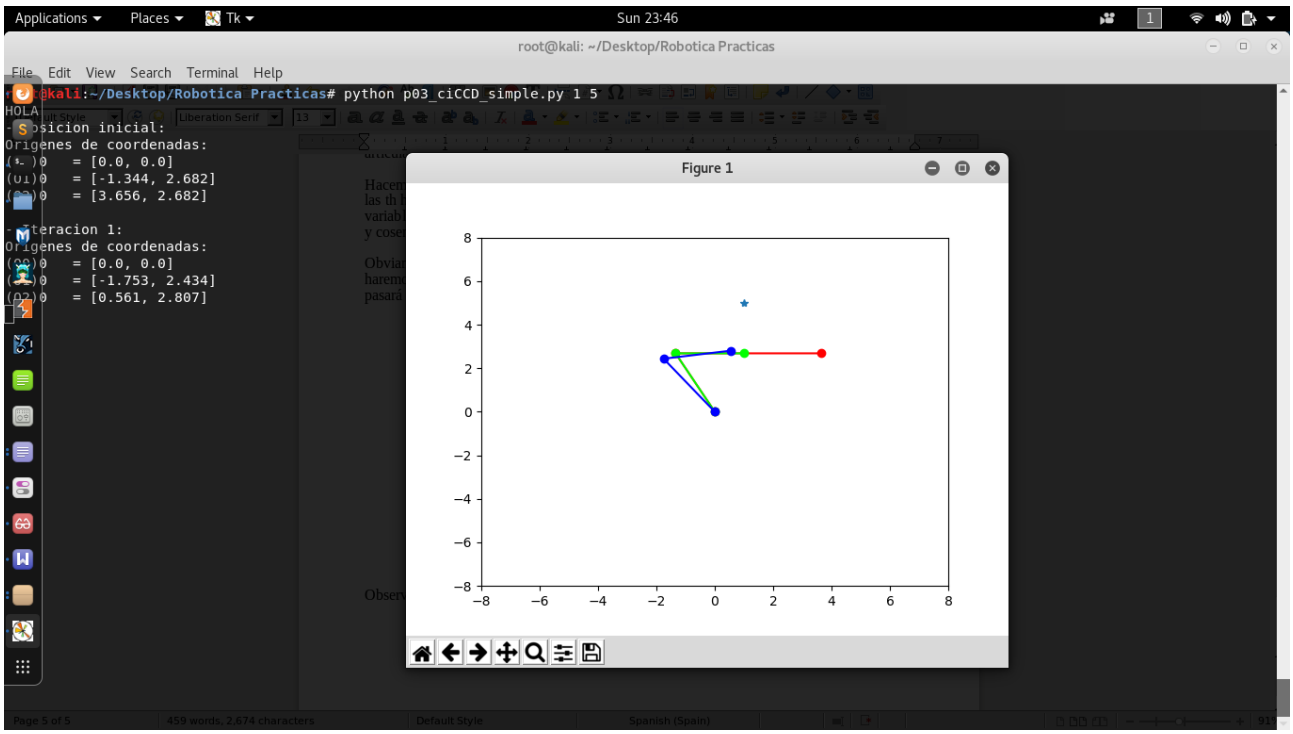
Ahora, dentro del bucle del codigo, una vez hayamos diferenciado entre los dos tipos de articulaciones:

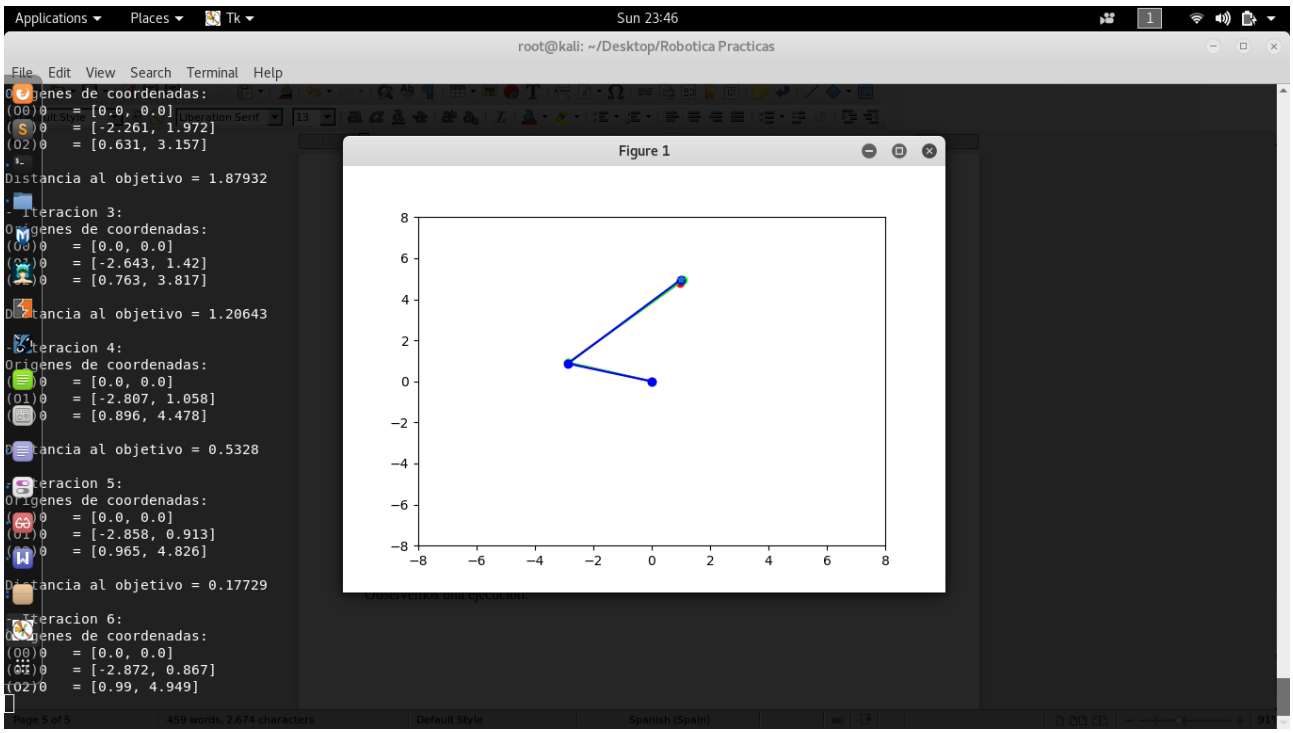
Hacemos el cálculo de w en el caso de articulación prismática el cual sera la suma de todas las th hasta la actual (inclusive). Posteriormente, calcularemos el seno y coseno de la variable w y por ultimo, realizaremos el calculo de “d”, que serña la multiplicacion del seno y coseno de la variable w por la distancia desde R (objetivo) hasta On.

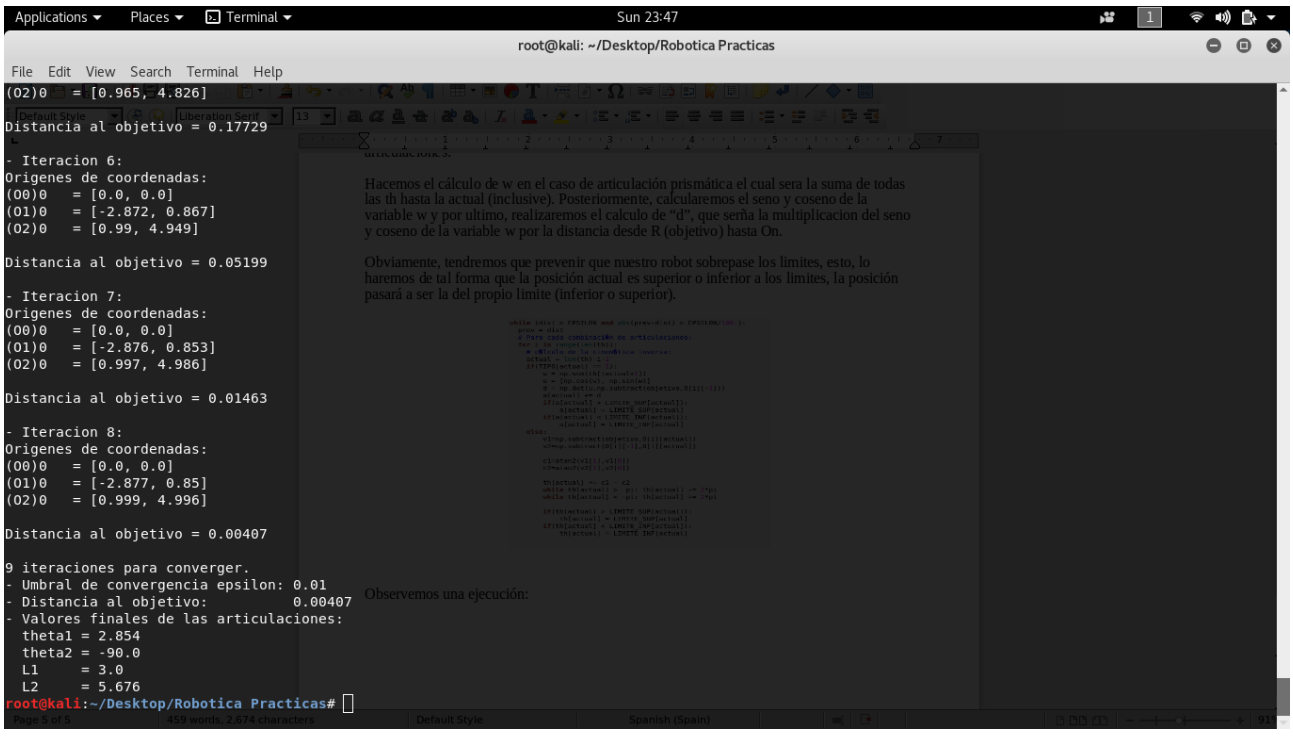
Obviamente, tendremos que prevenir que nuestro robot sobrepase los limites, esto, lo haremos de tal forma que la posición actual es superior o inferior a los limites, la posición pasará a ser la del propio limite (inferior o superior).

```
while (dist > EPSILON and abs(prev-dist) > EPSILON/100.):
  prev = dist
  # Para cada combinaci?n de articulaciones:
  for i in range(len(th)):
    # c?lculo de la cinem?tica inversa:
    actual = len(th)-i-1
    if(TIPO[actual] == 1):
        w = np.sum(th[:actual+1])
        u = [np.cos(w), np.sin(w)]
        d = np.dot(u,np.subtract(objetivo,O[i][-1]))
        a[actual] += d
        if(a[actual] > LIMITE_SUP[actual]):
            a[actual] = LIMITE_SUP[actual]
        if(a[actual] < LIMITE_INF[actual]):
            a[actual] = LIMITE_INF[actual]
    else:
        v1=np.subtract(objetivo,O[i][actual])
        v2=np.subtract(O[i][-1],O[i][actual])

        c1=atan2(v1[1],v1[0])
        c2=atan2(v2[1],v2[0])

        th[actual] += c1 - c2
        while th[actual] >  pi: th[actual] -= 2*pi
        while th[actual] < -pi: th[actual] += 2*pi

        if(th[actual] > LIMITE_SUP[actual]):
            th[actual] = LIMITE_SUP[actual]
        if(th[actual] < LIMITE_INF[actual]):
            th[actual] = LIMITE_INF[actual]
```

Observemos una ejecución: Nuestro robot tiene que desplazarse hasta el punto x = 1 y =5







articulación.

Hacemos el cálculo de w en el caso de articulación prismática el cual sera la suma de todas las th hasta la actual (inclusive). Posteriormente, calcularemos el seno y coseno de la variable w y por ultimo, realizaremos el calculo de "d", que seña la multiplicacion del seno y coseno de la variable w por la distancia desde R (objetivo) hasta On.

Obviamente, tendremos que prevenir que nuestro robot sobrepase los limites, esto, lo haremos de tal forma que la posición actual es superior o inferior a los limites, la posición pasará a ser la del propio limite (inferior o superior).

Observemos una ejecución:

Localización

Se considera un robot que se mueve en un mundo unidimesional donde se procederá a definir una rejilla de tal manera que la posicion vendr dada por un valor xi, que corresponde al valor de la rejilla donde está el robot. Suponemos que tendremos N rejillos para los cuales i=1,2 … N.

En cuanto al enfoque probabilista la posición vendrá dada por una densidad de probabilidad p(x). Si se considera una rejilla se tendra pi = p(xi) verificandose que el sumatorio de todas las probabilidades tiene que ser igual a 1.

En el .zip que se nos ofrece en el campus virtual hemos tenido que realizar una serie de cambios para que la trayectoria que siga el robot sea mas precisa, puesto que tenemos problemas en los casos en los que la probabilidad está debajo de un determinado umbral.

Se han clasificado los robots si son holonómicos o no, es una distincion relacionada con su movilidad. Podemos decir que robot o sistemas holonómicos son capaces de modificar su dirección instantaneamente y sin necesidad de rotar previamente.

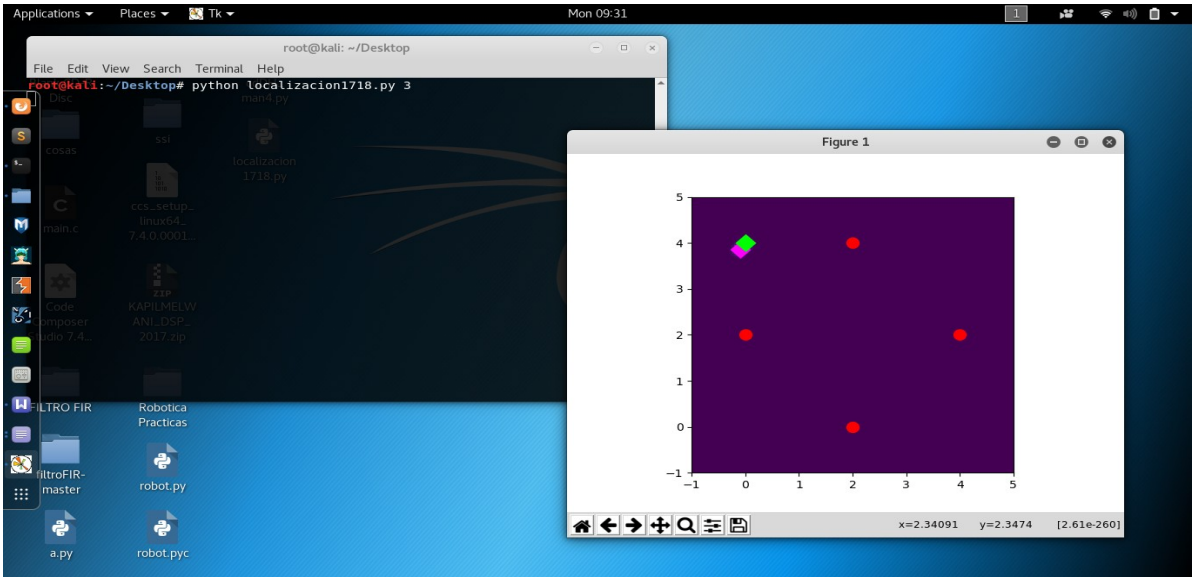
Por tanto, en un sistema holonómico, en el caso del GIROPARADO con un angulo cuyo valor absoluto sea mayor que .01, calculamos la distancia del robot real hasta el objetivo, posteriormente, con esa medida, calculamos la probabilidad.

En el caso de que la probabilidad sea menor de 0.8, volveremos a recalcular la posición del robot para que este no se desplace demasiado de la trayectoria ideal.

```
for punto in objetivos:
  while distancia(tray_ideal[-1],punto) > EPSILON and len(tray_ideal) <= 1000:
    pose = ideal.pose()

    w = angulo_rel(pose,punto)
    if w > W:  w =  W
    if w < -W: w = -W
    v = distancia(pose,punto)
    if (v > V): v = V
    if (v < 0): v = 0

    if HOLONOMICO:
      if GIROPARADO and abs(w) > .01:
        v = 0
      ideal.move(w,v)
      real.move(w,v)
      //medidas1 = real.sense(objetivos)
      //prob1 = ideal.measurement_prob(medidas1,objetivos);
      //if(prob1 < 0.8):
        //localizacion(objetivos,real,ideal,ideal.pose(),real.sense_noise,mostrar=0)
    else:
      ideal.move_triciclo(w,v,LONGITUD)
      real.move_triciclo(w,v,LONGITUD)
    tray_ideal.append(ideal.pose())
    tray_real.append(real.pose())

    espacio += v
    tiempo  += 1
```

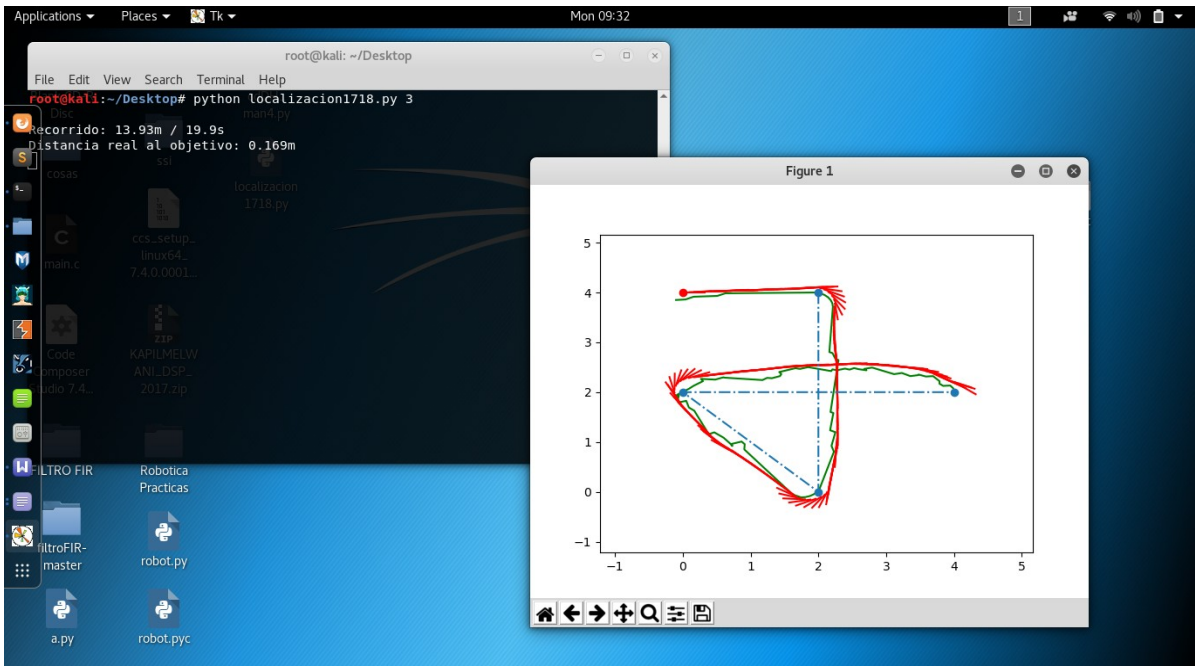
En este caso utilizaremos el escenario 3 (#python localizacion1718.py 3) donde en primer lugar vemos el estado inicial del robot.

Consideraremos que no tenemos información de la localizacion del robot. La densidad de probabilidad que denominamos “a priori” será uniforme sobre todas las posibles localizaciones.



Si pulsamos “enter” sobre la terminal, podremos ver la ejecución de la trayectoria que toma el robot en comparativa con la trayectoria ideal que deberia tomar.

En la siguiente imagen podemos contemplar algunos momentos de la ejecución en las que posiblemente la probabilidad sea menor de 0.8 y hayamos tenido que recalcular la trayectoria real del mismo.

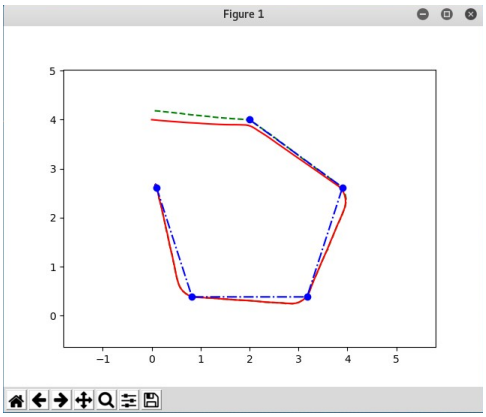


Obteniendo un resultado bastante optimo, ya que la distancia real al objetivo establecida está a 0.169m, lo cual es un resultado bastante bueno.

Filtrado de Partículas

En el filtrado de particulas debemos distinguir tres fases diferentes:

- Movimiento
- Pesado
- Remuestreo



En la primera movemos todo el filtro de particulas (el filtro de particulas es el conjunto de robots reales con diferentes ruidos).

En la fase de pesado hallaremos la probabilidad de aquel que se parezca mas a la trayectoria que tiene que seguir con las balizas

Y por último, en la fase de remuestreo, la cual es opcional puesto que se realiza nada mas cuando vemos que las partículas están muy dispersas y pueden acarrear errores, lo que hacemos es eliminar aquellas particulas que están muy disperar a la posicion del robot ideal y creamos nuevas particulas, de tal manera que, queden todas más agrupadas y podamos tener una mejor medida

```
140
141 tiempo  = 0.
142 espacio = 0.
143 for punto in objetivos:
144   while distancia(trayectoria[-1],punto) > EPSILON and len(trayectoria) <= 1000:
145     #seleccionar pose
146     w = angulo_rel(trayectoria[-1],punto)
147     if w > W:  w =  W
148     if w < -W: w = -W
149     v = distancia(trayectoria[-1],punto)
150     if (v > V): v = V
151     if (v < 0): v = 0
152     if HOLONOMICO:
153       if GIROPARADO and abs(w) > .01:
154           v = 0 # Mover todas las partículas aquí
155       real.move(w,v)
156       measurement = real.sense(objetivos)
157       for i in range(len(filtro)):
158           filtro[i].move(w, v)
159           filtro[i].measurement_prob(measurement, objetivos)
160     else:
161       real.move_triciclo(w,v,LONGITUD)
162       measurement = real.sense(objetivos)
163       for i in range(len(filtro)):
164           filtro[i].move_triciclo(w, v, LONGITUD)
165           filtro[i].measurement_prob(measurement, objetivos)
166     # Seleccionar hipótesis de localización y actualizar la trayectoria
167     trayectoria.append(hipotesis(filtro))
168     trayectreal.append(real.pose())
169     if peso_medio(filtro) < 0.1:
170       filtro = resample(filtro, N_PARTIC)
171
172     #pesos nuevos del filtro y añadir la hipótesis a la trayectoria.
173
174     trayectreal.append(real.pose())
175     mostrar(objetivos,trayectoria,trayectreal,filtro)
176
177     # remuestreo
```

En la siguiente captura del código, podemos ver la distinción de movimientos, lo que hacemos es mover el robot real y posteriormente todo el filtro, ademas, calculamos el peso por tanto hacemos la fase de movimiento y pesado.

Por último, donde comprobamos si el peso medio del filtro es menor que 0.1 quiere decir que comprobamos si tenemos que realizar la fase de remuestreo