ROBÓTICA

Trabajo de evaluación de la asignatura
Robótica Curso 2021

Manuel Tejada Guzmán
manuel.tejada724@alu.uhu.es

Índice

Índice de ilustraciones

Robótica

En este curso de robótica vamos a crear un robot capaz de identificar si existen objetos delante de él, llegando incluso a crear un mapa del lugar por donde pasa y esquivando los objetos impuestos.

Al comienzo de la asignatura el profesor no recomendó guardar el id de nuestro robot, lo podrás encontrar en el siguiente enlace:

- *ID Robot:*
  - id: 001653819d8f
  - brick hw: v0.70
  - brick fw: v1.04h

Cada Pieza

1. *Cabeza*
2. *Cuerpo*

1. Cabeza

Motor de movimiento

Empezamos el experimento comprobando que leíamos correctamente el giro de la rueda que posteriormente usaremos para el movimiento de la cabeza. Este movimiento lo hacemos inicialmente con la mano.

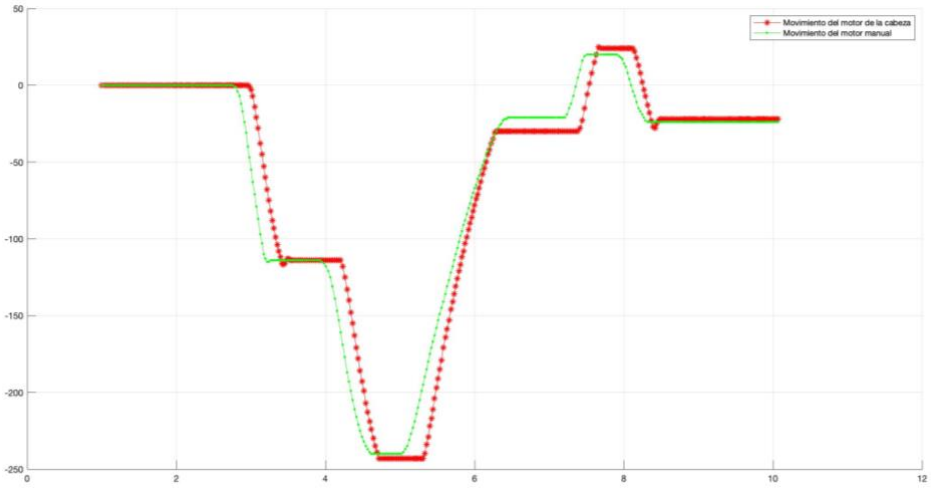

Ilustración 1: Movimiento del motor a mano con K=0.3

Vídeo demostrativo:
https://github.com/manuTGrt/robotica/blob/main/videos/mover_cabeza_con_motor_manual.gif

Posteriormente, movemos la cabeza unos grados indicados, en los que vemos que, variando la ganancia, la cabeza se movía de forma inestable, llegando a la conclusión de que un buen valor de ganancia para que no sea inestable sea 0.3.

Luego, intentamos mover la cabeza, cambiando la referencia de los grados indicados previamente a un valor variable, el cual variamos con la rueda, teniendo en cuenta el valor de la ganancia para ajustarlo a su mejor valor, en mi caso 0.3.

Esto lo podemos ver con la siguiente gráfica.

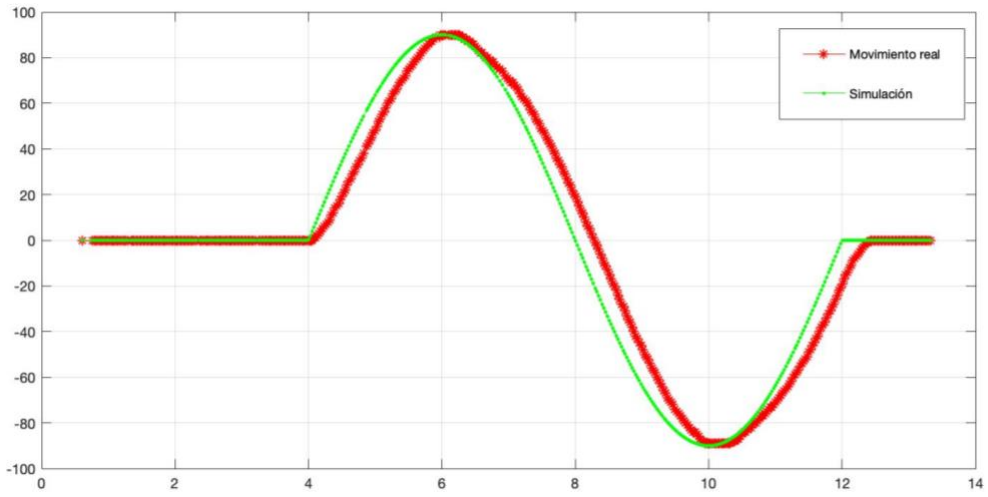

Ilustración 2: Movimiento de cabeza con K=0.3

- En color verde tenemos el movimiento que debería seguir el motor en una simulación en la que el movimiento sería perfecto.
- En color rojo tenemos el movimiento real que ha seguido el motor, con la ganancia en 0.3, como vemos, es casi perfecto.

Sensor sonar 👀

Este es el sensor que irá leyendo los obstáculos que se encuentre, para poder esquivarlos sin sufrir un accidente.
Lo primero que hicimos fue hacer el dibujo del robot, haciendo que el dibujo moviera la cabeza de manera simulada.

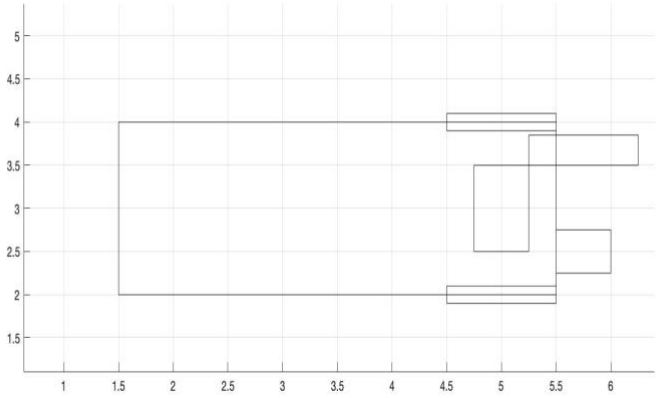

Ilustración 3: Dibujo del robot

Posteriormente hacemos que se dibuje una simulación de lo que sería la lectura del sonar, a una distancia fija con un giro de la cabeza.

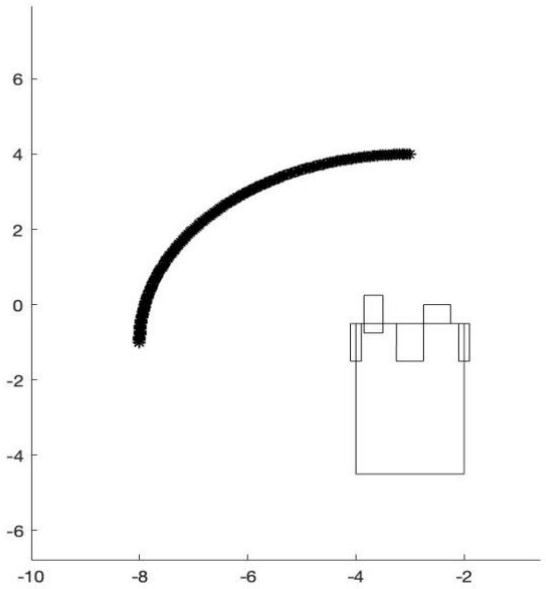

Ilustración 4: Dibujo del robot con lectura sonar

Luego lo implementamos en el robot y comprobamos que gira la cabeza y va leyendo los obstáculos que se va encontrando. En las pruebas muestro cómo lo hace.

2. Cuerpo

El cuerpo del robot es su cerebro, el que controla todos los motores y sensores de cada parte del robot.

Montaje

El montaje del robot se realiza con piezas lego, con las que poco a poco vamos dándole forma para así acoplar todas y cada una de las partes.

El profesor nos ha dejado un vídeo para que el montaje del robot sea más rápido y sencillo para nosotros.

Vídeo de cómo se monta el robot:
http://www.legoengineering.com/the-harvester-a-quick-ev3-robot-build/

La cabeza del robot la hemos modificado de sitio para que esté centrada y pueda moverse.
También hemos añadido un pulsador en el frontal, para que, si el sensor sonar no detecta ningún objeto, éste se pulse y pueda retroceder y buscar otro camino.

Puesta en funcionamiento

Para poner en funcionamiento todo el robot, el profesor ha puesto a nuestra disposición una carpeta:

- ***Carpeta***

Ésta contiene los siguientes elementos:

- maquina_estado_basica.m: Contiene la programación del robot en su mayoría, para que vaya cambiando de estado según la situación en que se encuentre.
    - Al principio de este fichero limpiamos todas las variables necesarias, luego declaramos e inicializamos las variables necesarias, además de iniciar los motores del robot.
    - Los diferentes estados son:
        - 1- Marchando para delante.
        - 2- Parando.
        - 3- Girando la cabeza con sonar.
        - 4- Girando sobre sí mismo.
        - 5- Marcha atrás.
    - Al final del fichero manda el comando "para", que ejecutará la función "para" para parar por completo el robot.
- para.m: Contiene la programación que se encarga de parar el robot. En caso de tener algún fallo, podemos ejecutarlo desde la línea de comandos y pararemos manualmente el robot.
- Signal_reading_odo.m: Contiene la programación que se encarga de leer las señales del sensor sonar y de los encoders de los motores de la cabeza y las ruedas para evitar oscilaciones en los procesos de control del giro de la cabeza y las ruedas, las consultas al estado de los motores de tracción solo se hacen en los estados necesarios.
    - Si está en el estado 3:

        - Significa que está girando la cabeza y, por tanto, no lee los encoders de las ruedas pero sí el de la cabeza.
    - Si no está en el estado 3:
        - Significa que la cabeza no está girando y, por tanto, no lee el encoder de la cabeza pero sí los de las ruedas.
- Traction_motor_control.m: Es el encargado de mandar las señales de control a los motores, controlando la velocidad máxima a la que se mueven en cada uno de los sentidos.
- WiFi_connection.m: Es el encargado de realizar la conexión mediante wifi del programa MatLab con el robot.
- calculo_odometria.m: Es la función que se encarga de calcular la odometría del robot teniendo en cuenta la distancia entre las ruedas y el radio de ellas. Ésta devuelve un vector con los ángulos calculados.
- calculo_referencia.m: Ésta se encarga de calcular la referencia que tiene que seguir la cabeza.
- mapa.dat: Es el fichero que generamos con la lectura del sonar, para formar un mapa de los obstáculos que ha ido encontrando y así ser conscientes de por qué gira el robot en los casos necesarios.
- pintar_robot_v2.m: Esta función es la que se encarga de pintar el robot y actualizar su posición cuando éste se mueve.
- referencia_cabeza.m: Es la función que devuelve el ángulo que tiene que girar la cabeza según la amplitud, el tiempo, el periodo y el desfase que le indiquemos por parámetros.

Ejecutando las pruebas

Como menciono anteriormente, lo primero que hicimos fue mover la cabeza, después hicimos que la cabeza reconociera los obstáculos, posteriormente movíamos el robot sin tener en cuenta la cabeza, luego movimos el robot y, cuando la cabeza encontraba un obstáculo, éste se paraba y giraba la cabeza, moviendo el robot después 90º a la izquierda, y, por ultimo, al girar la cabeza generaba un mapa de los obstáculos que se encontraba y decidía hacia qué lado girar, pudiendo así completar un recorrido.

También hicimos que el robot siguiera un camino generado mediante una spline, de manera que hicimos dos caminos, uno corto y otro largo en el que se quedaba aparcado.

Mover cabeza y lectura sonar

Para mover la cabeza ejecutamos el siguiente script, en el que posteriormente fuimos modificando diferentes parámetros para que se comportara de la forma que nos interesase.

```
clear all
clc
%conexión con lego, sensor y motores
myev3 = legoev3('usb');
mytouchsensor = touchSensor(myev3);
motor_A=motor(myev3,'A');
motor_B=motor(myev3,'B');

%inicio de motores
start(motor_A);
start(motor_B);

%reseteo de encoders
resetRotation(motor_A);
resetRotation(motor_B);

%boton = readTouch(mytouchsensor);
t(1)=0;
giro(1)=readRotation(motor_B);
grados=90;
referencia(1)=grados;
error(1)=referencia(1)-giro(1);
k=0.12;
i=0;

asterisco=animatedline('Marker','*','Color','r');
ref=animatedline('Marker','.','Color','g');
tstart=tic;
tiempo(1)=0;

while readTouch(mytouchsensor)==0
end

while readTouch(mytouchsensor)==1
end

grid on;
while readTouch(mytouchsensor)==0
    i=i+1;
    giro(i)=readRotation(motor_B);
    referencia(i)=grados;
    error(i)=referencia(i)-giro(i);
```

```
    %definicion del controlador
    controlador=k*error(i);

    %actuación sobre el motor
    power=int8(controlador);
    if power>100
        power=100;
    else
        if power <-100
            power=-100;
        end
    end

    %actuación de los motores
    motor_B.Speed=power;

    %pintando gráfica
    tiempo(i)=toc(tstart);
    y(i)=double(readRotation(motor_B));
    x(i)=double(referencia(i));
    addpoints(asterisco,tiempo(i),y(i));
    addpoints(ref,tiempo(i),x(i));

end
drawnow
stop(motor_B);
```

Primero la movemos, como vemos en este código, unos grados concretos, en este caso 90º.

Posteriormente modificamos las líneas necesarias expuestas en el siguiente código para moverla mediante el giro manual de otro motor.

```
%grados=90;
referencia(1)=readRotation(motor_A);
```

Luego hicimos que moviera la cabeza de un lado a otro y por último al centro.

```
%con esto inicializamos las variables
grados=90;
tiempo(1)=0;
desfase=4;
periodo=8;
referencia(1)=referencia_cabeza(grados,tiempo(1),desfase,periodo);

%grafica que seguirá la cabeza
t=0:0.01:periodo+desfase+desfase;
for j=1:length(t)
    angulo_cabeza(j)=referencia_cabeza(grados,t(j),desfase,periodo);
end
plot(t,angulo_cabeza)
```

En el bucle de funcionamiento, tendremos que indicar también que gire tomando como referencia la función generada.

```
tiempo(i)=toc(tstart);
referencia(i)=referencia_cabeza(grados,tiempo(i),desfase,periodo);
error(i)=referencia(i)-giro(i);
```

Aquí muestro una demostración de cómo la cabeza gira

https://github.com/manuTGrt/robotica/blob/main/videos/movimiento_cabeza.gif

Es entonces cuando implementamos la lectura del sonar, donde mapa es el vector en el que se encuentran todos y cada uno de los objetos dibujados.

```
%leo la distancia
distancia(i) = double(readDistance(mysonicsensor))*100;

%muevo la cabeza del robot y apunto la distancia a la que se encuentran
%los objetos
mapa=pintar_robot_v2(0,0,0,double(readRotation(motor_B))*pi/180,SR_robot,SR_cabeza,double(distancia(i)),mapa);
```

Así reconoce el entorno mientras está moviendo la cabeza

https://github.com/manuTGrt/robotica/blob/main/videos/robot_mueve_cabeza_y_lee_sonic.gif

Moviemiento del robot

Para mover el robot ya hemos hecho una máquina de estados en la que según el estado en el que se encuentre el robot podrá hacer diferentes cosas como:

1. marchando para adelante
2. parando
3. girando cabeza con sonar
4. girando sobre si mismo
5. Marcha atrás

Con esta máquina de estados comprobamos que, mientras no haya problemas, el robot seguirá andando recto, en cambio, cuando se encuentre un obstáculo, éste pasa al estado 2, el cual lo para y comprueba que, si no se ha chocado ni hay un obstáculo cerca, vuelve al estado 1 para avanzar hacia adelante, si ha detectado un obstáculo cerca pasa al estado 3, en el que gira la cabeza para decidir el giro necesario, y, si se ha chocado, pasa al estado 5, el cual da marcha atrás y vuelve al estado 2 y decidir en qué situación está.

Las comprobaciones en cada uno de los estados son:

```
switch estado
      case 1 %andando hacia delante
```

```
          %if (readDistance(Sonar)<stop_distance) %si la distancia e
s menor que 35 para
           if (distancia(i)<stop_distance) || (readTouch(Detecta_col
ision)==1)%si la distancia es menor que 35 o choca para
               estado=2; %transición de estado de paro
               transicion=i; %indice que marca el inicio del estado 2
          end

      case 2 %parando
          if (vel==0)
               if (distancia(i)>stop_distance) && (readTouch(Detecta_
colision)==0)
                    estado=1; %la transición a estado marcha hacia del
ante
                    transicion=i; %indice que marca el inicio del esta
do 1
               elseif (distancia(i) < stop_distance)
                    estado=3; %transición a estado girando cabeza
                    transicion=i; %indice que marca el inicio del esta
do 3
                    desfase=t(transicion)+1;
               else %Si hay choque
                    estado=5; %la transición a estado marcha hacia del
ante
                    transicion=i; %indice que marca el inicio del esta
do 1
               end
          end

      case 3 %girando cabeza
          if(t(i)>(desfase+Periodo+1.5)) %espera a que pasen el desf
ase+periodo mas 2s
               Power_cabeza=0; %para el giro de la cabeza
               estado=4; %la transición a estado girando robot
               transicion=i; %indice que marca el inicio del estado 4
          end

      case 4 %girando robot
          if(t(i)-t(transicion)>t_giro)
               estado=2;
               transicion=i;
          end
          %estado=5; %la transición a estado marcha atrás
          %transicion=i; %indice que marca el inicio del estado 5

     case 5 %marcha atrás
          if (t(i)-t(transicion)>t_marcha_atras)
               estado=2; %transición a estado girando cabeza
               transicion=i; %indice que marca el inicio del estado 2
```

```
        end

end
```

Las indicaciones para los movimientos de los motores son:

```
switch estado
      case 1 %andando hacia delante
      %establece los valores de control
          vel=20;
          Power1=vel;
          Power2=vel;

      case 2 %parando
        %establece los valores de control
          vel=0;
          Power1=vel;
          Power2=vel;

      case 3 %girando cabeza
          %cálculo de la referencia
              referencia(i)=referencia_cabeza(Amplitud,t(i),Periodo,
desfase);
          %cálculo del error
              error_cabeza(i)=(referencia(i)-giro_cabeza(i));
          %ganancia del controlador proporcional
              k=0.35;

          %Definición del controlador
              controlador=k*error_cabeza(i);

          %Actuación sobre el motor
              Power_cabeza=int8(controlador);

      case 4 %girando sobre si mismo
          vel=20;
          Power1=vel;
          Power2=-vel;

      case 5 %andando hacia atrás
          %establece los valores de control
         vel=-20;
         Power1=vel;
         Power2=vel;

end

%---------------------
%Manda los comandos de control a los motores
```

```
%---------------------
Traction_motor_control;
```

La demostración de funcionamiento

https://github.com/manuTGrt/robotica/blob/main/videos/robot_sigue_recorrido.gif

Una vez que hemos hecho que el robot pueda resolver un camino de forma autónoma, programamos la forma de que el robot pudiera seguir un camino generado.

Primero hicimos que girara durante un tiempo determinado de manera simulada.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Simulación del movimiento de un robot móvil
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all
clc

%j=1;

global l
global camino
global pose
global punto
%cargamos el camino
camino=load('camino.dat');

global l
l=3.5; %distancia entre rudas delanteras y traseras, tambien definid
o en modelo

%Estos son distintos ejemplos de Condiciones iniciales

%pose0=[15; 15; -pi/2];

%pose0=[30; 30; 0];

pose0=[0; 0; pi/2];

%tiempo inicial
t0=0;

%final de la simulación
%tf=100;
tf=15;

%paso de integracion
h=0.1;
```

```
%vector tiempo
t=0:h:tf;
%indice de la matriz
k=0;

%inicialización valores iniciales
pose(:,k+1)=pose0;

t(k+1)=t0;

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

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %valores de los parámetros de control
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

 % estas son las variables de control
 velocidad=5;
 volante=-0.1416;

 %ambas se combinan en la variable conducción
 conduccion=[velocidad volante];

 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

 %para representar el punto onjetivo sobre la trayectoria

 punto=[30 30];

%metodo de integración ruge-kuta y representación gráfica

pose(:,k+1)=kuta(t(k),pose(:,k),h,conduccion);

end
```

Demostración de giro
https://github.com/manuTGrt/robotica/blob/main/videos/robot_giro_tiempo.gif

Después, lo pusimos en el robot.

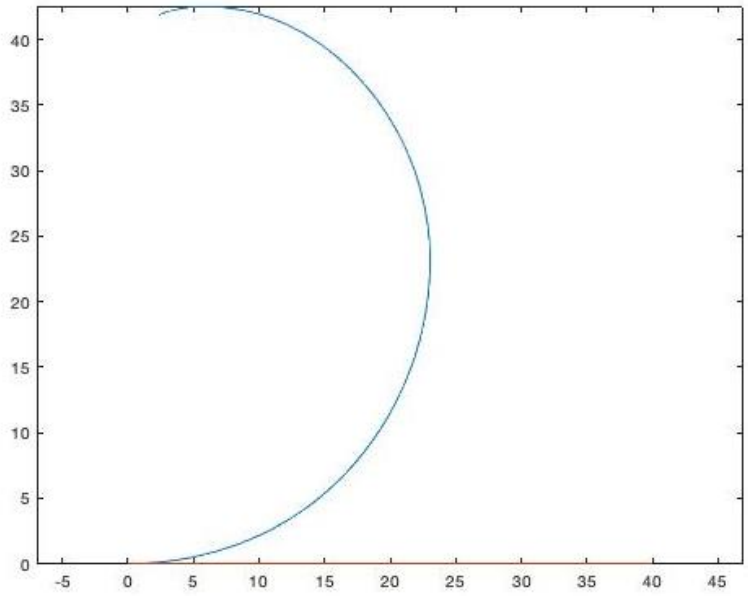

Ilustración 5: Robot gira media circunferencia

Vídeo de demostración
https://github.com/manuTGrt/robotica/blob/main/videos/media_circunferencia.gif

También hicimos que siguiera una línea recta.

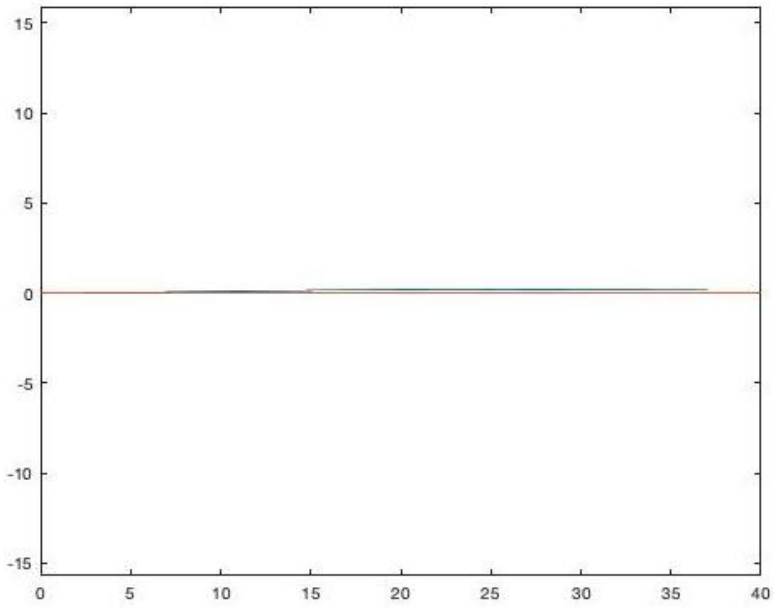

Ilustración 6: Robot va en línea recta

Vídeo de demostración
https://github.com/manuTGrt/robotica/blob/main/videos/linea_recta.gif

Luego, le indicamos que haga un cuarto de circunferencia hacia un lado y otro cuarto al otro lado, con lo que realiza la forma de una "S".

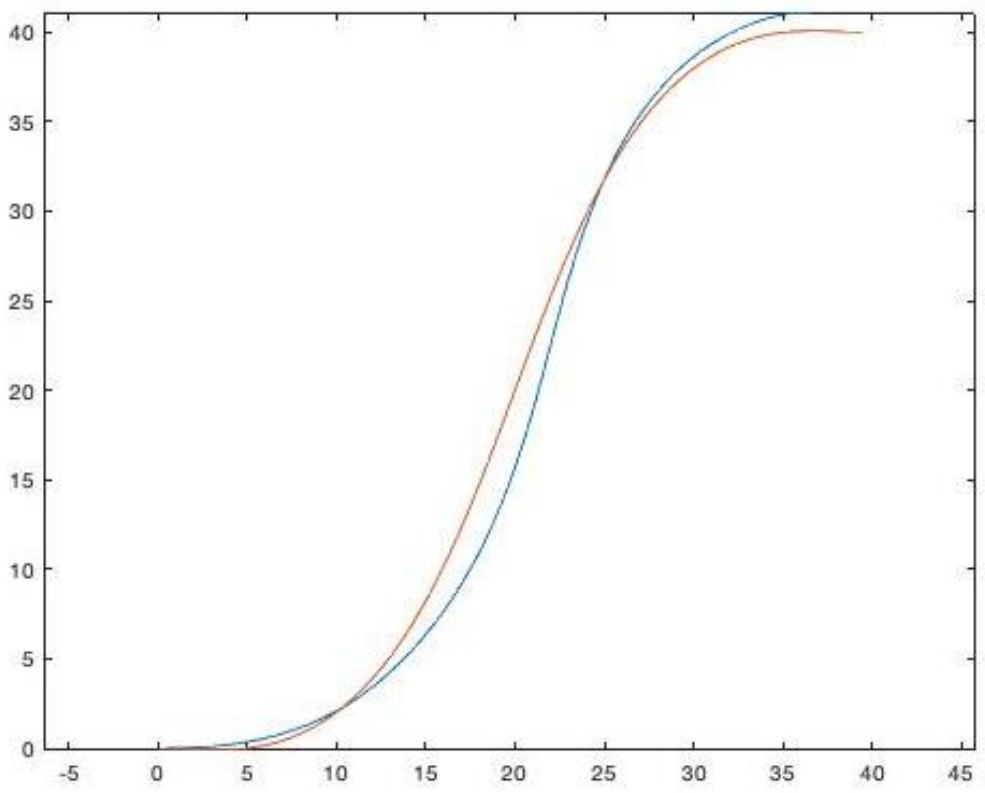

Ilustración 7: Robot va en forma de "S"

Vídeo de demostración
https://github.com/manuTGrt/robotica/blob/main/videos/formaS.gif

Con esto, conectamos el robot mediante wifi e hicimos dos pruebas en la mesa central de la clase.
Primero un recorrido corto.
https://github.com/manuTGrt/robotica/blob/main/videos/corto.gif
Luego, un recorrido largo donde quedaba aparcado al final.
https://github.com/manuTGrt/robotica/blob/main/videos/largo.gif
Para esto hicimos el siguiente código.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Controla la convertgencia del robot EV3 a un punto mediante una es
trategia de control geométrico.
% utiliza los encoders de los motores para estimar calcular la odome
tría
% utiliz un  switch, para comenzar y terminar  la rutina
%
% Utiliza los script:
% Traction_motor_control_laboratorio.m; Signal_reading_odo_path_foll
owing.m;  Para.m.
%
% Utiliza las funciones: calculo_calculo_odometria.m;
```

```
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 29/11/2020. FGB.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clc
%clear all

%variables control angulo del robot
    clear yaw
    clear x y theta
    clear giro_derecho giro_izquierdo

    global radio_rueda
    global l %distancia entre ruedas

    %mi_Robot = legoev3('USB')

%----------------------------------------
% Variables para la representación gráfica
%----------------------------------------

% %Crea los sistemas de referencia del robot y de la cabeza para la
% %representación utilizando la función pinta_robot_v2
%     SR_robot = hgtransform;
%     SR_cabeza = hgtransform('Parent',SR_robot);
%----------------------------------------

%------------------------------
%Valores para la odometría
      l=5.8;
      radio_rueda=2.85

%---------------------------


% Declaración de sensores
    Detecta_colision = touchSensor(mi_Robot,1); %Switch conectado al
puerto 1.
    Pulsador = touchSensor(mi_Robot,2); %Switch conectado al puerto
2.
    Sonar = sonicSensor(mi_Robot); %definición del sonar

% Declaración de los motores
    motor_cabeza = motor(mi_Robot,'A') %motor de la cabeza
    motor_izquierdo = motor(mi_Robot,'B') %Motor izquierdo
    motor_derecho = motor(mi_Robot,'C') %Motor_derecho

%Activacion de los motores
```

```
    start(motor_cabeza);
    start(motor_izquierdo);
    start(motor_derecho);

%inicializa velocidad de motores
    Power1=0;
    Power2=0;
    Power_cabeza=0;


%reset del encoder de motores
    resetRotation(motor_cabeza);
    resetRotation(motor_izquierdo);
    resetRotation(motor_derecho);

%indice inicial
    i=1;

%valores iniciales de los encoders
    giro_derecho(i)=0;
    giro_izquierdo(i)=0;


%--------------------
% Valores iniciales
%--------------------
%tiempo inicial
    t(i)=0;
    x(i)=0;
    y(i)=0;
    theta(i)=0;
    yaw(i)=0;


%comienza el bucle
disp('pulsa el bumper para comenzar')

%camino_x=0:1:40;
%camino_y=0*camino_x;
%camino=[camino_x' camino_y'];

%definir camino
dd=5;
da=dd;

posicion_despegue=[x(i)+(dd*cos(theta(i))) y(i)+(dd*sin(theta(i)))];
posicion_aterriza=[100-(da*cos(pi/2)) 115-(da*sin(pi/2))];

xc=[0 posicion_despegue(1) 30 60 posicion_aterriza(1) 100];
yc=[0 posicion_despegue(2) 30 60 posicion_aterriza(2) 115];
```

```
ds=1; %distancia entre puntos en cm.
camino=funcion_spline_cubica_varios_puntos(xc,yc,ds)';
%-------------------------------


while(readTouch(Pulsador)==0)
end

while(readTouch(Pulsador)==1)
end


disp('comienza el bucle')
tf=60;
%referencia tiempo inicial
    tstart = tic;

while  (readTouch(Pulsador)==0) & (t(i)<tf)

        i=i+1; %indice global
        t(i)= toc(tstart); %tiempo global del bucle
    %---------------------
    %lectura señales y calculo del heading
    %-------------------------------

        Signal_reading_odo_path_following;

      %----------------------------
       %Calcula Odometría
      %--------------------------
    %calculo odometria
        [x(i) y(i) theta(i)]=calculo_odometria(giro_derecho,giro_izq
uierdo,x,y,theta,i);
    %para controlar el giro
        yaw(i)=theta(i)*180/pi;

%----------------------------
% Control Geométrico

%para converger a un punto
%punto más cercano
 orden_minimo= minima_distancia_new (camino, [x(i) y(i)]);

 %para representar el punto onjetivo sobre la trayectoria
 %hay que corregir el Look_ahead
 Look_ahead=10;
 seguir=orden_minimo+Look_ahead;
 if(orden_minimo+Look_ahead>length(camino))
     seguir=length(camino);
```

```
 end
 punto=[camino(seguir,1), camino(seguir, 2)];

 %punto=[0 40];

 delta= (x(i)-punto(1))*sin(theta(i))-(y(i)-punto(2))*cos(theta(i));

 LH=sqrt((x(i)-punto(1))^2+(y(i)-punto(2))^2);

 rho=2*delta/LH^2;

%Control proporcional de la velocidad
 Kp=1.1;
 %final=[camino(end,1) camino(end,2)]; %para converger al final del
camino
 final=punto; %para converger a un punto
 Distance_to_end=sqrt((x(i)-final(1))^2+(y(i)-final(2))^2);
 velocidad=Kp*Distance_to_end;

 if velocidad>17
     velocidad=17;
 end

 if Distance_to_end<3
     break
 end

 %-------------------------------------
 % modelo Inverso
 %-------------------------------------

    velocidad_derecha=velocidad*(1+l*rho)/radio_rueda;

    velocidad_izquierda=velocidad*(1-l*rho)/radio_rueda;

%-----------------------------------------------------
% Cversión de velocidad a potencia
    potencia_equivalente=7.0;

    Power1_a(i)=velocidad_derecha*potencia_equivalente;
    Power2_a(i)=velocidad_izquierda*7.0;

    Power1=Power1_a(i);
    Power2=Power2_a(i);

      %---------------------
        %Manda los comandos de control a los motores
      %-------------
        Traction_motor_control_laboratorio;
```

```
end %del while

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Para motores y cierra sensores
%-------------------------------
 Para;

% plot(t,giro_izquierdo)
% hold on
% plot(t,giro_derecho)

velocidad_izquierda
giro_izquierdo(end)/t(end)

velocidad_derecha
giro_derecho(end)/t(end)

figure

plot(x,y)
hold on;
plot(camino(:,1),camino(:,2));

axis equal

%axis([0 90 0 90]);
```

Simulación de resolución de camino y aparcamiento.

Por último, hicimos una simulación que, cuando se tiene un mapa determinado, la función "A_estrella" se encarga de generar un camino para que el robot no se choque, después se genera el camino con una spline y lo recorre, al llegar al final genera un camino para aparcar y aparca.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Simulación del movimiento de un robot móvil
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all
clc

j=1;

%joy = vrjoystick(1);

global l
global radio_rueda
global camino
global pose
```

```
global pose2
global punto
%cargamos el camino
%camino=load('camino.dat');

%x=10:0.5:80;
%y=10*ones(size(x));
%xd=70;
%yd=80;
%ds=1;
%phi=atan(yd/xd);

%x=0:ds*cos(phi):xd;
%y=0:ds*sin(phi):yd;

%camino=[x' y'];

l=3.5; %semidistancia entre rudas delanteras y traseras, tambien def
inido en modelo
radio_rueda=1;

%Carga el fichero  BMP

MAPA = imread('micuadro.bmp');

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

delta=50;

%genera la ruta óptima
Optimal_path=A_estrella(MAPA, delta);

%Condiciones iniciales
pose0=[Optimal_path(1,1); Optimal_path(1,2); pi/2];
posef=[Optimal_path(end,1); Optimal_path(end,2); 3*pi/2];
%pose0=[camino(end,1); camino(end,2); pi/2]; %el último punto es el
mismo
%que el primero
%pose0=[10;10;pi];

%Condiciones iniciales con camino A_estrella
%pose0=[camino(1,1); camino(1,1); 0];
%posef=[camino(end,end); camino(end,end); 0];

%definir camino
dd=5;
```

```
da=dd;

posicion_despegue=[pose0(1)+(dd*cos(pose0(3))) pose0(2)+(dd*sin(pose
0(3)))];
posicion_aterriza=[posef(1)-(da*cos(posef(3))) posef(2)-(da*sin(pose
f(3)))];

xc=[pose0(1) posicion_despegue(1) Optimal_path(2:end-1,1)' posicion_
aterriza(1) posef(1)];
yc=[pose0(2) posicion_despegue(2) Optimal_path(2:end-1,2)' posicion_
aterriza(2) posef(2)];

ds=1; %distancia entre puntos en cm.
camino=funcion_spline_cubica_varios_puntos(xc,yc,ds)';
%camino=A_estrella(MAPA, 50);
%--------------------------------


t0=0;

%final de la simulación
tf=70;

%paso de integracion
h=0.1;
%vector tiempo
t=0:h:tf;
%indice de la matriz
k=0;

%inicialización valores iniciales
pose(:,k+1)=pose0;

t(k+1)=t0;

while (t0+h*k) < tf,

    %actualización
    k=k+1;

 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

 %punto más cercano
 orden_minimo= minima_distancia_new (camino, pose(1:2,k));

 %para representar el punto onjetivo sobre la trayectoria
 %hay que corregir el Look_ahead
 Look_ahead=20;
 seguir=orden_minimo+Look_ahead;
 if(orden_minimo+Look_ahead>length(camino))
```

```
     seguir=length(camino);
 end
 punto=[camino(seguir,1), camino(seguir, 2)];

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

    %V0=10;
    Distancia_al_final=sqrt((pose(1,k)-camino(end,1))^2 + (pose(2,k)
-camino(end,2))^2);

    V0=1*LH;
    if(V0>50)
        V0=50;
    end

    W=V0*rho;
    %----------------------------------------
    %V0=-V0;
    %--------------
    %Modelo inverso
    velocidad_derecha=(1/radio_rueda)*(V0+W*l);
    %velocidad_derecha=velocidad*(1+1*rho)/radio_rueda;
    velocidad_izquierda=(1/radio_rueda)*(V0-W*l);
    %velocidad_izquierda=velocidad*(1+1*rho)/radio_rueda;
    %--------------

    conduccion=[velocidad_derecha velocidad_izquierda];

%metodo de integración ruge-kuta

%pose(:,k+1)=kuta_diferencial(t(k),pose(:,k),h,conduccion);
pose(:,k+1)=kuta_diferencial_mapa(t(k),pose(:,k),h,conduccion,MAPA);

end

%PROGRAMO EL APARCAMIENTO
%genero el camino del aparcamiento
Optimal_path2=[posef(1) posef(2);posicion_aterriza(1) posicion_aterr
iza(2);375 75;325 75];
camino=funcion_spline_cubica_varios_puntos(Optimal_path2(:,1)',Optim
al_path2(:,2)',ds)';

%ahora, la posición inicial de este bucle es la final del anterior
pose0=[posef(1); posef(2); posef(3)];

%t0=0; Esta variable no cambia
```

```
%final de la simulación -- amplio en 3 segundos el tiempo de finaliz
ación
tf=tf+30;

%paso de integracion -- Da igual si lo dejo, ya que no cambia
%h=0.1;
%vector tiempo -- Tengo que aumentarlo ya que el tiempo de finalizac
ión ha
%cambiado
t=0:h:tf;
%indice de la matriz -- Lo comento porque tene que continuar con el
mismo
%valor del bucle anterior
%k=0;

%inicialización valores iniciales
pose(:,k+1)=pose0;

t(k+1)=t0;

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

 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

 %punto más cercano
 orden_minimo= minima_distancia_new (camino, pose(1:2,k));

 %para representar el punto onjetivo sobre la trayectoria
 %hay que corregir el Look_ahead
 Look_ahead=10; %para el aparcamiento necesitamos más precisión en e
l seguimiento de la línea
 seguir=orden_minimo+Look_ahead;
 if(orden_minimo+Look_ahead>length(camino))
     seguir=length(camino);
 end
 punto=[camino(seguir,1), camino(seguir, 2)];

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

    Distancia_al_final=sqrt((pose(1,k)-camino(end,1))^2 + (pose(2,k)
-camino(end,2))^2);

    V0=1*LH;
    if(V0>20)
```

```
        V0=20;
    end

    W=V0*rho;
    %Modelo inverso
    velocidad_derecha=(1/radio_rueda)*(V0+W*l);
    %velocidad_derecha=velocidad*(1+1*rho)/radio_rueda;
    velocidad_izquierda=(1/radio_rueda)*(V0-W*l);
    %velocidad_izquierda=velocidad*(1+1*rho)/radio_rueda;
    %--------------

    conduccion=[-velocidad_derecha -velocidad_izquierda]; %le cambio
el signo a los motores para que el robot vaya marcha atrás

%metodo de integración ruge-kuta

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

Demostración de la simulación

https://github.com/manuTGrt/robotica/blob/main/videos/simulacion_A_estrella.gif

Construido con

La herramienta utilizada para la programación del robot es MatLab, en general he usado la versión 18, pero en casa para algunas simulaciones he usado la versión 20.

- *MatLab* - Versión 18

Para la conexión wifi, el profesor nos ha dejado un adaptador wifi de tamaño mini. También nos ha construido la programación de éste.

El robot ha sido construido con piezas lego, utilizando el ladrillo ev3.

Autor

Este proyecto ha sido realizado por:

- **Manuel Tejada Guzmán** - *Todo mi GitHub* - *manuTGrt*

Con ayuda del profesor Fernando Gómez Bravo, que ha llevado el seguimiento y resuelto todas las dudas sobre este proyecto.
