Escuela Técnica Superior de Ingeniería

Universidad de Huelva

Grado en Ingeniería Informática

Robótica

Memoria de Actividades

Autor: Wadadi Sidelgaum Lima

Fecha: 11/10/2024

Contenido

1. Descripción de la Actividad 1

La primera actividad de este curso tiene como objetivo la introducción a MATLAB, un entorno de programación ampliamente utilizado para realizar cálculos matemáticos complejos y simulaciones.

En esta práctica, exploramos las siguientes competencias:

1. **Operaciones con matrices**: Crear y manipular matrices es una habilidad fundamental en MATLAB, ya que muchas de las simulaciones robóticas dependen de cálculos matriciales. Se nos enseña a definir matrices, acceder a sus elementos, modificar valores específicos y realizar operaciones matemáticas como transposiciones, multiplicaciones y determinación de autovalores y auto vectores.
2. **Simulación gráfica**: A lo largo de la actividad, MATLAB se utiliza para generar gráficos y visualizar resultados numéricos, como funciones y trayectorias de movimiento. Esta capacidad gráfica facilita la interpretación de los datos y ayuda a comprender el comportamiento del robot en diferentes situaciones.

Estas habilidades serán fundamentales para las actividades posteriores en el curso, donde exploramos simulaciones más avanzadas y trabajaremos con controladores proporcional, PI y PID, entre otros.

Para el desarrollo de esta actividad, se ha diseñado un fichero en la primera clase práctica en matlab llamado simulacion_robot.m que permite realizar una simulación gráfica del movimiento de un robot, para ello, se ha hecho uso de las siguientes funciones proporcionadas por el profesor:

1. Robot_sim(): toma como argumentos los valores del tiempo en el instante actual, las variables de control del vehículo, paso de integración(h) y el vector pose que define la posición del robot en eje x-y, ángulo de orientación, ángulo de giro de la cabeza y la velocidad del giro, finalmente, devuelve los valores de pose en el siguiente instante.

2. pinta_robot_v3(): Se encarga de devolver el mapa actualizado con el último punto detectado a partir de la medida del sonar, después se dibuja la simulación con drawnow,

esta función toma como parámetros todos los valores de pose menos la velocidad de giro, la medida proporcionada por el sensor y una variable mapa que contiene todas las coordenadas de todos los puntos anteriores, esta función se invoca al final del bucle del controlador.

3. signal_vf_v2(): se encarga de generar una función de referencia para el movimiento de la cabeza del robot.

A continuación, se explica el código que se ha desarrollado en clase, para resolver los distintos ejercicios de la actividad:

- En la primera parte del código se define los parámetros necesarios y se inicializan:

```
mapa=[];

%Condiciones iniciales
pose0=[0; 0; 0; 0; 0];

%final de la simulación
tf=10;

%paso de integracion
h=0.1;

%indice de la matriz
k=0;
%punto=[30 30];
%inicialización de variables
 pose(:,k+1)=pose0;
%valores para actualizazación
 tiempo=0;
 Integral=0; % valor de la integral en el instante 0
 error_ant=0;

 % otras constantes
 Delay=2;
 Periodo=6;
 Amplitud=pi/2;
```

- Después, entramos en el bucle temporal:

```
while tiempo < tf
    %actualización
    k=k+1;
    %t(k)=toc(tstart);
    t(k)=tiempo+h;
    dt=t(k)-tiempo;
    tiempo=t(k);
```

- En el bucle, después de la actualización del tiempo, definimos las acciones de control, en primer lugar, calculamos la referencia que puede ser (constante, variable o invocando a la función signal mencionada anteriormente) y el error:

```
%--------------------------------
% Aquí se calculan las referencias
%--------------------------------
    referencia(k)=90;
    referencia(k) = 200 * t(k);
    referencia(k) = signal_vf_v2(t(k),Periodo,Delay,Amplitud);
```

```
error(k)=referencia(k)-pose(4,k);
```

- El siguiente paso es calcular el controlador, que puede ser (proporcional, integral o proporcional integral derivativo):

```
%% CONTROL PROPORCIONAL

 kp=0.6984; % constante del controlador proporcional

 Potencia_Cabeza(k)=kp*error(k); % señal la rampa
```

```
%% Control PI
%% ------------------
 Ganancia_p = 1.0226;
 Ganancia_I = 0.290;

Integral = Integral+error(k)*dt;
error_ant=error(k);

 Potencia_Cabeza(k) = Ganancia_p*error(k) + Ganancia_I*Integral; %controlador
```


```
%% Control PI
%% ------------------
 Ganancia_p = 1.0226;
 Ganancia_I = 0.290;

Integral = Integral+error(k)*dt;
error_ant=error(k);

 Potencia_Cabeza(k) = Ganancia_p*error(k) + Ganancia_I*Integral; %controlador
```

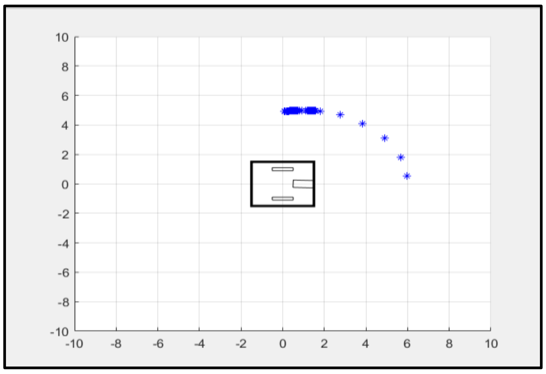

- Definimos los valores de los actuadores:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Aquí se definen los valores de los actuadores
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    if Potencia_Cabeza(k)>100
         Potencia_Cabeza(k)=100;
    elseif Potencia_Cabeza(k)<-100
        Potencia_Cabeza(k)=-100;
    end
    Potencia_rueda_Derecha=0;
    Potencia_rueda_Izquierda=0;
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   if t(k)>5
       Perturbacion= -10;
   else
       Perturbacion = 0;
   end
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   conduccion=[Potencia_rueda_Derecha Potencia_rueda_Izquierda, Potencia_Cabeza(k),Perturbacion];
```

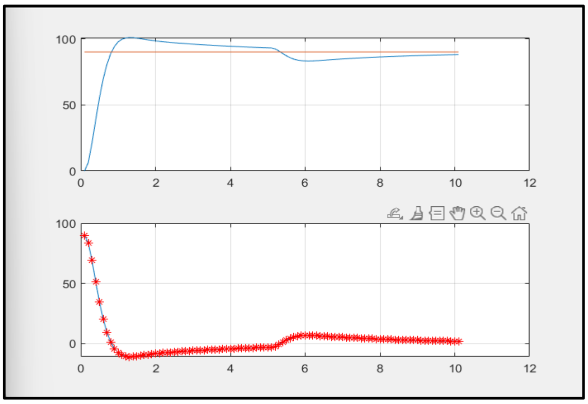
- finalmente, invocamos a las funciones de cálculo y dibujamos el robot:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Se integra el modelo del robot diferencia para simularlo
    pose(:,k+1)=Robot_sim(t(k),pose(:,k),h,conduccion);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Representación del robot
   distancia=5; %proporcionada por el sonar
   mapa=pinta_robot_v3(pose(1,k+1),pose(2,k+1),pose(3,k+1),pose(4,k+1)*pi/180,distancia,mapa);
   grid on
   drawnow
   pause(0.05)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

En la siguiente figura se muestra la simulación del robot:



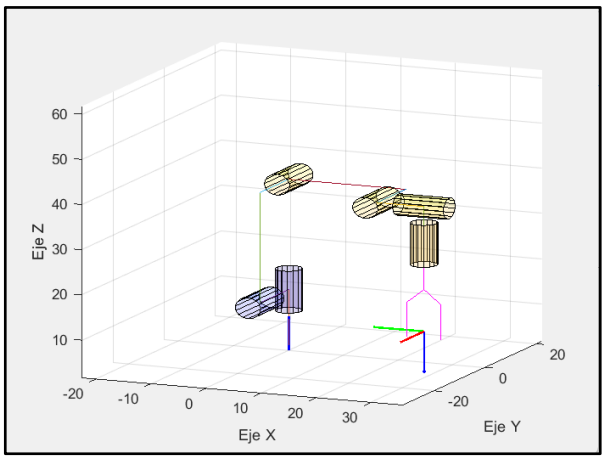
La siguiente imagen representa el valor de la referencia y el error en tiempo por separado:



2. Descripción de la Actividad 2

El objetivo de esta actividad es familiarizarse con el entorno del manipulador UR3, donde controlaremos el movimiento y la orientación de una serie de objetos mediante transformaciones geométricas.

El manipulador está formado por elementos articulados entre sí, destinados a tareas de manipulación y agarre de objetos, a continuación, se muestra un esquema del manipulador UR3 en matlab:



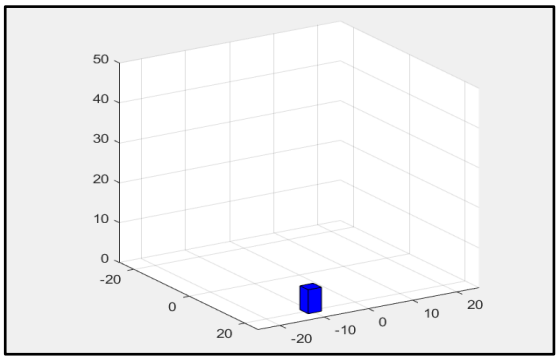
Esta actividad se divide en dos partes:

2.1. Primer Parte:

La primera parte consiste en hacer transformaciones de matrices básicas (desplazamientos, rotaciones) para establecer el procedimiento que determina la configuración de la pinza del manipulador en diferentes situaciones, para ello se ha propuesto los siguientes ejercicios:

1. El primer ejercicio consiste en representar un bloque en distintas configuraciones del espacio:

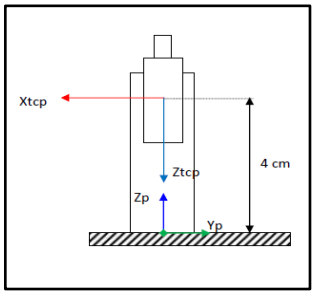
```
%% Ejecicio A)
%configuración de la pieza
posicion=[20 -10 0];
alfa=0; beta=0; gamma=0;

matriz_pieza=Desplazamiento(posicion(1), posicion(2), ...
    posicion(3))*Rotacionz(alfa)*Rotaciony(beta)*Rotacionx(gamma);

%pinta_pieza_delgada(matriz_pieza)
pinta_bloque(matriz_pieza,'b')
```



Para poder representar el bloque, tenemos que definir lo siguiente:

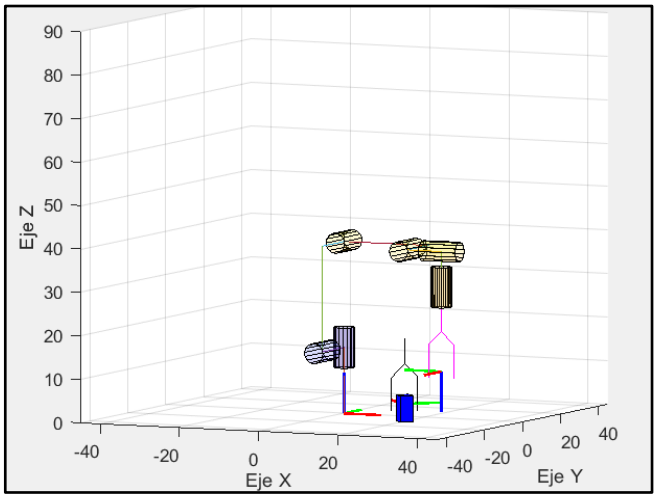
a. La posición del bloque en el eje de coordenadas.
b. Los ángulos de Euler Z-Y-X, que define los ángulos que permiten hacer rotaciones para los ejes z-y-x del bloque, aquí están a 0.
c. Calculamos la matriz de transformación correspondiente al bloque, para ello se utiliza la función Desplazamiento, que genera la matriz asociada a la traslación, pasándole las posiciones del bloque con las rotaciones correspondiente al bloque que se calculan mediante las funciones (Rotacionz, Rotaciony, Rotacionx).
d. Pintamos el bloque con la función pinta_bloque(), donde le indicamos por parámetro la matriz de transformación de la pieza y el color en el que se tiene que representar.

2. En este ejercicio tenemos que determinar la matriz de transformación asociada a la pinza para que ésta se sitúe respecto de la pieza que se sitúa en la posición (20,-10,0) con todos los ángulos de euler a 0, según como se indica en la siguiente figura:



El código y el resultado se muestra en las siguientes figuras:

```
%% Ejecicio B)
clear all
clc
%pinta_bloque(Desplazamiento(8,8,8),'b');
alfa=0; beta=0; gamma=0; % Angulos de Euler
posicion=[20 -10 0];

matriz_pieza=Desplazamiento(posicion(1), posicion(2), posicion(3))*Rotacionz(pi/4);
matriz_pinza=eye(4,4); % matriz unidad

pinta_bloque(matriz_pieza,'b')

matriz_agarre = Desplazamiento(0,0,4)*Rotacionz(pi/2)*Rotacionx(pi); % *Rotaciony(pi/2)*Rotacionx(-pi/2)
% matriz_agarre = Desplazamiento(0,0,4)*Rotacionz(pi/4)*Rotacionz(-pi/2)*Desplazamiento(0,5,0)

matriz_pinza = matriz_pieza*matriz_agarre;
q=[0 -1.5700 -1.5700 -1.5700 1.5700 0];
matriz=funcion_pinta_UR3_new(q, matriz_pinza)
```



Explicación del código anterior:

a. Definimos la posición y los ángulos de euler correspondientes a la pieza.
b. Calculamos la matriz de transformación de la pieza con una rotación de 45º grados en el eje z.
c. Creamos la matriz de nuestra pinza, es una matriz de identidad 4*4.
d. Definimos la matriz de agarre de la pinza, con desplazamiento 4 cm en el eje z y rotaciones sobre el eje z y x para posicionar la pinza en función de

la transformación del bloque, que se calcula multiplicando la matriz de pinza anterior por la matriz de agarre.

e. Dibujamos el robot UR3 con la función funcion_pinta_UR3_new(), pasándole por parámetros la matriz q que define los ángulos de las articulaciones del manipulador y la matriz de la pinza.

2.2. Segunda Parte:

Esta parte consiste en resolver el problema cinemático inverso para el manipulador UR3, el objetivo se trata de encontrar los valores articulares que hagan que la pinza del manipulador llegue a una posición y orientación específica.

En esta actividad tenemos que resolver el siguiente ejercicio:

Ejercicio: Hacer una simulación de Resolución pick&place que soluciona el problema cinemático inverso, suponiendo que el pick se encuentra en la posición (30,-10,0) y el place en la posición (20,20,0).

Para resolver el ejercicio, hemos diseñado un código que hace lo siguiente:

Definimos 2 bloques uno en la posición inicial y otro en la final para comprobar la solución, y después calculamos las matrices de transformación que colocan la pinza en las posiciones de coger y soltar la pieza.

```
G_T_pinza1_pick=G_T_pieza1*pieza_T_pinza;
G_T_pinza2_place=G_T_pieza2*pieza_T_pinza;
```

El cálculo consiste en la multiplicación de la matriz de transformación de cada bloque por la matriz de agarre de la pinza, existen tres maneras de agarre:

```
% agarre 1 por arriba situando la pinza alineada
pieza_T_pinza=Desplazamiento(0,0,4)*Rotacionz(pi/2)*Rotaciony(pi);

% agarre 2 por arriba con el eje de la pinza invertido
pieza_T_pinza=Desplazamiento(0,0,4)*Rotacionx(pi);

% agarre 3 por el lado
pieza_T_pinza=Desplazamiento(0,-0.8,4)*Rotacionx(-pi/2)*Rotacionz(pi/2);
```

Una vez calculadas las matrices del pick y el place, configuramos los parámetros de la cinemática inversa (codo, avance y simétrico).

Para los cálculos de la configuración pick-place de una pieza tenemos los siguientes pasos:

1. Aproximación del pick: aproximar la pinza a la pieza multiplicando la matriz de transformación del bloque por un desplazamiento -6 en el eje z.

```
%Paso 1 ---aproxiamcion pick  con respecto al eje del sistema local x eso se multiplica dsp ----------------

matriz_aprox=matriz_pinza1_pick*Desplazamiento(0,0,-6);

[q1 q2 q3 q4 q5 q6]=inv_kinema_ur3_new(matriz_aprox,codo,avance,simetrico);

funcion_pinta_UR3_new([q1 q2 q3 q4 q5 q6],matriz_aprox);

pause
cla
```

2. Pick: posicionar el robot para agarrar la pieza.

```
[q1 q2 q3 q4 q5 q6]=inv_kinema_ur3_new(matriz_pinza1_pick,codo,avance,simetrico);

funcion_pinta_UR3_new([q1 q2 q3 q4 q5 q6],matriz_pinza1_pick);
```

3. Despegue del pick: el despegue consiste en desplazar el pick 6 cm en el eje z.

```
% Movimento del despegue del pick con respecto al eje de referencia global
 matriz_despegue_pick = Desplazamiento(0,0,6)*matriz_pinza1_pick;
 [q1 q2 q3 q4 q5 q6]=inv_kinema_ur3_new(matriz_despegue_pick,codo,avance,simetrico);
 funcion_pinta_UR3_new([q1 q2 q3 q4 q5 q6],matriz_despegue_pick);
```

4. Aproximación del place: consiste en poner la pinza 6 cm arriba del place.

```
matriz_aprox2=Desplazamiento(0,0,6)*matriz_pinza2_place;
[q1 q2 q3 q4 q5 q6]=inv_kinema_ur3_new(matriz_aprox2,codo,avance,simetrico);
```

5. Place: posicionar el robot para soltar la pieza.

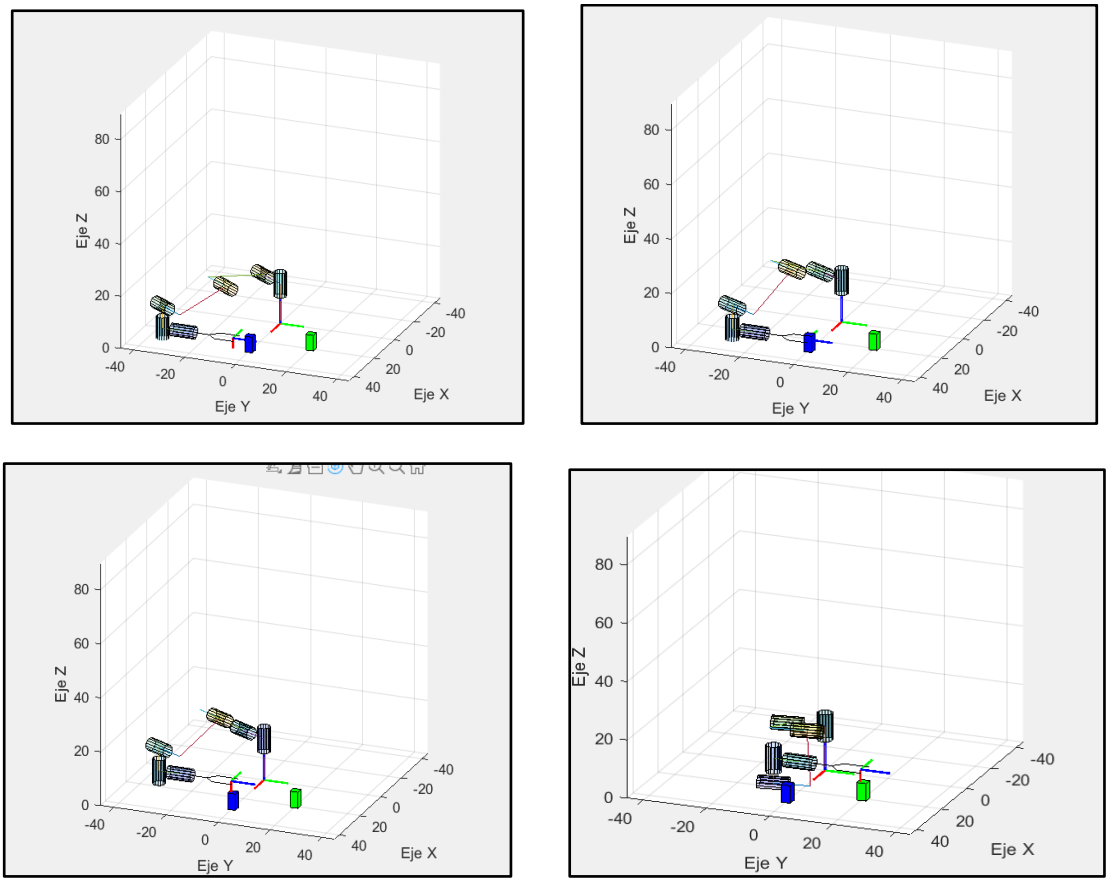
```
[q1 q2 q3 q4 q5 q6]=inv_kinema_ur3_new(matriz_pinza2_place,codo,avance,simetrico);

funcion_pinta_UR3_new([q1 q2 q3 q4 q5 q6],matriz_pinza2_place);
```

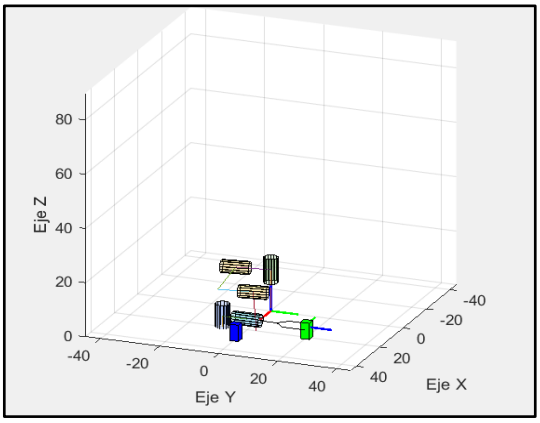
6. Despegue del place: el despegue consiste en desplazar el pick -6 cm en el eje z.

```
 matriz_despegue_place = matriz_pinza1_pick*Desplazamiento(0,0,-6);
[q1 q2 q3 q4 q5 q6]=inv_kinema_ur3_new(matriz_despegue_place,codo,avance,simetrico);
funcion_pinta_UR3_new([q1 q2 q3 q4 q5 q6],matriz_despegue_place);
```

La función inv_Kinema_ur3_new() soluciona el problema cinemático inverso para nuestro manipulador UR3, donde le indicamos la matriz de transformación y los parámetros de la cinemática inversa.

A continuación, se muestra el resultado de la simulación:





3. Descripción de la Actividad 3

La actividad tiene como objetivo explorar y analizar el comportamiento de robots móviles con diferentes modelos cinemáticos (triciclo y diferencial) mediante simulaciones implementadas en MATLAB. Se utilizan archivos proporcionados (simulación_triciclo.m y simulación_diferencial.m) para simular el movimiento y evaluar la respuesta del sistema bajo distintas configuraciones.

Para el desarrollo de los ejercicios vistos en clase, esta actividad se divide en dos partes(clases):

3.1. Primera Parte:

Se trata de hacer uso del fichero simulacion_triciclo, que contiene un bucle temporal que permite simular el movimiento de un robot con la cinemática del triciclo, la simulación se apoya en la función kuta_triciclo (que recibe el valor del tiempo, la configuracion del robot, el paso de integracion(h) y un array de valores de control del robot).

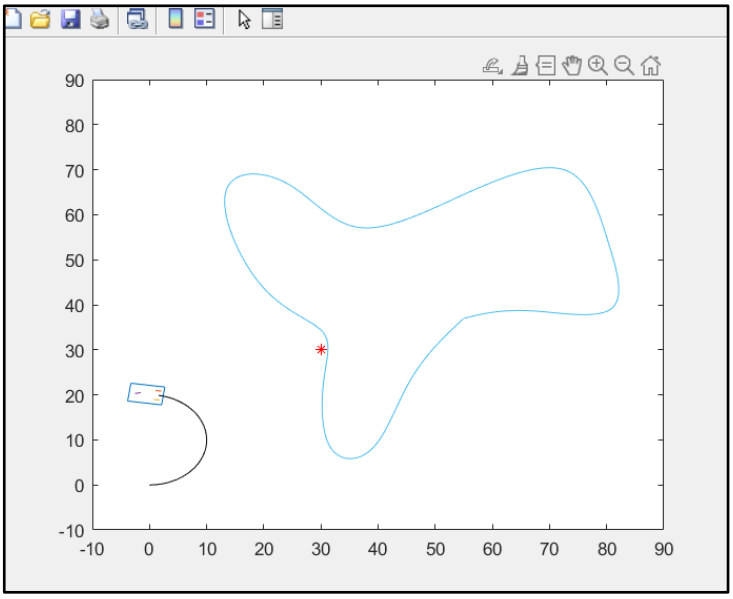
En esta parte, se pide determinar ajustar los parámetros de control del robot para que describa una trayectoria circular con un radio predefinido de 10 unidades, implicando el análisis de las relaciones cinemáticas del modelo de triciclo, siguiendo la siguiente fórmula para calcular el valor del volante:

$$R = \frac{1}{\rho} = \frac{l}{\tan\phi}$$

de manera que p (es el valor del volante), l (el valor de la distancia entre la rueda delantera y trasera) y R (el radio de la trayectoria), la modificación realizada en el bucle temporal es la siguiente:

```
while (t0+h*k) < tf
    %actualización
    k=k+1;

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %valores de los parámetros de control
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    volante=atan(l/R);
    velocidad=2;
    conduccion=[velocidad volante];
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %para representar el punto onjetivo sobre la trayectoria
     punto=[30 30];
    %metodo de integración ruge-kuta
     pose(:,k+1)=kuta_triciclo(t(k),pose(:,k),h,conduccion);

end
```

A continuación, se muestra una figura de la trayectoria del robot:



3.2. Segunda Parte:

Se trata de hacer uso del fichero simulacion_diferencial, que permite realizar una simulación similar al triciclo, pero con vehículo con un con cinemática diferencial. La diferencia más significativa de esta simulación está en que la variable conducción debe contener los valores de las velocidades de rotación de la rueda derecha e izquierda respectivamente.

Para ello, se pide obtener el valor de las velocidades de giro de las ruedas para conseguir que el vehículo describa un radio de longitud igual a 10 unidades, utilizando las siguientes fórmulas, en primer lugar, para el volante:

$$\rho = \frac{1}{R} \qquad \phi = \operatorname{atan}(l \cdot \rho)$$

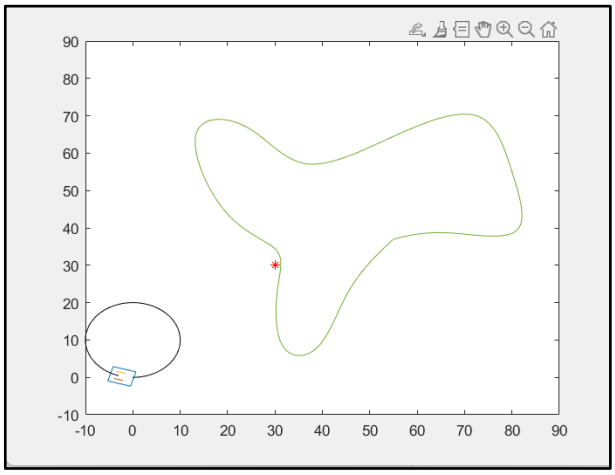
Para las velocidades de las ruedas del coche:

$$\dot{\alpha}_1 = \frac{V + \omega \cdot l}{r}$$
$$\dot{\alpha}_2 = \frac{V - \omega \cdot l}{r}$$

de manera que la V (es la velocidad en este caso equivale a 2 unidades), R (el radio de giro de la trayectoria), l (distancia entre la rueda delantera y trasera) y r (el radio de las ruedas), en la siguiente imagen se muestra el cálculo de los valores en el bucle temporal del fichero simulacion_diferencial:

```
while (t0+h*k) < tf
    %actualización
    k=k+1;

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %valores de los parámetros de control
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    velocidad=2;
    R=10;
    rho=1/R;
    velocidad_derecha=velocidad*(1+l*rho)/radio_rueda;
    velocidad_izquierda=velocidad*(1-l*rho)/radio_rueda;
    phi=atan(rho*l);
    %volante=-0.1416;
    volante=phi;
    conduccion=[velocidad_derecha velocidad_izquierda];
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %para representar el punto onjetivo sobre la trayectoria
    punto=[30 30];
    %metodo de integración ruge-kuta
    pose(:,k+1)=kuta_diferencial(t(k),pose(:,k),h,conduccion);

end
```

Finalmente, se muestra una figura del comportamiento del robot:



4. Descripción de la Actividad 4

Esta actividad consiste en diseñar un controlador geométrico para un vehículo diferencial, que nos permita situarlo sobre un punto determinado, para ello, tomamos como base el código de la segunda parte de la actividad anterior.

4.1. Primera Parte (Ejercicio A):

Se pide determinar el valor de rotación de las dos ruedas para que el vehículo trace una trayectoria circular que conecte la configuración inicial con el punto objetivo, para ello fijamos un valor constante de velocidad para la velocidad lineal del vehículo.

En primer lugar, calculamos el valor de la curvatura utilizando la siguiente fórmula:

$$\rho = \frac{1}{R} = f(\Delta) = \frac{2(\Delta)}{L_H^2}$$

Para calcular delta, que mide la desviación del vehículo con respecto al camino deseado, aplicamos la siguiente fórmula:

$$\Delta = (x_0 - x_f)\cdot \sin\theta_0 - (y_0 - y_f)\cdot \cos\theta_0 ;$$

donde (x0, y0) son las coordenadas de la posición actual del vehículo y (xf,yf) son las coordenadas del punto objetivo, y el alfa es el ángulo de orientación del vehículo.

Una vez obtenida delta, calculamos la distancia directa con nuestro punto objetivo (L_h), donde:

$$L_H = \sqrt{(x_0 - x_f)^2 + (y_0 - y_f)^2} ;$$

A continuación, se encuentra el desarrollo del código en matlab:

```
while (t0+h*k) < tf
    %actualización
    k=k+1;
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %valores de los parámetros de control
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    punto=[30 5];

    delta=(pose(1,k)-punto(1))*sin(pose(3,k))-((pose(2,k)*punto(2))*cos(pose(3,k)));
    L_h=sqrt(((pose(1,k)-punto(1))^2)+((pose(2,k)-punto(2))^2));
    rho=2*delta/(L_h^2);

    velocidad = 5;
    velocidad_derecha=velocidad*(1+l*rho)/radio_rueda;
    velocidad_izquierda=velocidad*(1-l*rho)/radio_rueda;
    conduccion=[velocidad_derecha velocidad_izquierda];
    %metodo de integración ruge-kuta
    pose(:,k+1)=kuta_diferencial(t(k),pose(:,k),h,conduccion);

end
```

4.2. Segunda Parte (Ejercicio B):

En este ejercicio, el objetivo es modificar el controlador para que la velocidad del vehículo sea proporcional a la distancia entre su posición actual y el punto objetivo.

Para ello, hemos diseñado las siguientes funciones que facilitan el desarrollo del código:

- Función Controlador Geométrico: esta función se encarga de devolver el valor de la curvatura calculado con las fórmulas del ejercicio anterior y el valor de la velocidad que consiste en el producto de una constante por la distancia, esta función recibe como parámetro la posición actual del vehículo y las coordenadas del punto objetivo.

```
function [V, p] = funcion_controlador_geometrico(pose, punto)
    delta=(pose(1)-punto(1))*sin(pose(3))-(pose(2)-punto(2))*cos(pose(3));
    L_h=sqrt(((pose(1)-punto(1))^2)+((pose(2)-punto(2))^2));
    distancia=sqrt(((pose(1)-punto(1))^2)+((pose(2)-punto(2))^2));
    p=2*delta/(L_h^2);
    kpv=1;
    V=kpv*distancia;
end
```

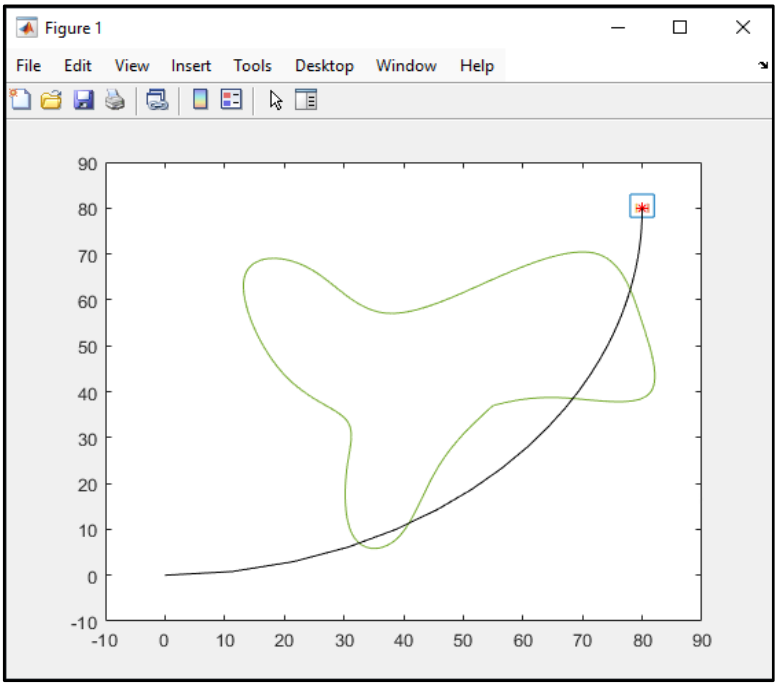
- Función Modelo Cinemático Inverso: recibe por parámetro el valor de la velocidad y la curvatura obtenidos con la función anterior y devuelve el valor de las velocidades de las ruedas del vehículo.

```
function [v_d, v_i] = funcion_modelo_cinematico_inverso(V, p)
global l
global radio_rueda

v_d=V*(1+l*p)/radio_rueda;
v_i=V*(1-l*p)/radio_rueda;

end
```

En la siguiente imagen se encuentra el código con las llamadas a las funciones anteriores:

```
while (t0+h*k) < tf
    %actualización
    k=k+1;
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %valores de los parámetros de control
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %punto=[80 80]
    %punto=[camino(k,1) camino(k,2)];
    punto=[camino(500,1) camino(500,2)];

    [V p]=funcion_controlador_geometrico(pose(:,k),punto);
    [velocidad_derecha velocidad_izquierda]=funcion_modelo_cinematico_inverso(V,p);

    conduccion=[velocidad_derecha velocidad_izquierda];
    %metodo de integración ruge-kuta
    pose(:,k+1)=kuta_diferencial(t(k),pose(:,k),h,conduccion);

end
```

A continuación, se muestra la trayectoria del vehículo con la velocidad proporcional a la distancia hacia el punto (80,80):



Además, en clase se ha pedido la implementación de la trayectoria del robot hacia un array de puntos que se encuentran en el camino en vez de fijar su valor, de manera que el robot una vez alcance un punto objetivo, toma el siguiente como objetivo.

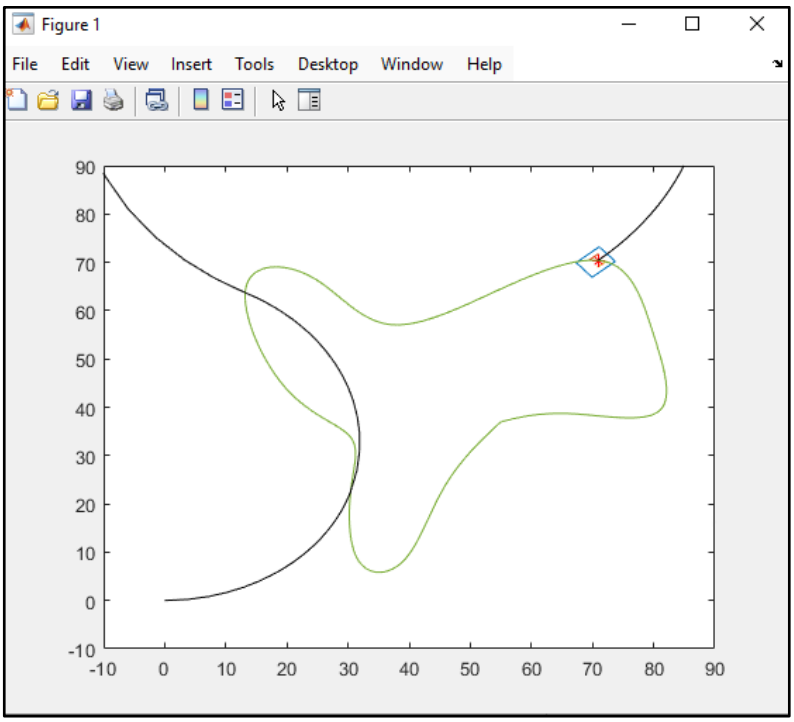
Primero, definimos el array de punto:

```
puntos=[camino(500,:); camino(800,:); camino(1200,:)];
```

Después, se ha modificado el bucle temporal de la siguiente manera:

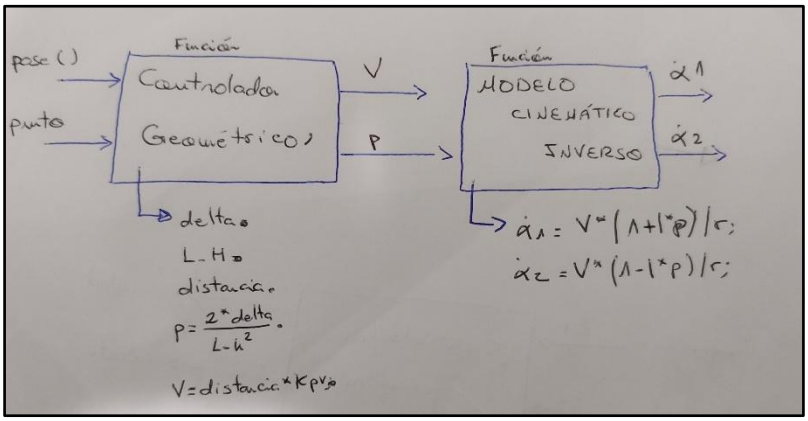
```
while (t0+h*k) < tf
    %actualización
    k=k+1;
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %valores de los parámetros de control
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %punto=[80 80]
    %punto=[camino(k,1) camino(k,2)];
    if i>3
        i=3;
    end
    punto=puntos(i,:);

    [V p]=funcion_controlador_geometrico(pose(:,k),punto);
    [velocidad_derecha velocidad_izquierda]=funcion_modelo_cinematico_inverso(V,p);
    conduccion=[velocidad_derecha velocidad_izquierda];
    %metodo de integración ruge-kuta
    pose(:,k+1)=kuta_diferencial(t(k),pose(:,k),h,conduccion);
    if V<0.01 % para no coger la distancia, ya que la velocidad es proporcional a la distancia
        i=i+1;
    end
end
```

El primer if sirve para controlar que no se pase el índice del array de puntos, después definimos nuestro punto objetivo como el punto del índice i, la variable i se incrementa si la velocidad o la distancia entre el vehículo y el punto objetivo en ese momento es menor que 0.01, que es lo contempla en el último if, finalmente, la trayectoria del vehículo es la siguiente con el array de puntos anteriores:



5. Descripción de la Actividad 5

Esta actividad es una mezcla de las dos anteriores, que se trata de implementar un seguimiento de trayectoria más preciso del vehículo, diseñado un controlador que sigue el siguiente esquema donde el primer bloque es el controlador geométrico que hemos diseñado en la actividad 4 y el segundo bloque es el modelo cinemático inverso desarrollado en la actividad 3:



Para ello se pide el desarrollo de las siguientes nuevas funciones:

- Funcion_Controlador_Geometrico2: es una implementación de la función funcion_controlador_geometrico modificando el cálculo de la distancia, se manera que la distancia este entre la posición del vehículo actual y un punto final pasado por parámetro.

```
function [V, p] = funcion_controlador_geometrico2(pose, punto,punto_final)  % punto_final del camino
    delta=(pose(1)-punto(1))*sin(pose(3))-(pose(2)-punto(2))*cos(pose(3));
    L_h=sqrt(((pose(1)-punto(1))^2)+((pose(2)-punto(2))^2));
    distancia=sqrt(((punto_final(1)-pose(1))^2)+((punto_final(2)-pose(2))^2));
    p=2*delta/(L_h^2);
    kpv=1;
    V=kpv*distancia;
    if V>30
        V=30;
    end
end
```

- Funcion_minima_distancia: esta función se encarga de que dado un camino y la posición actual devuelve el índice del punto más cercano en el camino.

```
function orden_minimo= funcion_minima_distancia(path, punto)  % indice del punto mas cercano

distancia=sqrt((punto(1)-path(:,1)).^2+(punto(2)-path(:,2)).^2);

[minimo orden_minimo]=min(sqrt(distancia));

end
```

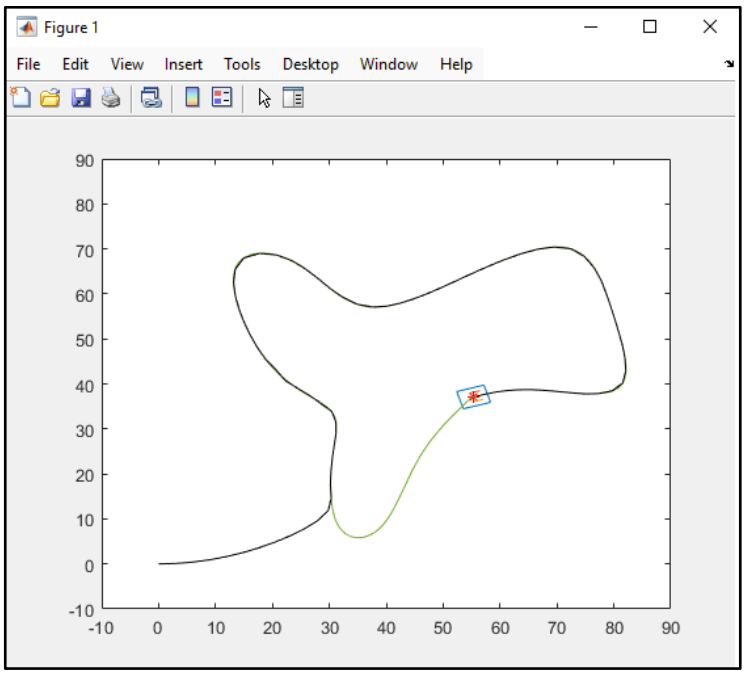
- Funcion_pure_persuit: recibe por parámetro el camino, la posicion actual del vehiculo y el look_ahead (constante para aproximar los pasos hacia adelante para seleccionar el punto final), calcula el punto más cercano y al índice le suma el look_ahead para devolver el punto objetivo, el if sirve para controlar que no se pase el índice de la longitud del camino.

```
function punto = pure_presuit(camino,pose,look_ahead)

 ind=funcion_minima_distancia(camino,[pose(1) pose(2)]);
 if(ind + look_ahead >length(camino))
     %punto=camino(mod(ind+look_ahead,length(camino)),:);
     punto=camino(length(camino),:);
 else
     punto=camino(ind+look_ahead,:);
 end
end
```

Finalmente, el código del bucle temporal es el siguiente:

```
V=10;
look_ahead=20;
while (t0+h*k) < tf
    %actualización
    k=k+1;

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %valores de los parámetros de control
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    punto=pure_presuit(camino,pose(:,k),look_ahead);
    punto_final=camino(length(camino),:);
    %[V p]=funcion_controlador_geometrico(pose(:,k),punto);

    [V p]=funcion_controlador_geometrico2(pose(:,k),punto,punto_final);
    [velocidad_derecha velocidad_izquierda]=funcion_modelo_cinematico_inverso(V,p);
    conduccion=[velocidad_derecha velocidad_izquierda];

    %metodo de integración ruge-kuta

    pose(:,k+1)=kuta_diferencial(t(k),pose(:,k),h,conduccion);
end
```

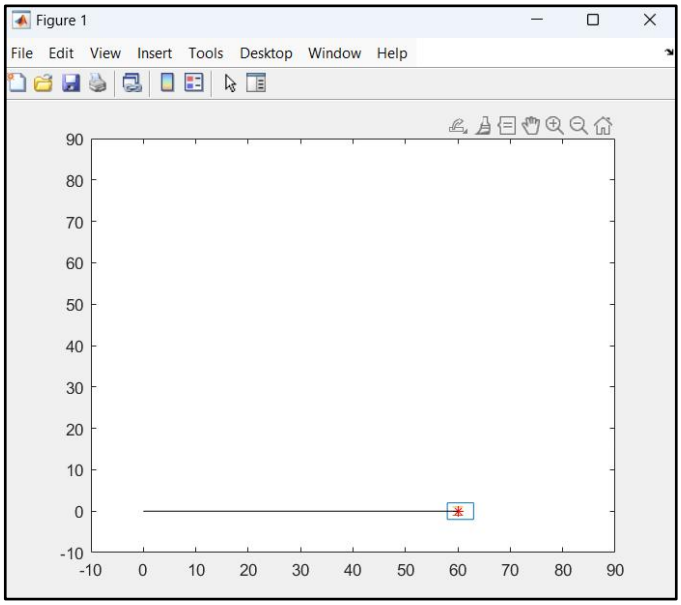
El bucle se encarga de invocar a las funciones que calculan los valores de las velocidades de las ruedas del vehículo, además el comportamiento del controlador consiste en desde la configuración inicial del vehículo se calcula el punto más cercano añadiendo el look_ahead que en este caso son 20 pasos, hasta el llegar al último punto del camino, en la siguiente captura se muestra comportamiento del vehículo durante el bucle:



Otra implementación de esta actividad consiste en modificar el camino, de manera que sea lineal, para ello se ha modificado el código anterior para que, en vez de cargar el camino, definirlo de la siguiente manera:

```
x=20:0.1:60;
y=zeros(size(x));
camino=[x' y'];
```

La trayectoria del vehículo es la siguiente:



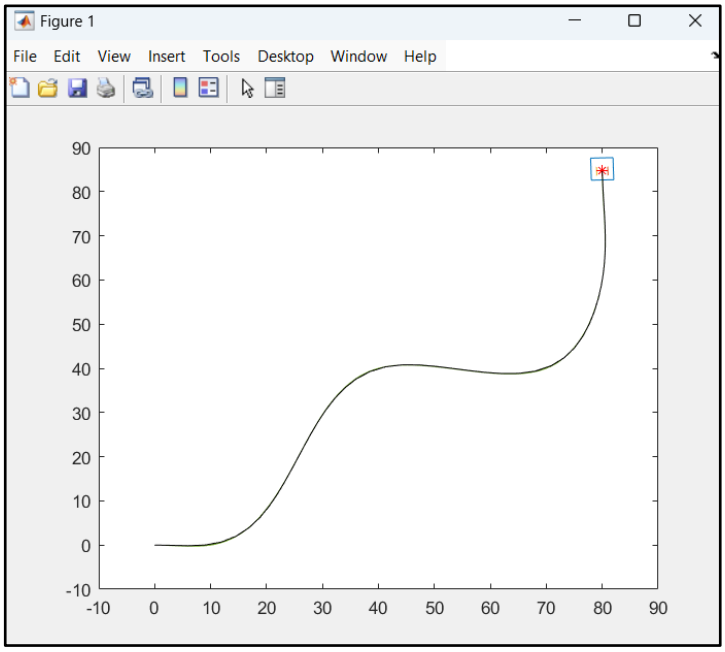
6. Descripción de la Actividad 6

El desarrollo de esta actividad consiste en resolver un ejercicio de manera que el robot siga una trayectoria con splines, para ello se ha usado la función funcion_spline_cubica_varios_puntos que recibe como argumentos un vector xc que corresponde a las coordenadas x de los puntos del camino y otro yc con las coordenadas y, además recibe una variable ds que define la distancia entre los puntos que forman la curva y finalmente devuelve el camino que contiene las coordenadas x e y de los puntos que definen la curva separados por una la distancia ds.

Para la implementación se ha modificado el código de la actividad anterior, de forma que el camino final es el array devuelto por la llamada a la función anterior:

```
xc=[0 10 40 70 80 80];
yc=[0 0 40 40 60 85];

ds=1;
camino=funcion_spline_cubica_varios_puntos(xc,yc,ds)';
```

A continuación, se observa la trayectoria del robot con la definición del camino con splines:



Una segunda parte de la actividad que se ha implementado en la clase de teoría consiste en hacer que la curva sea tangente a la configuración de la salida y llegada, por lo que es necesario tener en cuenta el ángulo que el robot tiene en ambas configuraciones, para ello, se deben fijar dos puntos adicionales: el punto de despegue (Pd) y el punto de aterrizaje (Pa) que vienen dados por la siguientes formulas:

$$Pdx=X_0+dd*\cos(\theta_0)\ ;\ Pdy=Y_0+dd*\sin(\theta_0)$$

$$Pax=X_f-da*\cos(\theta_f)\ ;\ Pay=Y_0-da*\cos(\theta_f)$$

donde dd es la distancia de despegue, que se fija para determinar la distancia arbitraria del punto de despegue del vehículo a la configuración del vehículo, y da es la distancia de aterrizaje será también la distancia arbitraria del punto de aterrizaje a nuestra configuración final del vehículo, estas dos distancias van en dirección del ángulo de orientación, la implementación en matlab es la siguiente:

```
%diferencia de despegue y aterriza
dd=9*direccion;
da=dd;

posicion_despegue=[pose0(1)+(dd*cos(pose0(3))) pose0(2)+(dd*sin(pose0(3)))];

posicion_aterriza=[posef(1)-(da*cos(posef(3))) posef(2)-(da*sin(posef(3)))];
```

La variable dirección define la dirección del ángulo de orientación del robot.

La definición del camino con splines está dada por el siguiente fragmento de código:

```
%definicion del poligono

xc=[pose0(1) posicion_despegue(1) posicion_aterriza(1) posef(1)];
yc=[pose0(2) posicion_despegue(2) posicion_aterriza(2) posef(2)];

ds=3;
camino=funcion_spline_cubica_varios_puntos(xc,yc,ds)';
```

Donde los arrays de xc e yc están compuestos por las coordenadas de la posición inicial, posición del punto del despegue y aterrizaje y el de la posición de la configuración final del vehículo.

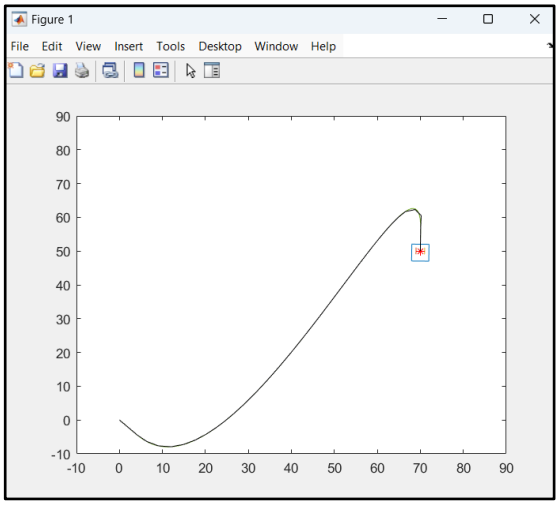
Además, para el cálculo de la velocidad de las ruedas del robot dentro del bucle temporal, se ha creado una nueva función que contiene el código del controlador_geométrico2 más el argumento dirección que su valor es 1 o -1, según la dirección de la orientación del robot del robot, por eso se multiplica por la constante kpv, el código del controlador_geomtrico3 es el siguiente:

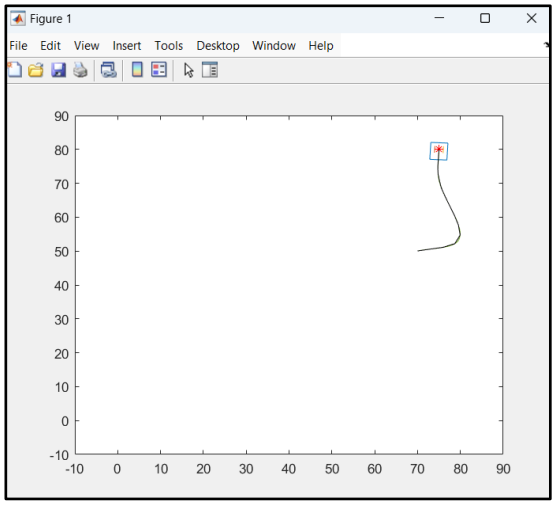
```
function [V, p] = funcion_controlador_geometrico3(pose, punto,punto_final,direccion)
    delta=(pose(1)-punto(1))*sin(pose(3))-(pose(2)-punto(2))*cos(pose(3));
    L_h=sqrt(((pose(1)-punto(1))^2)+((pose(2)-punto(2))^2));
    distancia=sqrt(((punto_final(1)-pose(1))^2)+((punto_final(2)-pose(2))^2));
    p=2*delta/(L_h^2)
    kpv=2*direccion;
    V=kpv*distancia;
    if V>30
        V=30;
    end
    if V<-30
        V=-30;
    end
end
```

Para la simplificación de la ejecución del código de la actividad, se ha creado un nuevo fichero llamado version2 que define las configuraciones de la posición inicial y final del vehículo e invoca al fichero Actividad_6 que contiene el código de todos los calculo anteriores:

```
Actividad_6.m | Version2.m | funcion_controlador_geometrico3.m | +
1
2   clc, close all, clear all
3
4
5   pose0=[0; 0; -pi/4];
6   posef=[70; 50; -pi/2];
7
8
9   direccion=1;
10
11  Actividad_6
12
13
14
15  pose0=pose(:,end);
16  posef=[75; 80; -pi/2];
17
18  pose=[]
19
20  direccion=-1;
21
22  Actividad_6
23
```

Finalmente, las trayectorias del robot según las configuraciones que se muestran en el fichero anterior son las siguientes:





7. Descripción de la Actividad 7

Esta actividad tiene como objetivo Planificar rutas del robot con algoritmos de búsqueda como el a_ estrella evitando el choque con una serie de objetos.

Por un lado, se ha modificado el código de la actividad 6 de manera que podamos representar el mapa a la vez de la trayectoria del robot, el código de la representación del mapa es el siguiente:

```
%-----------------------------------------
% Carga y representación del mapa
%

MAPA = imread('cuadro4.bmp');
```

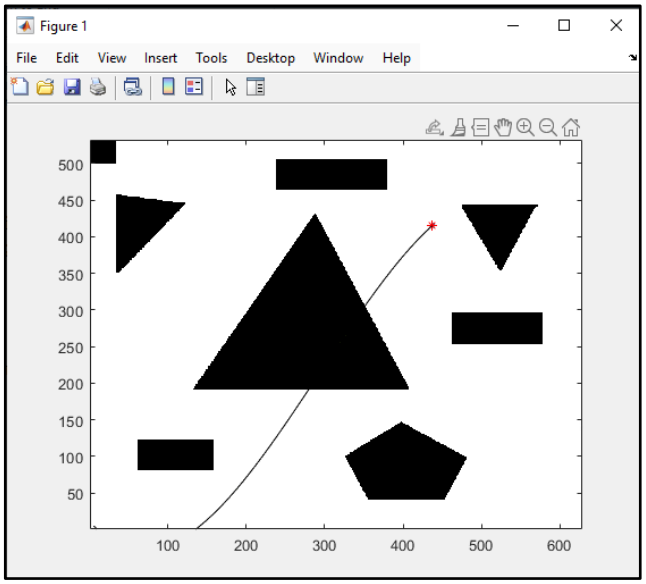
Cargamos la imagen del mapa, hacemos la transformación para colocar el origen de sistema de referencia en el lugar correcto y lo dibujamos mediante el comando image:

```
%Transformación para colocar correctamente el origen del Sistema de
%Referencia
MAPA(1:end,:,:)=MAPA(end:-1:1,:,:);
image(MAPA);
axis xy
%-------------------------------------------------
```

Para representar la trayectoria del robot con la representación del mapa, se invoca al final del bucle temporal a la función kuta_diferencial_mapa, donde le pasamos los mismos argumentos que la kuta_diferencial añadiendo el mapa:

```
%metodo de integración ruge-kuta
pose(:,k+1)=kuta_diferencial_mapa(t(k),pose(:,k),h,conduccion,MAPA);
```

A continuación, se muestra una captura de la ejecución:

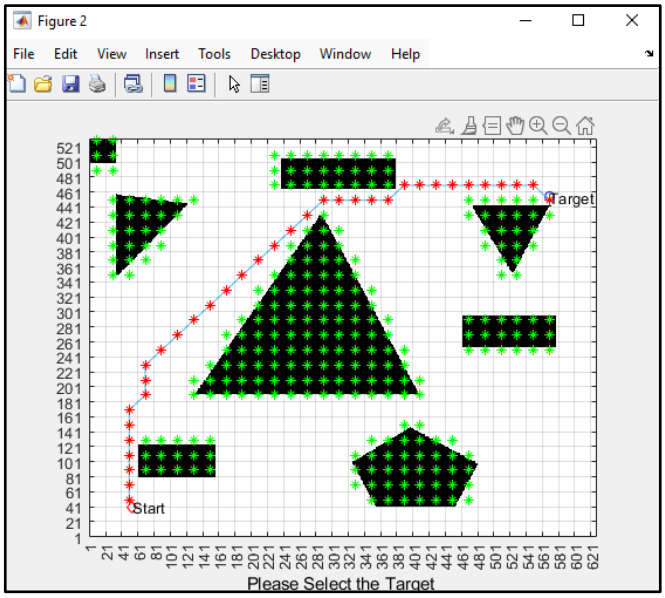


Por otro lado, para poder evitar el choque con los objetos que forman la imagen anterior, es necesario un algoritmo de búsqueda de manera que devuelva la mejor ruta si existe entre un punto inicial y otro final sin pasar por encima de ningún objeto, para ello se ha proporcionado una función denominada A_estrella que recibe el mapa y una variable delta, que para invocar la función se ejecuta el siguiente fragmento de código, donde se muestra una figura con una serie de puntos formados por cuadrículas y podemos seleccionar entre dos puntos para que nos devuelva la mejor ruta:

```
clear all
clc
%Carga el fichero  BMP
MAPA = imread('.\cuadro4.bmp');
%Transformación para colocar correctamente el origen del Sistema de
%Referencia
MAPA(1:end,:,:)=MAPA(end:-1:1,:,:);
%Tamaño de las celdas del grid
delta=20;  % el tamaño de cuadricula, si es muy pqueña puede ocurrir que la
%Llamada del algoritmo
Optimal_path=A_estrella(MAPA, delta); % optimal_path es un array que contie

%Dibujo de la ruta
plot(Optimal_path(:,1), Optimal_path(:,2),'*r')
```

Delta define el tamaño de cada cuadrícula, en este caso se ha usado 20, pero según el tamaño puede ocurrir que la navegación no sea segura si el valor de delta es muy bajo, o al contrario, que no haya eficiencia al encontrar caminos entre dos puntos si es alto, el Optimal_path devuelve un array con las coordenadas x e y de las posiciones de la ruta entre el punto inicial y el final.

El resultado de la ejecución se representa en la siguiente figura, donde hemos seleccionado un punto start y otro target.



Finalmente, para el desarrollo de la actividad se ha proporcionado otro fichero A_estrella_modificado_2 de forma que le pasamos el mapa, delta, posiciones de los puntos que queremos que nos devuelva la ruta entre ellos.

Se ha añadido al código de la primera parte de esta actividad, la invocación de la función, que se hace de la siguiente manera:

```
%Llamada del algoritmo a*
delta=25;
Optimal_path=A_estrella_modificado_2(MAPA,delta,pose0,posef);
```

Además, se ha modificado el código de forma que la función funcion_spline_cubica_varios_puntos recibe como argumentos xc e yc que esta formadas por la concatenación de la configuración de la posición inicial del robot, posición de despegue, posiciones de los puntos de la ruta devueltos por el a_estrella, punto aterrizaje y configuración de la posición final:

```
ds=0.1;
xc=[pose0(1) posicion_despegue(1) Optimal_path(:,1)' posicion_aterriza(1) posef(1)];
yc=[pose0(2) posicion_despegue(2) Optimal_path(:,2)' posicion_aterriza(2) posef(2)];
camino=funcion_spline_cubica_varios_puntos(xc,yc,ds)';
```

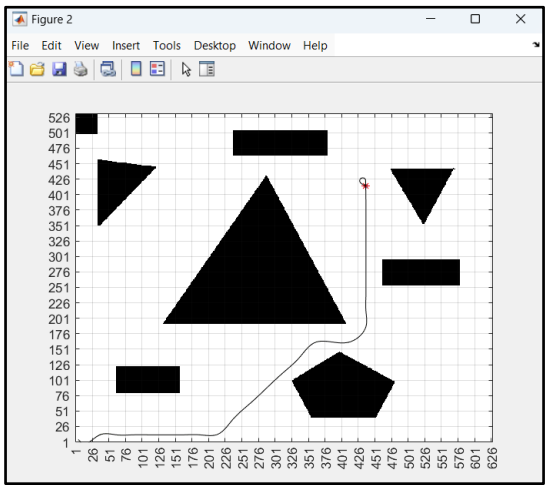
Para la ejecución se ha modificado el fichero de version2 de la actividad 6, donde las configuraciones de las posiciones y las invocaciones al fichero Actividad_7_mapa_v0 son las siguientes:

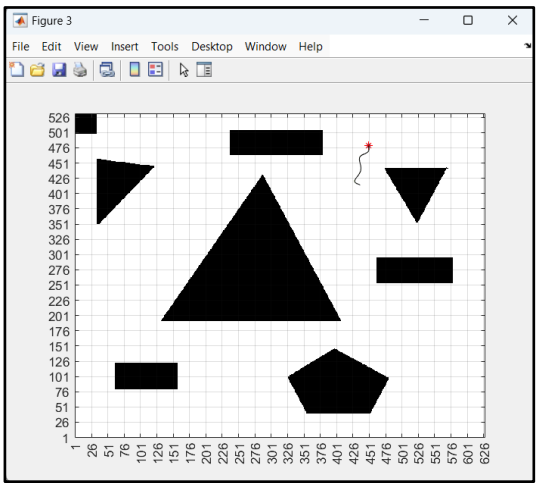
Editor - C:\Users\wadad\OneDrive - UNIVERSIDAD DE HUELVA\UHU\Repositorio\4\1-Cuatrimestre\RO\Acti

Actividad_7_v0_mapa.m | llamada_script_A7.m

```
1
2   clc, close all, clear all
3
4
5   pose0=[5; 5; -pi/4];
6   posef=[437; 415; -pi/2];
7
8
9   direccion=1;
10
11  Actividad_7_v0_mapa
12
13
14  pose0=pose(:,end);
15  posef=[450; 480; -pi/2];
16
17  pose=[];
18  direccion=-1;
19
20  Actividad_7_v0_mapa
```

En las siguientes figuras, se observa el resultado de la ejecución del código anterior:



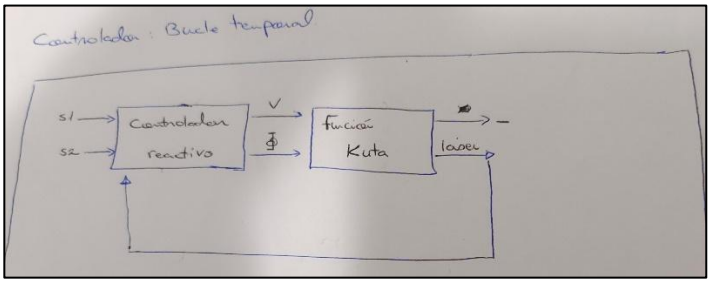


8. Descripción de la Actividad 8

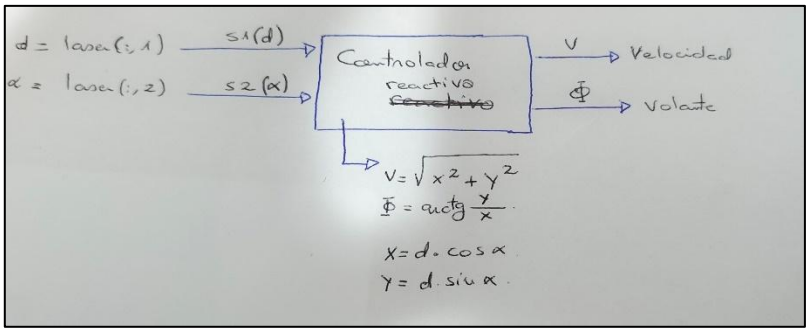
Esta actividad se trata de desarrollar e implementar una estrategia de control reactivo para un robot móvil simulado, en la que se fusiona las medidas obtenidas de un sensor de proximidad (LIDAR), lo que permite al robot determinar la percepción de su entorno y reaccionar en tiempo real a los cambios evitando obstáculos y adaptándose dinámicamente al escenario, esta actividad se divide en dos partes (Controlador reactivo con sensor laser, con seguimiento de trayectoria planificada).

8.1. Control Reactivo con el sensor Láser:

Esta parte implementa un controlador reactivo que permite que el robot mediante la percepción del sensor laser, determina la velocidad y el ángulo de dirección del robot evitando choque con los obstáculos que forman un mapa, en la siguiente figura se muestra el esquema de nuestro controlador:



Donde el controlador reactivo está formado por el siguiente esquema:



Para el desarrollo del se ha proporcionado una serie de ficheros que contiene el mapa o el entorno, función kuta que se encarga de actualiza la nueva posición y orientación del vehículo, y el script laser_reactivo que se divide en las siguientes partes:

- Configuración inicial: definimos el mapa que forma nuestro entorno, ajustamos las dimensiones de los parámetros del coche y inicializamos el vector x0 que contiene las posiciones, el ángulo de orientación y la velocidad del robot:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%  Inicialización de las variables fundamentales
% NO TOCAR ESTA PARTE DEL CODIGO
%-------------------------------------------------
%dimensiones del coche
ancho = 10;
largo = 16;
MAPA = imread('.\cuadro_nuevo3.bmp');
volante=0;
velocidad=0;
x0=[450; 185;0; 0]
```

- Simulación de los elementos que forman la representación gráfica:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%  ELEMENTOS DE REPRESENTACIÓN GRÁFICA
% NO TOCAR ESTA PARTE DEL CODIGO
%-------------------------------------------------
%crea las figuras
 scrsz = get(0,'ScreenSize');

 figure('Position',[1 scrsz(4)*5/30 scrsz(3)*25/30 scrsz(4)*25/30])
  axHndl=gca
  set(axHndl,'Userdata',1,'Drawmode','fast', ...
      'Visible','on','NextPlot','add','Userdata',1)
  axis([-10 1034 -10 778]);
   image(MAPA);
for numero=1:(2*8+2)
    handles_coche(numero) = plot(0,'Erasemode','xor');
    hold on
end
for numero=1:10
    handles_plots(numero) = plot(0,'Erasemode','xor');
    hold on
end

handle_vector = plot(0,'Erasemode','xor','LineWidth', 2.0);
```

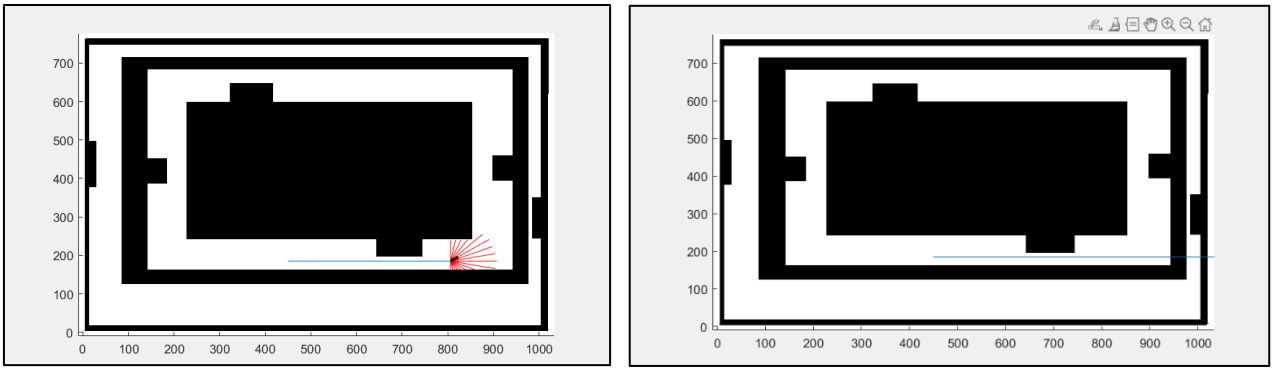
- Inicialización de los elementos que componen el estado del sistema y las medidas del láser:

```
%--------------------------------------------------------------------
%Da valores inciales al estado y a las medidas del laser
[x(:,k+1),laser]=kuta(t(k+1),x,h,0,0,handles_plots,handles_coche,handle_vector,[45 45],MAPA);
```

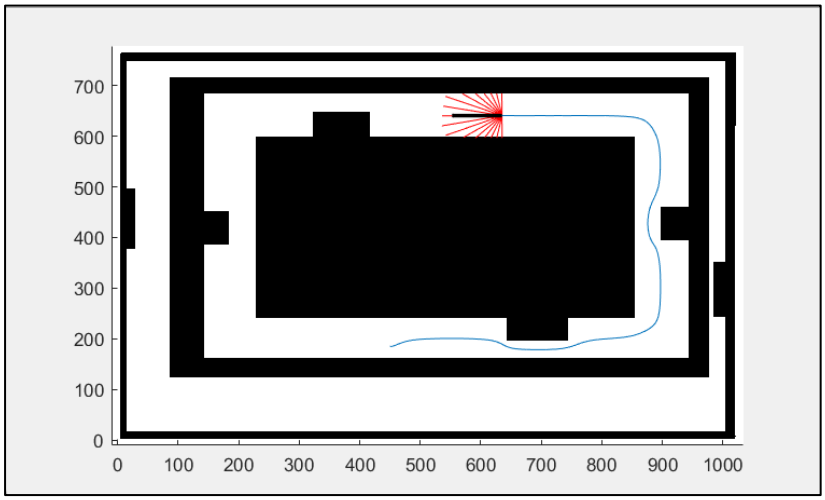
- Buce temporal: se encarga de calcular los componentes reactivos a partir de las mediciones del laser o sea la distancia y el ángulo de orientación, después se controla el movimiento del robot estableciendo que la velocidad sea proporcional al modulo del vector que forman las componentes anteriores de manera que si el valor de la velocidad es muy pequeño existe más probabilidad de chocar con algún obstáculo, y el ángulo de vehículo se determina mediante la multiplicación de una contante por el arco tangente de la componente_y entre la componente_x, finalmente, actualizamos las medidas del sensor laser:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
while (t0+h*k) < tf
   %actualización
   k=k+1;
   componentes_x = sum(laser(:,1).*cos(laser(:,2)));
   componentes_y = sum(laser(:,1).*sin(laser(:,2)));
   %%%%%%%%%%%%%%%%%%%%%%;%%%%%%%%%%%%%%%%%%%%
   % Aquí hay que definir el controlador
   %-------------------------------------
   velocidad=sqrt(componentes_x^2+componentes_y^2);  % es proporcional al vector para no pasar de velo
   volante= 5*atan2(componentes_y,componentes_x);  %angulo de conduccion
   %-------------------------------------
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   %Limitación del ángulo de conducción
   if volante>pi/4 volante=pi/4;
   end
   if volante<-pi/4 volante=-pi/4;
   end
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   %metodo de integración ruge-kuta, representación gráfica y simulacion del laser
   [x(:,k+1),laser]=kuta(t(k),x,h,volante,velocidad,handles_plots,handles_coche,handle_vector ...
       ,[componentes_x componentes_y],MAPA);
end
```

En la siguiente figura se muestra el comportamiento del robot siendo los componentes reactivos (x e y) constante a 15 y 10 respectivamente, y la velocidad a 70:



Mientras que la trayectoria del robot cuando las componentes (x e y) y los parámetros del robot son proporcionales a las medidas del sensor es la siguiente:



8.2. Controlador reactivo con Seguimiento de trayectoria planificada:

Esta parte consiste en añadir al comportamiento del robot con el controlador reactivo anterior , el seguimiento de una trayectoria planificada usando la función funcion_spline_cubica_varios_puntos generada en las actividades anteriores, de manera que recibiendo las coordenadas x e y que forman la configuración del punto inicial, la posición de despegue e aterriza, y la configuración del punto objetivo, para ello se hacemos los mismo cálculos que en la actividad 6, aplicando las funciones pure_presuit y mínima_distancia:

- Para el cálculo del camino se ha añadido el siguiente código:

```
%---------------------------------------------
% Cálculo del camino mediante una spline
%---------------------------------------------
pose0=[x0(1); x0(2); x0(3)];
posef=[800; 500; -pi/4];
punto_paso=[600 300];
%Definición de la spline
%ditancia de despegue y aterriza.
dd=50;
da=dd;
posicion_despegue=[pose0(1)+(dd*cos(pose0(3))) pose0(2)+(dd*sin(pose0(3)))];
posicion_aterriza=[posef(1)-(da*cos(posef(3))) posef(2)-(da*sin(posef(3)))];
%Definición del polígono
xc=[pose0(1) posicion_despegue(1) punto_paso(1)  posicion_aterriza(1) posef(1)];
yc=[pose0(2) posicion_despegue(2) punto_paso(2) posicion_aterriza(2) posef(2)];
ds=1; %distancia entre puntos en cm.
camino=funcion_spline_cubica_varios_puntos(xc,yc,ds)';
```

- El bucle temporal está compuesto por los siguientes fragmentes:

Primero, definimos la configuración del controlador reactivo vistos en la parte anterior, para ello, actualizamos los valores de las variables componentes_x, componentes_y, la velocidad reactiva e el volante:

```
while (t0+h*k) < tf
   %actualización
   k=k+1;
 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 % Aquí hay que se define el controlador reactivo
 %-----------------------------------------------
 componentes_x=sum([laser(:,1).*cos(laser(:,2))]);
 componentes_y=sum([laser(:,1).*sin(laser(:,2))]);
 modulo=sqrt(componentes_x.^2+componentes_y.^2);
 coseno=componentes_x/modulo;

 velocidad_reactiva(k)=1*modulo;
 volante_reactivo(k)=8*atan2(componentes_y,componentes_x);
```

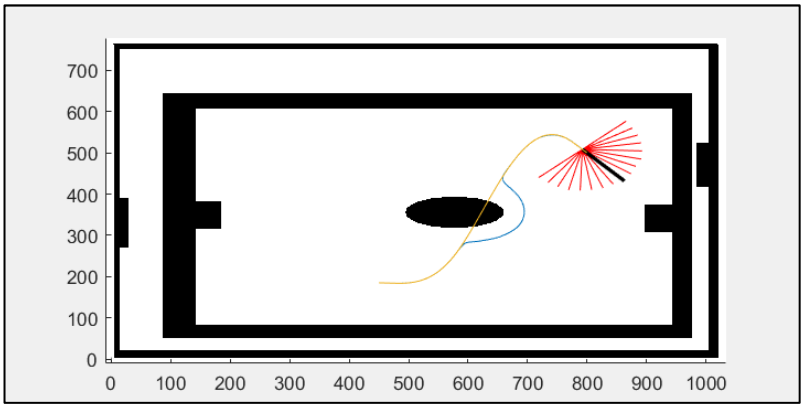
Después, configuramos el controlador pure_persuit que permite al robot seguir una trayectoria planificada ajustando la dirección en función de la posición del punto final, los cálculos son los mismos que los realizados en la actividad 5, finalmente, actualizamos los valores de la velocidad e el volante del robot de forma que sean proporcional al punto final, así conseguimos que la velocidad se vaya disminuyendo al cercarse al punto final:

```
 %-----------------------------------
 % Controlador p_pursuit
 %-----------------------------------
orden_minimo= minima_distancia_new(camino, [x(1,k) x(2,k)]);
look_ahead=10;
if (orden_minimo+look_ahead)<length(camino)
    punto=[camino(orden_minimo+look_ahead,1) camino(orden_minimo+look_ahead,2)];
else
    punto=[camino(end,1) camino(end,2)];
end
delta= (x(1,k)-punto(1))*sin(x(3,k))-(x(2,k)-punto(2))*cos(x(3,k));
LH=sqrt((x(1,k)-punto(1))^2+(x(2,k)-punto(2))^2);
rho=2*delta/LH^2;
 volante_tracking(k)=atan(largo*rho);

velocidad_traking(k)=5*sqrt((x(1,k)-camino(end,1))^2+(x(2,k)-camino(end,2))^2);
```

Por otro lado, utilizamos el factor coordinador para combinar el control reactivo y el pure_presuit, permitiendo al robot decidir si es mejor evitar un obstáculo o seguir la trayectoria:

```
%-----------------------------------
% Combinación de comportamientos
%-----------------------------------
%coordinador(k)=1;
coordinador(k)=coseno; % para conduccion planificada

%volante(k)=(coordinador(k)*volante_tracking(k))+((1-coordinador(k))*volante_reactivo(k));
volante(k)=(coordinador(k)*volante_tracking(k))+13*((1-coordinador(k))*volante_reactivo(k));
%velocidad(k)=(coordinador(k)*velocidad_traking(k))+((1-coordinador(k))*velocidad_reactiva(k));
velocidad(k)=0.125*(coordinador(k)*velocidad_traking(k))+((1-coordinador(k))*velocidad_reactiva(k));
```

A continuación, se muestra el comportamiento del robot en un controlador de seguimiento de trayectoria planificado:



Otra implementación del controlador reactivo con seguimiento de trayectoria planificada se trata de diseñar un nuevo mapa, y ajustar el comportamiento del robot, para lo que se ha diseñado los dos siguientes mapas mediante la aplicación Paint:

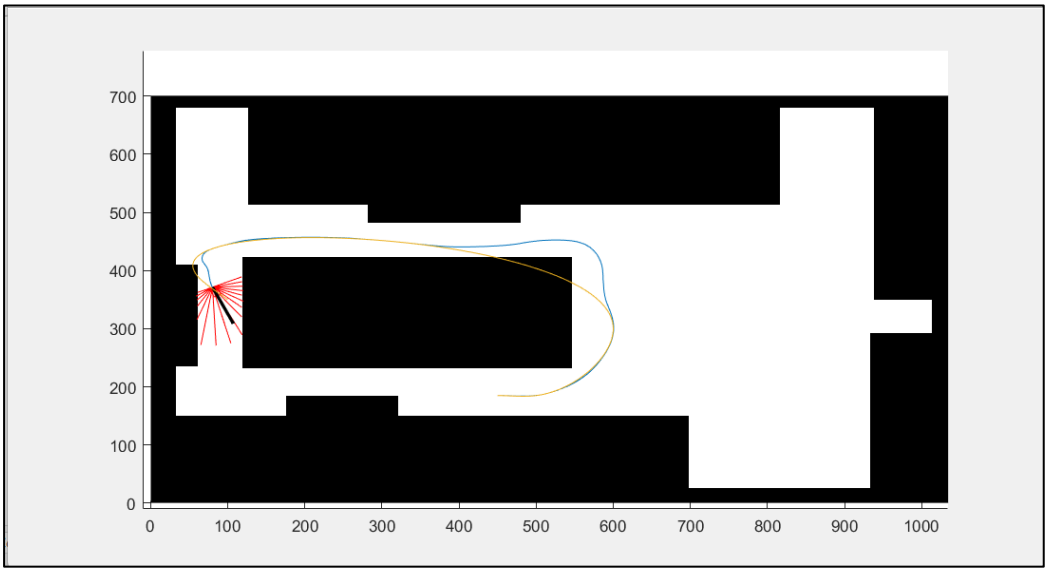
Mapa_nuevo.bmp

Mapa_2.bmp

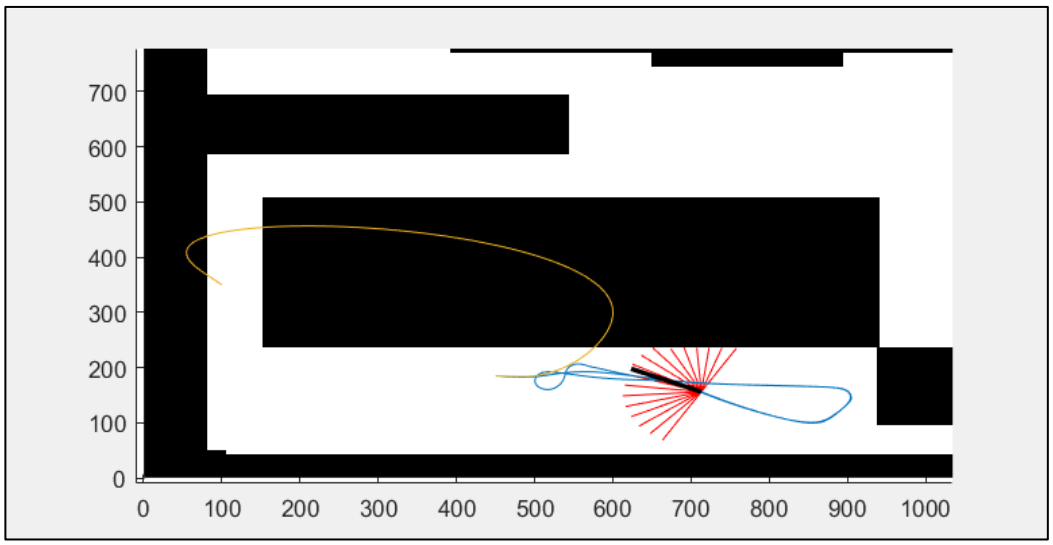
Se ha modificado la configuración del punto final, para comprobar mejor la trayectoria:

```
pose0=[x0(1); x0(2); x0(3)];
posef=[100; 350; -pi/4];
punto_paso=[600 300];
```

Finalmente, en las siguientes capturas se observa la trayectoria que sigue el robot en los dos mapas generados anteriormente, la trayectoria está bien ya que evita los obstáculos, el fallo existe en que al cargar la imagen se observa de color lila, lo he intentado solucionar, pero no sé porque lo hace:



Mapa_nuevo.bmp



Mapa_2.bmp

9. Enlaces

Los siguientes enlaces nos llevan a las carpetas que corresponden al código desarrollado en clase y casa de todas las actividades, las carpetas están separadas por proyectos donde están diferenciados por el nombre que corresponde a la fecha del día en la que ha sido impartida en clase o se ha empezado:

ACTIVIDAD1: https://universidadhuelva-my.sharepoint.com/:f:/g/personal/uadad_sidelgaum_alu_uhu_es/EjRuEghAvitNqmjOYtyLbRUB45H1p3h1wMBwfxKprEPQCw?e=3aWfxw

ACTIVIDAD2: https://universidadhuelva-my.sharepoint.com/:f:/g/personal/uadad_sidelgaum_alu_uhu_es/EmQrwk6OCCNJsrCPGxGgY2oBM2is2Pr585ktkgh9nZ5RQw?e=ocSIqy

ACTIVIDAD3: https://universidadhuelva-my.sharepoint.com/:f:/g/personal/uadad_sidelgaum_alu_uhu_es/EmH3T6VbyEBJjvh2kqs4hpoBcvVUIc9qi1oQxdGo8IsBCA?e=WSZiFd

ACTIVIDAD4: https://universidadhuelva-my.sharepoint.com/:f:/g/personal/uadad_sidelgaum_alu_uhu_es/Eld71QaUBPdDpkONZcpPcwYBhf5F5O6D-L1qCYMYgPNG-g?e=MuDLRI

ACTIVIDAD5: https://universidadhuelva-my.sharepoint.com/:f:/g/personal/uadad_sidelgaum_alu_uhu_es/Esrg2damYUFEhUzi_KR5PbgB7kVCDpg7q6Ihq03pVzJ-mg?e=MnSGUE

ACTIVIDAD6: https://universidadhuelva-my.sharepoint.com/:f:/g/personal/uadad_sidelgaum_alu_uhu_es/EuhTCycZwahDjjYeUTQAm5EBn-3Hf8W1j1iS-w0wBANqZw?e=81XCkn

ACTIVIDAD7: https://universidadhuelva-my.sharepoint.com/:f:/g/personal/uadad_sidelgaum_alu_uhu_es/ErCz5mXY5VtAkd0d4rrYyr4BFayjdpMQCGRAS-QqW74rkA?e=iHWtab

ACTIVIDAD8: https://universidadhuelva-my.sharepoint.com/:f:/g/personal/uadad_sidelgaum_alu_uhu_es/EjeNHQJdZZ1OimYV405FRUQBfTNG-PHGILbc8ggprmPyNw?e=zRGez6

ACTIVIDAD EVALUABLE: Actividad_Evaluable