SEGUNDA PRÁCTICA

SEGUIMIENTO DE CAMINOS

MIGUEL IAN GARCÍA POZO

GIERM

ESCUELA DE INGENIERÍAS INDUSTRIALES

UNIVERSIDAD DE MÁLAGA

Índice

Navegación Punto a Punto

- Desarrollo

En este ejercicio vamos a simular un robot diferencial siguiendo una ruta definida por puntos. Para ello, usaremos el modelo que hicimos en la práctica 1. La ruta que vamos a usar va a ser la definida en el ejercicio por los puntos:

Coordenada x (m)	Coordenada y (m)
0	0
20	0
20	20
-10	30
-20	-10
0	-30
0	0

Figura 1.- Puntos de la ruta

El ejercicio también nos indica que la velocidad va a ser de $1,2\ m/s$, que la posición del robot nos la dará el GPS diferencial cada $0,3\ s$ y que el robot cambiará de objetivo cuando esté a $1\ m$ de su objetivo anterior.
Considerando todas estas condiciones, ahora vamos a crear el código. Para empezar, vamos a definir las variables necesarias en la Figura 2.

```
x = 0; % Posición en x
y = 0; % Posición en y
o=0; % Orientación
xgps = 0; % Posición en x según el gps
ygps = 0; % Posición en y según el gps
ogps = 0; % Orientación según el gps
ruedas = 0.8; % Distancia entre las ruedas (m)
r = 0.1; % Radio de las ruedas (m)
dt = 0.025; % Tiempo de muestreo (s)
sgps = 0; % Variable que calcula cuando recibimos la información del gps
i = 1; % Indicador del punto objetivo
v = 1.2; % Velocidad del robot (m/s)
puntos = [0 0;20 0;20 20;-10 30;-10 -10;0 -30;0 0]; % Puntos de la ruta

N = 40*t; % Define el tiempo de simulación ya que los saltos son de 0.025s
wi = zeros(1,N); % Vector con los valores de velocidad angular de la rueda izquierda
wd = zeros(1,N); % Vector con los valores de velocidad angular de la rueda izquierda
wid = zeros(1,N);
wdd = zeros(1,N);
```

Figura 2.- Variables iniciales

Después de esto, vamos a empezar a simular el comportamiento del robot con un bucle *for* y vamos a ir pintando los puntos por los que pasa el robot. Primero, vamos a crear la figura donde vamos a pintar la localización y la velocidad del robot. Lo vemos en la Figura 3.

```
figure
subplot(2,1,1)
plot(0,0,"*b")
hold on
subplot(2,1,2)
plot(0,0,"*k")
hold on
```

Figura 3.- Iniciamos las figuras para pintar las trayectorias

Ahora, entremos en el bucle *for*:
Lo primero que vamos a hacer, va a ser comprobar si nos toca o no recibir la posición del robot según el gps:

```
% Vamos recibiendo cada 0.3 segundos la información del gps
if(sgps < 0.3)
    sgps = sgps + dt;
else
    sgps = 0;
    pos = DGPS(x,y,o);
    xgps = pos(1);
    ygps = pos(2);
    ogps = pos(3);
end
```

Figura 4.- Obtención del gps cada 0,3 segundos

Tras esto, pasamos a coordenadas locales el punto al que queremos ir, para saber qué curvatura y dirección hay que tomar para llegar allí. Lo vemos en la Figura 5.

```
% Pasamos el punto al que queremos ir a coordenadas locales
puntox = (puntos(i,1)-xgps)*cos(ogps)+(puntos(i,2)-ygps)*sin(ogps);
puntoy = (puntos(i,2)-ygps)*cos(ogps)-(puntos(i,1)-xgps)*sin(ogps);

% Calculamos la distancia al punto para pasar al siguiente objetivo
% cuando estemos a menos de un metro del punto
distancia = sqrt(puntox^2+puntoy^2);
if (distancia <= 1)
    i = i + 1;
end

% Calculamos el ángulo y la curvatura que va a tener el robot en
% cada instante yendo a por el punto
anguloe = atan2(puntoy,puntox);
curva = 2*anguloe;
```

Figura 5.- Coordenadas locales

Con todos estos cálculos, ya podemos usar el Modelo Cinemático Inverso, en la Figura 6, del robot para obtener cuales son las velocidades angulares de las ruedas que queremos.

```
% Modelo cinemático inverso
wid(k) = v*(1-curva*ruedas/2)/r;
wdd(k) = v*(1+curva*ruedas/2)/r;
```

Figura 6.- Modelo cinemático inverso

Ahora debemos obtener el valor real de las velocidades angulares, para ello, usaremos las ecuaciones en diferencias de las ruedas y obtendremos las velocidades angulares en función de las deseadas y las anteriores en la Figura 7.

```
% Modelo del motor. Sacamos las ecuaciones en diferencias de la
% función de transferencia en Z:
% G = tf((k/tau),[1 1/tau]);
% Ghc = ((1-exp(-dt*s))/s);
%
% Gz = c2d(G,dt); -----> Y/X = (k*0.1881)/(z-0.8119)
%------------------
% w(k + 1) = k*0.1881*u(k) + 0.8119*w(k)
% w(k) = k*0.1881*u(k-1) + 0.8119*w(k-1)
wi(k) = gain*0.1881*wid(k-1)+0.8119*wi(k-1);
wd(k) = gain*0.1881*wdd(k-1)+0.8119*wd(k-1);
```

Figura 7.- Modelo del motor

Tras obtener las velocidades de las ruedas, debemos comprobar que ninguna sobrepase el límite de revoluciones por minuto que soportan los motores en la Figura 8.

```
% Comprobamos que la velocidad de las ruedas no supere el límite
if (wi(k)>15)
    wi(k) = 15;
end
if (wi(k)<-15)
    wi(k) = -15;
end
if (wd(k)>15)
    wd(k) = 15;
end
if (wd(k)<-15)
    wd(k) = -15;
end
```

Figura 8.- Asegurar la velocidad angular máxima

A partir de aquí, nos queda calcular el Modelo Cinemático Directo y calcular los desplazamientos del robot para llevar a cabo la simulación. Lo vemos en la Figura 9.

```
% Modelo Cinemático Directo
v1 = (wi(k)+wd(k))*r/2;
w1 = (wd(k)-wi(k))*r/ruedas;
subplot(2,1,2)
plot(k,v1,"*b")

% Ecuaciones que llevan a cabo la simulación del movimiento del
% robot
ds = v1*dt;
do = w1*dt;

o = o + do;
dx = cos(o).*ds;
dy = sin(o).*ds;
x = x + dx;
y = y + dy;
subplot(2,1,1)
plot(x,y,"*r")
```

Figura 9.- MCD y Odometría

- Simulación

Teniendo ya el código completo, probamos a realizar la simulación con un valor de ganancia de las ruedas de 1 en la Figura 10.

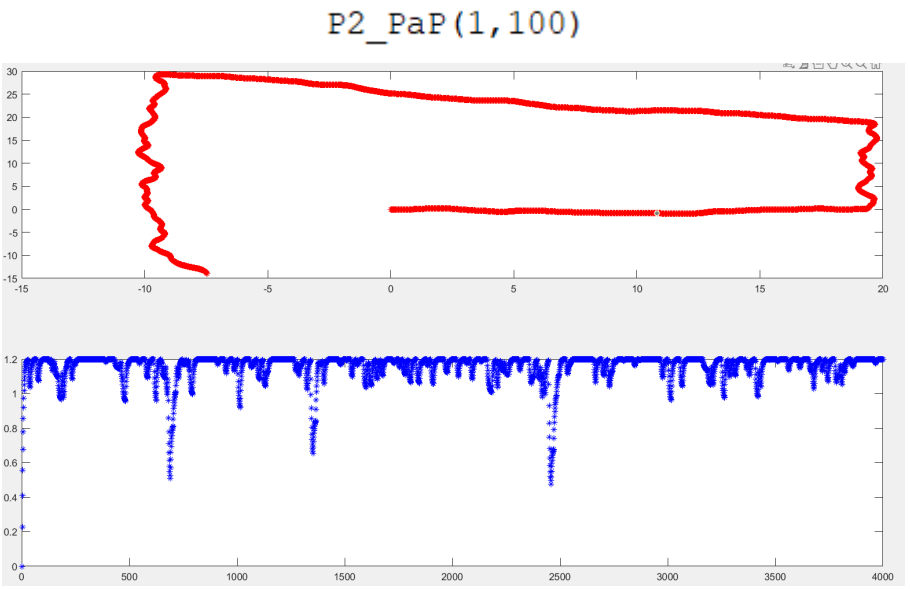

Figura 10.- Simulación con ganancia de las ruedas 1

Ahora, probemos con una ganancia de 1,5 en la Figura 11.

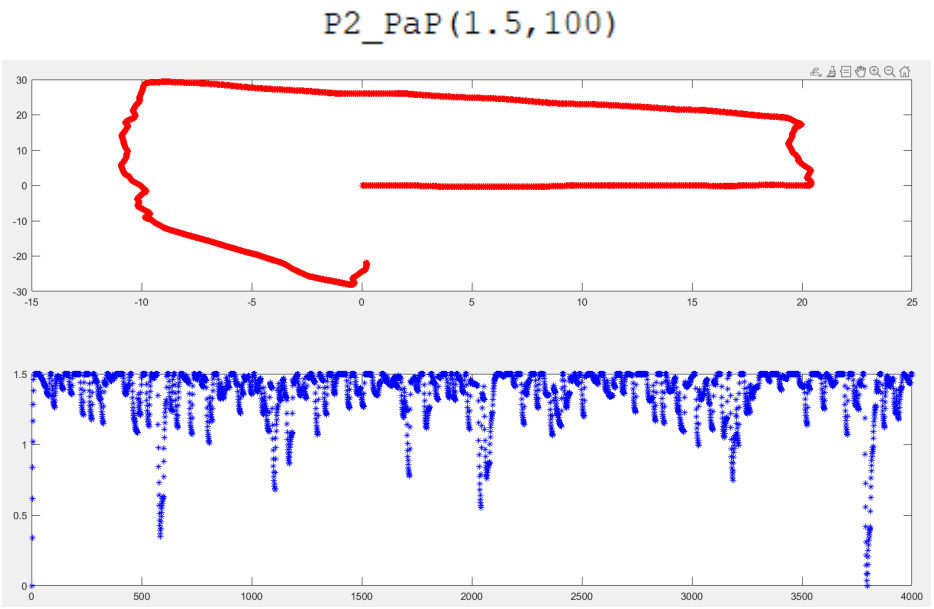

Figura 11.- Simulación con ganancia 1,5

Y probemos también con una ganancia de 0,5 en la Figura 12.

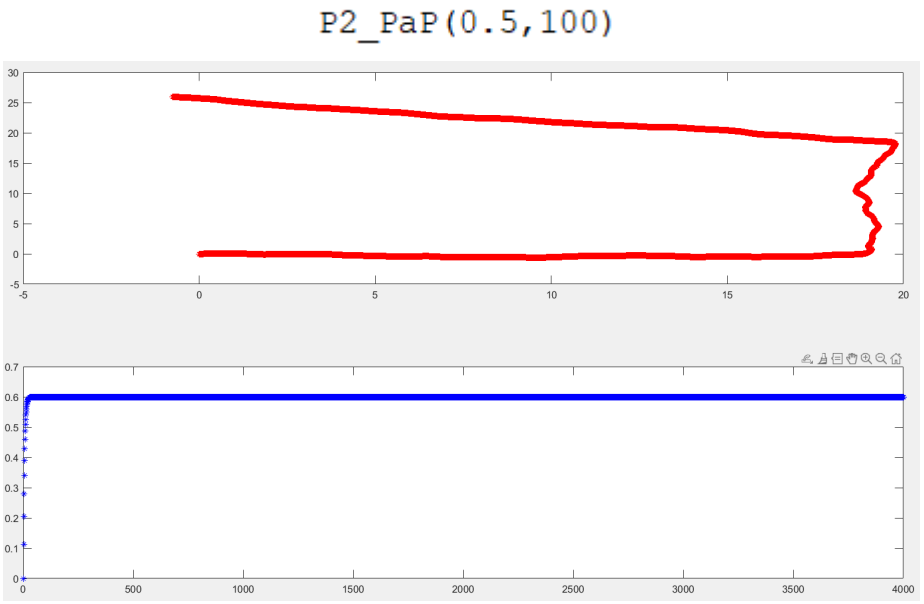

Figura 12.- Simulación con ganancia 0,5

Después de realizar estas tres simulaciones podemos sacar en claro que la velocidad a la que va a ir el vehículo depende directamente de la ganancia que le demos, por esa razón, con una ganancia mayor a 1, obtenemos una respuesta con una velocidad mayor a la consigna, al igual que si es menor a 1, la velocidad también es menor. También podemos ver, en las gráficas de la velocidad, que, a menor ganancia, la velocidad es más estable y no tiene tanta variación. Esto puede deberse a que, a menor velocidad, el robot no necesita dar muchos frenazos ni acelerones para corregir la posición que le ha dado el gps. Entonces, para buscar la ganancia ideal, con la que mantengamos una velocidad estable y

a la vez cercana a los $1,2\,m/s$ que queríamos inicialmente, debemos buscar un valor inferior y próximo a 1. Esto lo vemos en la Figura 13.

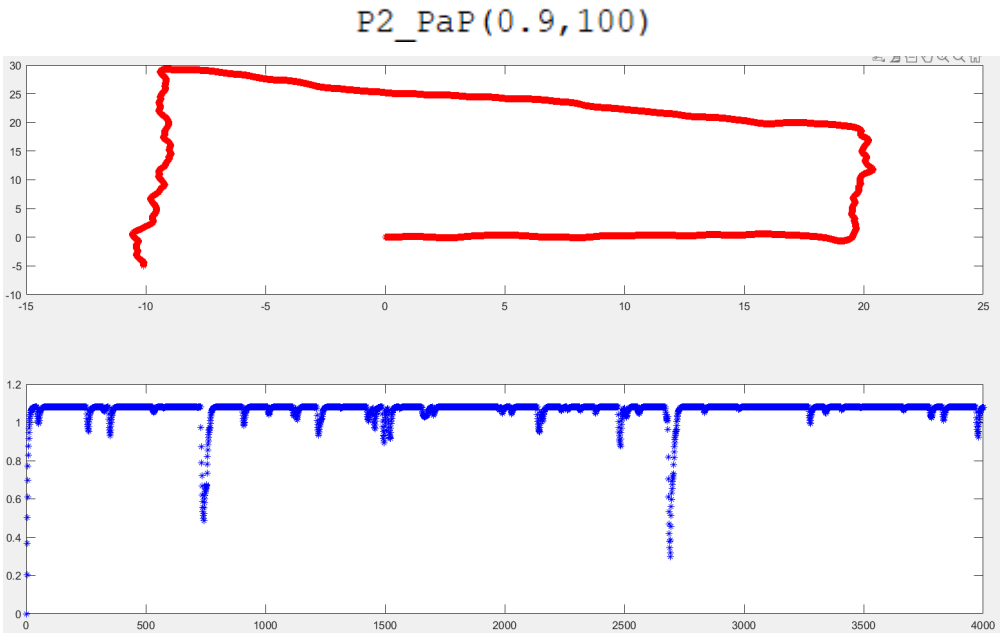

Figura 13.- Simulación con ganancia 0,9

Método de Persecución Pura

- Desarrollo

En este ejercicio, vamos a simular un robot que va a seguir un camino definido por un pasillo, es decir, el robot corregirá su posición para recorrer el pasillo por el centro de este. El esquema lo vemos en la Figura 14.

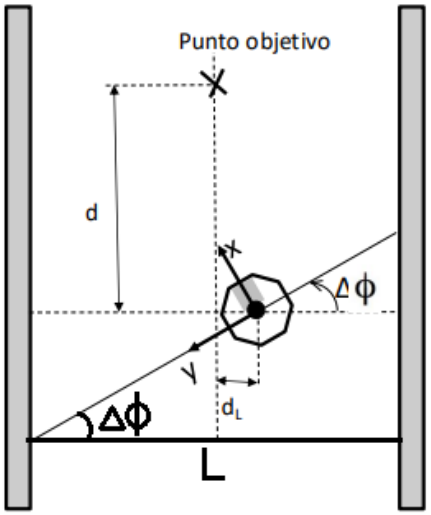

Figura 14.- Esquema del robot navegando por el pasillo

Este caso es parecido al anterior, con la diferencia de que el paso del punto a coordenadas locales del robot es algo distinto, usando los valores del sensor LiDAR que tenemos.

$$\begin{aligned} \delta x &= d_L \cdot \sin(\Delta\phi) + d \cdot \cos(\Delta\phi) \\ \delta y &= d_L \cdot \cos(\Delta\phi) - d \cdot \sin(\Delta\phi) \end{aligned}$$

Figura 15.- Ecuaciones para pasar a coordenadas locales

y la curvatura se define como $\gamma = \frac{2 \cdot \delta y}{D^2}$ siendo $D = distancia\ del\ punto\ al\ robot$.

Este ejercicio también nos pone un par de casos y condiciones: La velocidad no puede superar los $0,3\ m/s$, el láser, de 360º, nos da información cada $0,5\ s$ y el punto objetivo del camino que estamos mirando se encontrará $1\ m$ por delante del robot. El ejemplo de pasillo que nos da el ejercicio está en la Figura 16.

Coordenada x (m)	Coordenada y (m)
0	0
30	0
30	10
0	10
0	0

Figura 16.- Coordenadas del pasillo

Teniendo ya todo esto en cuenta, comencemos a crear el código.
Para empezar, al igual que en el ejercicio anterior, definimos las variables necesarias en la Figura 17.

```
x = x0; % Posición en x
y = y0; % Posición en y
o = o0; % Orientación
v = 0.3; % Velocidad máxima del robot
d_obj = 1; % Distancia al punto objetivo del camino
L = 10; % Ancho del pasillo en el eje y
ruedas = 0.8; % Distanccia entre las ruedas en metros
r = 0.1; % Radio de las ruedas en metros
dt = 0.025; % Tiempo de muestreo
slaser = 0; % Variable para el tiempo de obtenció de datos del laser

% Entorno cerrado definido por una lista de puntos
paredx=[0 30 30 0 0]';
paredy=[0 0 10 10 0]';

t = 70;
N = 40*t; % Define el tiempo de simulación ya que los saltos son de 0.025s
wi = zeros(1,N); % Vector con los valores reales de la velocidad angular izquierda
wd = zeros(1,N); % Vector con los valores reales de la velocidad angular derecha
wid = zeros(1,N);
wdd = zeros(1,N);
v1 = zeros(1,N);
w1 = zeros(1,N);
```

Figura 17.- Variables iniciales

A partir de aquí, comenzamos con el bucle *for* que va a llevar a cabo la simulación. Lo primero que hacemos, es comprobar si nos toca recibir la información del láser, y en caso afirmativo, obtenemos las distancias de cada uno de los rayos láser que ha lanzado el sensor LiDAR y sacamos las dos distancias en el eje y del robot, como indica el esquema anterior. Por último, comprobamos de qué pared estamos más cerca y calculamos la orientación que tiene el robot respecto al camino central del pasillo. Todo esto lo vemos en la Figura 18.

```
% Vamos recibiendo cada 0.5 segundos la información del laser
if(slaser < 0.5)
    slaser = slaser + dt;
else
    slaser = 0;
    rangos= laser2D(paredx,paredy, x, y, o);
    % rangos(19)--> distancia en la dirección y
    % rangos(55)--> distancia en la dirección -y
    dist = rangos(19) + rangos(55);
    if(rangos(19)>rangos(55))
        o = real(acos(L/dist))*sign(o);
    else
        o = -real(acos(L/dist))*(-sign(o));
    end
end
```

Figura 18.- Obtención cada 0,5 segundos de la información del Lidar

A continuación, pasamos el punto objetivo del camino a coordenadas locales del robot usando las ecuaciones que hemos visto antes.

```
dL = L/2 - y;

% Pasamos el punto al que queremos ir a coordenadas locales
deltax = dL*sin(o)+d_obj*cos(o);
deltay = dL*cos(o)-d_obj*sin(o);

distancia = sqrt(deltax^2 + deltay^2);

% Calculamos la curvatura que va a tener el robot en
% cada instante yendo a por el punto
curva = 2*deltay/distancia^2;
```

Figura 19.- Paso a coordenadas locales

A partir de aquí, el código vuelve a ser el mismo que en el ejercicio anterior. Primero calculamos el Modelo Cinemático Inverso, luego usamos las ecuaciones en diferencias para hallar las velocidades angulares de las ruedas, aseguramos que no sobrepasen las $15\,rpm$, calculamos el Modelo Cinemático Directo y, por último, calculamos los desplazamientos llevados a cabo por el robot. Todo esto lo vemos en la Figura 20.

```
            % Modelo cinemático inverso
            wid(k) = v*(1-curva*ruedas/2)/r;
            wdd(k) = v*(1+curva*ruedas/2)/r;
        wi(k) = gain*0.1881*wid(k-1)+0.8119*wi(k-1);
        wd(k) = gain*0.1881*wdd(k-1)+0.8119*wd(k-1);
    % Comprobamos que la velocidad de las ruedas no supere el límite
    if (wi(k)>15)
        wi(k) = 15;
    end
    if (wi(k)<-15)
        wi(k) = -15;
    end
    if (wd(k)>15)
        wd(k) = 15;
    end
    if (wd(k)<-15)
        wd(k) = -15;
    end

     % Modelo Cinemático Directo
     v1 = (wi(k)+wd(k))*r/2;
     w1 = (wd(k)-wi(k))*r/ruedas;
     subplot(2,1,2)
     plot(k,v1,"*b")

     % Ecuaciones que llevan a cabo la simulación del movimiento del
     % robot
     ds = v1*dt;
     do = w1*dt;

     o = o + do;
     dx = cos(o).*ds;
     dy = sin(o).*ds;
     x = x + dx;
     y = y + dy;
     subplot(2,1,1)
     plot(x,y,"*r")
```

Figura 20.- Código con el MCD, el MCI y la Odometría

Lo último que hacemos es pintar la escena con el camino recorrido por el robot en la Figura 21.

```
dibujaBarrido(paredx, paredy, x, y, o, rangos);
```

Figura 21.- Pintar el barrido del Lidar

- Simulación

Ahora que tenemos el código por completo, vamos a realizar las simulaciones en varios casos:

- Posición y orientación iniciales → (7,1,-0.5) → Figura 22

```
P2_PerPura(7,1,-0.5);
```

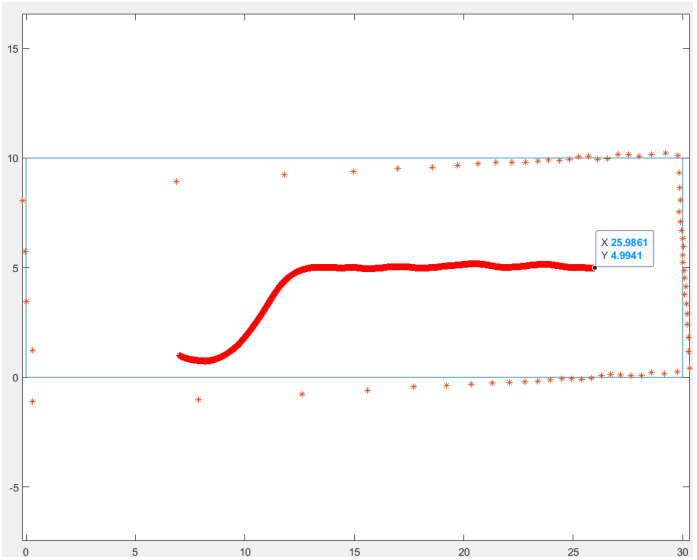

Figura 22.- Simulación iniciando con coordenadas (7,1,-0.5)

- Posición y orientación iniciales → (7,1,0.5) → Figura 23

```
P2_PerPura(7,1,0.5);
```

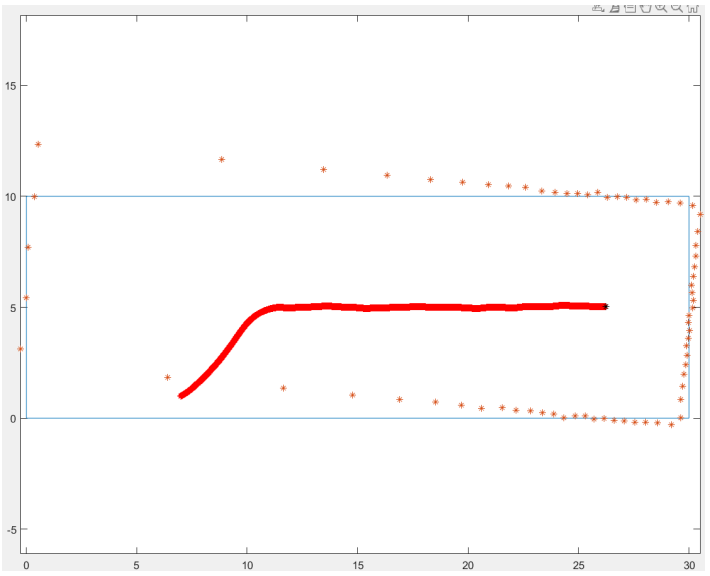

Figura 23.- Simulación con coordenadas iniciales (7,1,0.5)

- Posición y orientación iniciales → (7,9,-0.5) → Figura 24

```
P2_PerPura(7,9,-0.5);
```

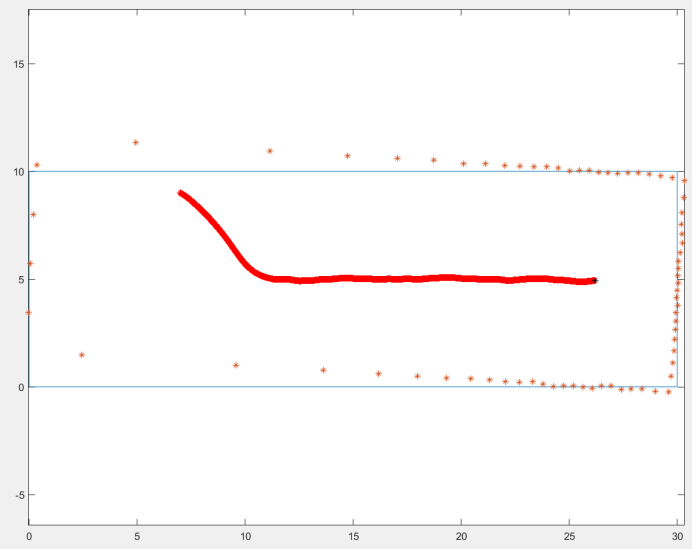

Figura 24.- Simulación con coordenadas iniciales (7,9,-0.5)

- Posición y orientación iniciales → (7,9,0.5) → Figura 25

```
P2_PerPura(7,9,0.5);
```

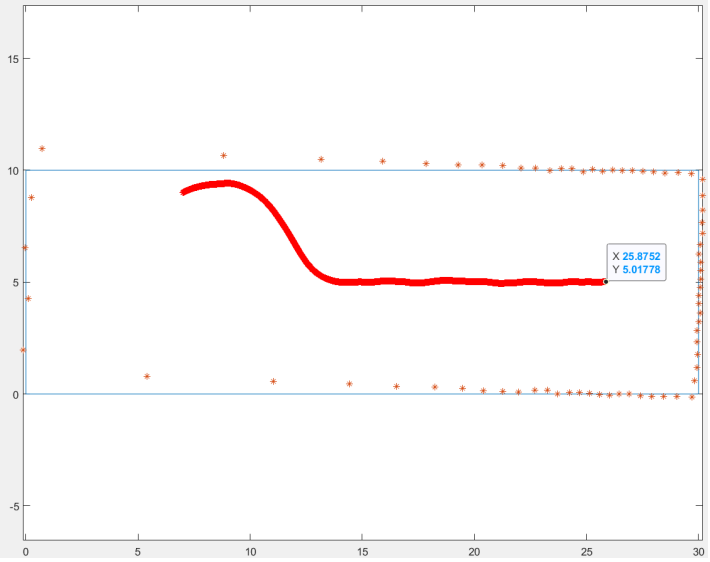

Figura 25.- Simulación con coordenadas iniciales (7,9,0.5)

Enlace a Github

https://github.com/MiguelIlan/AmpliacionRobotica/tree/main/Rob%C3%B3tica%20m%C3%B3vil/Caminos