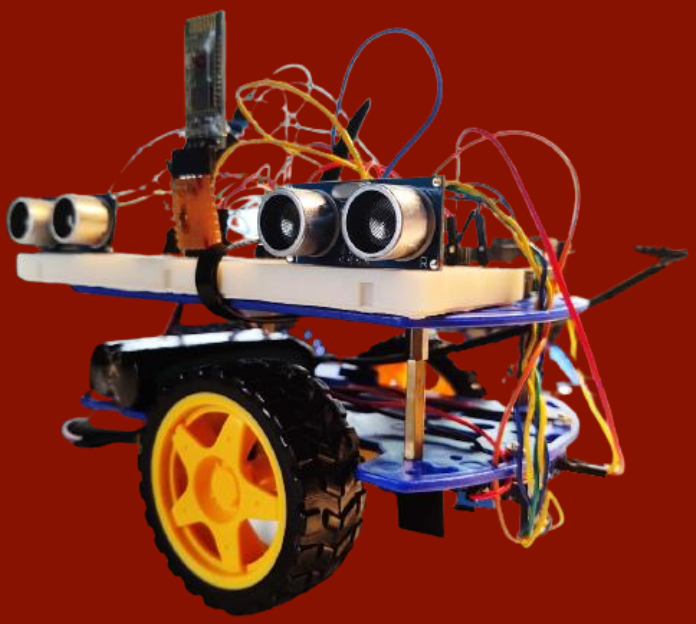
Escuela Técnica Superior Ingeniería
Universidad de Sevilla

Laboratorio de Robótica
Curso: 22/23
4º GIERM



**PEDRO LÓPEZ JAPÓN**
**GRUPO 09**

# ROBOT MÓVIL

# ÍNDICE

# I. INTRODUCCIÓN

## PROPÓSITO

El objetivo de esta parte de la asignatura es construir y controlar un robot móvil mediante realimentación sensorial. Para conseguir el objetivo hemos debido tener en cuenta varios aspectos importantes como son: la búsqueda de especificaciones técnicas de los sensores y actuadores (ultrasonidos, encoders, puente H, módulo Bluetooth), los modelos cinemáticos de vehículos (distancia entre ruedas, radio de las ruedas, velocidades angulares de las ruedas, etc), diferentes diseños de programaciones y análisis de sistemas de control para robots móviles (P, PI, PID), la influencia de no linealidades en sensores (uso de filtros de Kalman para filtrar medidas) y actuadores, así como la influencia del tiempo de procesado en el bucle de control. La comunicación con el robot móvil ha sido mediante Bluetooth además de usar el puerto serie para cargar en la placa los diferentes programas realizados.

## CONEXIONADO

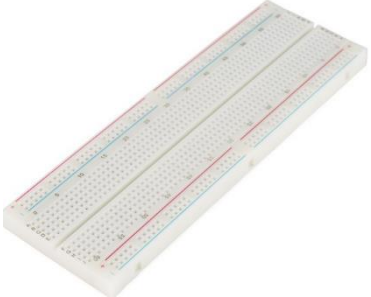
Para poder implementar todas las tareas hemos utilizado los siguientes componentes:

- Chasis del coche
- 2 motores
- Rueda loca y 2 ruedas
- Placa Arduino MEGA
- Placa de pruebas
- Batería de litio
- 2 sensores ultrasonidos HC-SR04
- Módulo con dos puentes H
- Módulo Bluetooth HC05
- 2 encoders
- Cables para conexiones





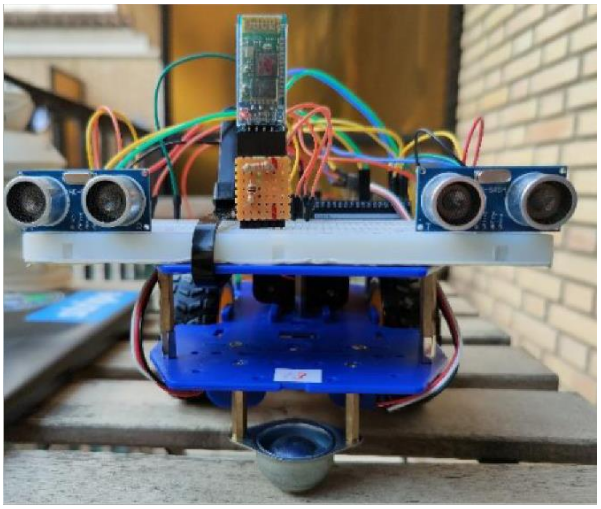


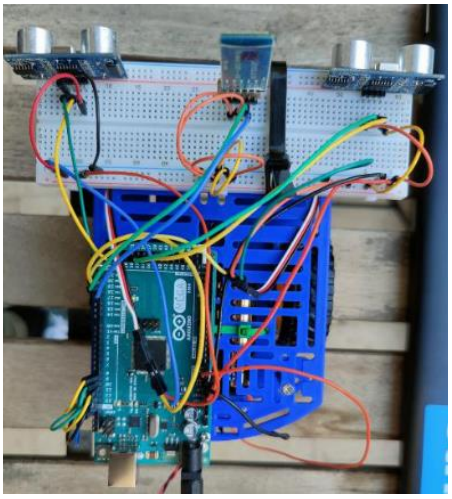


# II. FOTOS DE LOS MONTAJES

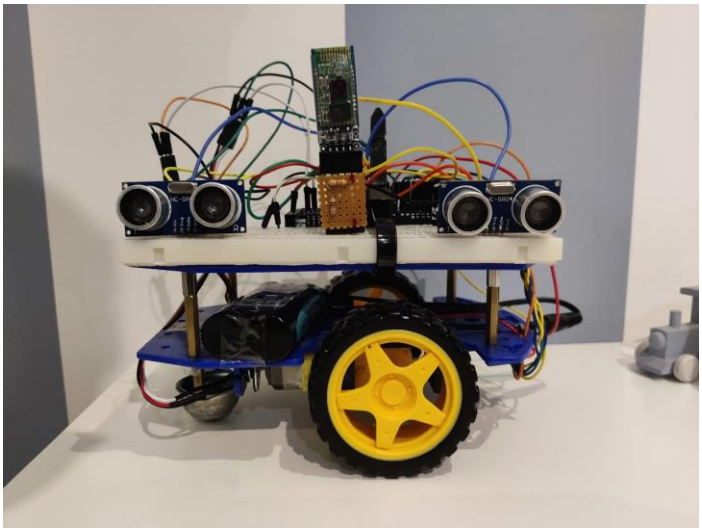
En este apartado podemos observar las fotos de los montajes individuales para cada modo en concreto.

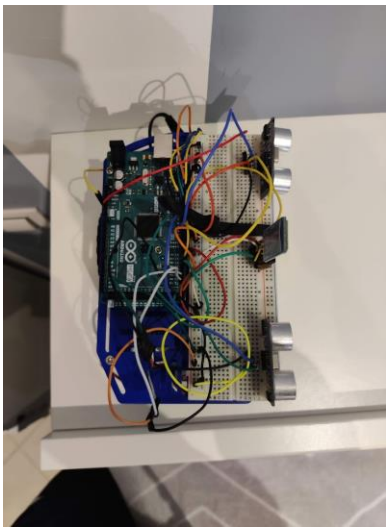
**MODO 1-2**





**MODO 3-4**





**MODO 5-6-7**


# III. MODOS DE FUNCIONAMIENTO

Antes de empezar a realizar los distintos modos previstos hemos hecho que el robot móvil ejecute los movimientos principales y que nos van a servir para todos los modos provenientes.
Lo primero que hemos hecho ha sido configurar todos los pines que vamos a usar:

```
int MotorDer1=13; //El pin 13 de arduino se conecta con el pin In1 del L298N
int MotorDer2=11; //El pin 11 de arduino se conecta con el pin In2 del L298N
int MotorIzq1=9; //El pin 9 de arduino se conecta con el pin In3 del L298N
int MotorIzq2=8; //El pin 8 de arduino se conecta con el pin In4 del L298N
int PWM_Derecho=12; //El pin 12 de arduino se conecta con el pin EnA del L298N
int PWM_Izquierdo=10; //El pin 10 de arduino se conecta con el pin EnB del L298N
```

Este conexionado se puede observar mejor en la siguiente tabla:

| PIN DE ARDUINO | PIN DE L298N | DEFINICION DEL PIN |
|---|---|---|
| 8 | IN4 (Motor Izq. Atrás) | MotorIzq2 |
| 9 | IN3 (Motor Izq. Avanza) | MotorIzq1 |
| 10 | EnB (PWM Izq.) | PWM_Izquierdo |
| 11 | IN2 (Motor Der. Atrás) | MotorDer2 |
| 12 | EnA (PWM Der.) | PWM_Derecho |
| 13 | IN1 (Motor Der. Avanza) | MotorDer1 |

Posteriormente se declaran los pines como salida para que se puedan mandar acciones y que se ejecuten los movimientos. Esta asignación se puede observar en el siguiente código:

```
pinMode(MotorDer1, OUTPUT);    pinMode(MotorDer2, OUTPUT);
pinMode(MotorIzq1, OUTPUT);    pinMode(MotorIzq2, OUTPUT);
pinMode(PWM_Derecho, OUTPUT);   pinMode(PWM_Izquierdo, OUTPUT);
```

Esta configuración la repetimos en todos los modos posteriores.

En este planteamiento inicial hemos hecho que se ejecuten todos los movimientos consecutivamente durante un tiempo de 2500ms.
Primero hemos hecho que avance, dándole un nivel alto a los pines de avanzar y un nivel bajo a los pines de retroceder de ambos motores. Además, le pasamos un PWM fijo de 100.

```
//Hacia delante
digitalWrite(MotorDer1,HIGH);
digitalWrite(MotorDer2,LOW);
digitalWrite(MotorIzq1,HIGH);
digitalWrite(MotorIzq2,LOW);
analogWrite(PWM_Derecho,100);
analogWrite(PWM_Izquierdo,100);
delay(2500);
```

Seguidamente hacemos que el robot pivote en el sentido antihorario, para lo que le damos un nivel alto a los pines de avance de la rueda derecha y de retroceso de la rueda izquierda, manteniendo los dos pines restantes a nivel bajo. Una vez pasado 2500ms, se pivota en sentido horario, adjudicando nivel alto a los

pines que antes estaban a nivel bajo, y viceversa. Ambos movimientos se ejecutan con un PWM fijo de 100.

```
//Pivote antihorario
digitalWrite(MotorDer1,HIGH);
digitalWrite(MotorDer2,LOW);
digitalWrite(MotorIzq1,LOW);
digitalWrite(MotorIzq2,HIGH);
analogWrite(PWM_Derecho,100);
analogWrite(PWM_Izquierdo,100);
delay(2500);
```

```
//Pivote horario
digitalWrite(MotorDer1,LOW);
digitalWrite(MotorDer2,HIGH);
digitalWrite(MotorIzq1,HIGH);
digitalWrite(MotorIzq2,LOW);
analogWrite(PWM_Derecho,100);
analogWrite(PWM_Izquierdo,100);
delay(2500);
```

Tras estos cinco segundos de pivotamiento, el robot recibe la orden de ir hacia atrás. Para ello ponemos a nivel alto los pines de retroceso de ambos motores y dejamos a nivel alto los de avance. En este caso, tras prueba y error, hemos fijado el PWM del motor derecho a 110 y el del motor izquierdo 100, para que realice un movimiento rectilíneo.

```
//Hacia atrás
digitalWrite(MotorDer1,LOW);
digitalWrite(MotorDer2,HIGH);
digitalWrite(MotorIzq1,LOW);
digitalWrite(MotorIzq2,HIGH);
analogWrite(PWM_Derecho,110);
analogWrite(PWM_Izquierdo,100);
delay(2500);
```

Por último, hicimos que el robot realizara un círculo. Para ello hemos activado los pines de avance y desactivado los de retroceso. Para que se ejecutara esta figura geométrica, fijamos el PWM derecho a 150 y el izquierdo a 100, haciendo que la rueda derecha vaya más rápido.

```
//CIRCULO
digitalWrite(MotorDer1,HIGH);
digitalWrite(MotorDer2,LOW);
digitalWrite(MotorIzq1,HIGH);
digitalWrite(MotorIzq2,LOW);
analogWrite(PWM_Derecho,150);
analogWrite(PWM_Izquierdo,100);
delay(10000);
```

Tras familiarizarnos con el robot y sus movimientos, pasamos a realizar los modos que se nos piden. Cabe destacar que estos movimientos los meteremos en funciones para simplificar los códigos.

## MODO 1

En este modo se ha implementado que el robot se pare frente a una pared a la distancia que se le pide por referencia Bluetooth sin importar la perpendicularidad.

La recepción de la referencia por Bluetooth la hacemos a través del puerto serie 2, ya que el uno lo reservamos para observar los resultados en la consola de Arduino. Para establecer la conexión se ha utilizado la aplicación “Serial Bluetooth Terminal” de Play Store.

Para poder medir la distancia hemos añadido los dos sensores de ultrasonido proporcionados junto con la librería <Ultrasonic.h> (archivo adjunto en el proyecto). Estos sensores obtienen la distancia según la cantidad de tiempo que tarde en recibir la señal desde que la envía. Hemos conectado un sensor en la parte frontal derecha conectado a los pines 6 y 7, y otro en la parte frontal izquierda conectado a los pines 4 y 5. Estos pines corresponden a Trigger y Echo, respectivamente.

Para el desarrollo de este modo, medimos con los sensores ultrasonidos la distancia del robot con la pared. Una vez obtenida esta información, saturamos la medida a 200 cm para un mejor control del PWM (evitando así picos indeseados) y sacamos la media de la distancia del sensor de la izquierda y de la derecha. Posteriormente calculamos el error proporcional como la distancia media menos la referencia, el error derivativo y el error integral, aunque este último no haga falta al haber usado un control PD. Hemos decido usar este control porque suaviza la respuesta. Las constantes del controlador la hemos obtenido de forma experimental.

En este primer modo hemos usado dos sensores, aunque con uno solo bastaba. Otro aspecto a destacar es que hemos usado un único control para ambas ruedas, ya que no nos importa la perpendicularidad del robot. Este control es el propio PWM que posteriormente le meteremos a las ruedas para que se realicen los movimientos necesarios.

Como hemos observado, la zona muerta de nuestro robot está cuando el PWM toma el valor de 70, por lo que se ha tenido que hacer un arreglo para que el robot siempre tenga la fuerza necesaria para vencer el rozamiento y moverse. Además, hemos saturado el PWM a 140 cuando se avance y a -80 cuando retroceda para que no se produzcan daños. Por último, se ha hecho que, cuando el error sea mayor que 2 el robot avance, y cuando el error sea menor que -2, el robot retroceda. Una vez se ha establecido en este rango, el robot debe pararse. Por tanto, observamos que el robot puede tener un error de ±3, al pararse, respecto de la referencia. Hemos decidido poner este error porque con la inercia que lleva, desde que se da la orden de paro hasta que realmente se pare, el robot sobrepasa la zona de paro y vuelve a corregir su posición, pudiendo esto llevar a un mal desarrollo de este modo.

La parte del código donde actúa el control es la siguiente:

```
distanciader = ultrader.read(CM)-1;
distanciaizq = ultraiz.read(CM)+1;
if(distanciader>200 || distanciaizq>200){
   distanciader=200;
   distanciaizq=200;
}
distanciamed=(distanciaizq+distanciader)/2;
error = distanciamed - (double)ref; //Error proporcional
cumError += error * (double)elapsedTime; //Error integral
rateError = error * (error-lastError)/(double)elapsedTime; //Error derivativo

pwmder = kp*error + ki*cumError + kd*rateError;

if(error>0){ //Arreglo de zona muerta
   pwmder=pwmder+70;
}
else if(error<0){
   pwmder=pwmder-70;
}
```

```
if(pwmder>140){ //Saturación de avance
   pwmder=140;
}
if(pwmder<-80){ //Saturación de retroceso
   pwmder=-80;
}
if(pwmder>0 && ref!=0 && error>2){
   avanza(abs(pwmder),abs(pwmder));
}
if(pwmder<0 && ref!=0 && error<-2){
   atras(abs(pwmder),abs(pwmder));
}
if (error < 3 && error > -3){
   para();
   error=0;
   rateError=0;
   cumError=0;
   lastError=0;
}
```

## MODO 2

En este modo lo que se nos pide es que, además de situarse el robo frente a una pared a la distancia que le digamos como referencia, lo haga lo más perpendicular posible a esta.

Para la realización de este modo nos basamos principalmente en la estructura del modo 1. En este caso, ya no usaremos la media de la distancia media entre los dos sensores, sino que utilizaremos cada una individualmente. Esto lo hacemos porque ahora hemos querido controlar cada una de las ruedas por separado para que se puedan corregir individualmente hasta quedar de forma perpendicular. Un aspecto importante a tener en cuenta es que hemos tenido que meter un filtro de Kalman para que se eliminen los ruidos y medidas indeseadas de los sensores.

En este caso también saturamos la medida de los sensores a 200 cm. Posteriormente calculamos los errores proporcional, integral y derivativo de ambos sensores, para poder hacer un control independiente para cada rueda. Una vez calculado el valor de estos controladores, se corrige la zona muerta, se saturan los PWM de avance y retroceso, a 140 y -80 respectivamente, y se empieza a actuar sobre las ruedas para que ejecuten los movimientos. De esta forma, si el resultado de los controladores sale positivo, se da la orden para avanzar y, por el contrario, si sale negativo, se ordena el retroceso. Cuando se obtenga la información de que los sensores tienen un error de ±1, se para el robot a la espera de una nueva referencia.

El código donde actúan los controles es el siguiente:

```
distanciaderbruto = ultrader.read(CM) - 1;
distanciaizqbruto = ultraiz.read(CM);
distanciader = kalman(distanciaderbruto);
distanciaizq = kalman(distanciaizqbruto);
```

```
if (distanciader > 200 || distanciaizq > 200) {
  distanciader = 200;
  distanciaizq = 200;
}

errorder = distanciader - (double)ref; //Error proporcional
errorizq = distanciaizq - (double)ref;

cumErrorder += errorder * (double)elapsedTime; //Error integral
cumErrorizq += errorizq * (double)elapsedTime;

rateErrorder = (errorder - lastErrorder) / (double)elapsedTime; //Error derivativ
o
rateErrorizq = (errorizq - lastErrorizq) / (double)elapsedTime;

pwmder = kp * errorder + ki * cumErrorder + kd * rateErrorder; //Salida del contr
olador
pwmizq = kp * errorizq + ki * cumErrorizq + kd * rateErrorizq;


if (errorder > 0 || errorizq > 0) { //Arreglo de zona muerta
  pwmder = pwmder + 70;
  pwmizq = pwmizq + 70;
}
else if (errorder < 0 || errorizq < 0) {
  pwmder = pwmder-70;
  pwmizq = pwmizq-70;
}

if (pwmder > 140 || pwmizq > 140) { //Saturación de avance
  pwmder = 140;
  pwmizq = 140;
}
if (pwmder < -80 || pwmizq < -80) { //Saturación de retroceso
  pwmder = -80;
  pwmizq = -80;
}
if (pwmder > 0 && ref != 0) {
  avanzader(abs(pwmder));
}
else if (pwmder < 0 && ref != 0) {
  atrasder(abs(pwmder));
}
if (pwmizq > 0 && ref != 0) {
  avanzaizq(abs(pwmizq));
}
else if (pwmizq < 0 && ref != 0) {
  atrasizq(abs(pwmizq));
}
```

```
if (abs(errorder) < 1) {
  parader();
  errorder=0;
  rateErrorder=0;
  cumErrorder=0;
  lastErrorder=0;
  pwmder=0;
}
if (abs(errorizq) < 1) {
  paraizq();
  errorizq=0;
  rateErrorizq=0;
  cumErrorizq=0;
  lastErrorizq=0;
  pwmizq=0;
}
```

## MODO 3

En este modo se pide que el robot avance paralelo a una pared. La distancia a la que se encuentra de la pared será la referencia que le indiquemos por Bluetooth, que en este modo en concreto es de medio metro. Para poder realizar este movimiento se ha de cambiar el montaje del robot, pasando de tener los dos sensores en la parte frontal a tenerlo en el lateral izquierdo del mismo.

Hemos tenido que introducir otro filtro de Kalman para usar uno para cada sensor, ya que al usar uno para ambos no se podía controlar correctamente el robot. Ahora, las medidas de los sensores ultrasonidos son usadas de forma distinta a los dos modos anteriores debido a que el control del movimiento hay que tratarlo de forma distinta. En este caso hemos usado un mismo control para dar el PWM necesario a las ruedas en cada momento.

En este modo hemos hecho que el error relativo sea la diferencia de las medidas de los sensores, es decir, le restamos la medida del sensor derecho a la medida del sensor izquierdo. Esto lo hacemos para ver si las medidas son iguales (paralelo a la pared) o no. Hemos establecido que se aplique el control (PD) cuando el error de las medidas de los sensores supere los ±1.5 cm. Cuando esto ocurra se ha saturado el error a ±4 para que el controlador no de picos de valores y se desestabilice el robot y, además, se hace que el robot gire a izquierda o derecha según el error sea negativo o positivo, respectivamente. Por último, si el error se encuentra en el rango de ±1.5 cm se da la orden de avanzar con un PWM fijo de 83.

Un aspecto a tener en cuenta es que se ha realizado el modo únicamente para que el robot avance, sin tener en cuenta la marcha atrás. Si queremos hacerlo de esta otra forma tendríamos que incluir una función con los mismos comandos que cuando hacemos retroceder al robot en el modo uno, y posteriormente llamar esta función en vez de la de avanzar.

La parte del código donde actúa el control se puede observar seguidamente:

```
distanciaderbruto = ultrader.read(CM);
distanciaizqbruto = ultraiz.read(CM);
distanciader = kalmander(distanciaderbruto);
```

```
distanciaizq = kalmanizq(distanciaizqbruto);

err = (double)distanciaizq - (double)distanciader; //calculo del error relativo e
ntre las dos medidas de los ultrasonidos
err_i += err * (double)elapsedTime; //error integral
err_d = err * (err-err_ant) / (double)elapsedTime; // error derivativo
pwm = kp*err + ki*err_i + kd*err_d; //salida del controlador

//Saturación de avance
if (pwm > 140) {
  pwm = 140;
}

if(err>=-1.5 && err<=1.5) //tomamos este margen para mantener el robot paralelo
{
  avanza(83);
  pwmizq=83;
  pwmder=83;
}
else if(err<-1.5) //el robot gira a la izquierda
{
  if(err>4 || err<-4){
    err=-4;
  }
  giroizq(abs(pwm));
  pwmder=95-abs(pwm);
  pwmizq=83;
}
else if(err>1.5) // el robot gira a la derecha
{
  if(err>4 || err<-4){
    err=4;
  }
  giroder(abs(pwm));
  pwmizq=95+abs(pwm);
  pwmder=83;
}
```

## MODO 4

En este modo se nos pide que el robot circule paralelo a una pared y se vaya adecuando a las referencias que se le transmiten por Bluetooth. De esta forma el robot se acercará o se alejará según le se le vaya indicando.

Para este modo vamos a usar como referencia la distancia media medida por los sensores ultrasonidos. Además, usaremos como error (‘desvío’ en el código), la diferencia entre estas dos distancias medidas. También es importante conocer el ángulo de desvío, es decir, el máximo ángulo que va a poder estar

girado el robot respecto a la pared sin cometer fallos en la lectura de los sensores ultrasónicos. Este ángulo lo sacamos haciendo la arcotangente del valor absoluto de la división del 'desvío' entre 13. Esta constante se corresponde a la distancia entre nuestros dos sensores de medidas. Por ultimo obtendremos la separación perpendicular del coche a la pared y el error entre la referencia y esta distancia mencionada.

```
if (elapsedTime >= 50) {
  //calculo las distancias
  distanciaderbruto = ultrader.read(CM);
  distanciaizqbruto = ultraiz.read(CM);
  distanciader = kalmander(distanciaderbruto);
  distanciaizq = kalmanizq(distanciaizqbruto);

  distancia_med=(distanciader+distanciaizq)/2;
  desvio=distanciader-distanciaizq;
  ang_desvio=atan(abs(desvio)/13); //Angulo de desvio sacado con el arcotangente
de       desvio entre separación de sensores (sen(ang)/cos(ang))
  distancia_real=distancia_med * cos(ang_desvio); //Separación del coche y la par
ed de forma perpendicular a la pared.
  err=ref-distancia_real;
```

Una vez obtenidos estos parámetros, ejecutamos una serie de órdenes y limitaciones para el correcto funcionamiento. Para esto hemos usado un control en cascada en el cual primeros observamos la posición del robot respecto la referencia y posteriormente controlamos el giro del robot y los movimientos a hacer.

Si el robot está lejos de la referencia, es decir, si el valor absoluto del error es mayor que 5 (hemos usado este error de 5 para que funcionase bien), la variable 'pwm' toma el valor del error, aunque se satura a 80 cuando este supera los ±80 centímetros. En este caso se realizará un control proporcional.
Pasamos a controlar el ángulo de giro. Si este es mayor que 10º, si la diferencia entre la medida del sensor derecho menos la del izquierdo es positiva, hacemos que se pare cuando queremos alcanzar una referencia cercana y que avance cuando el objetivo es la lejana. En cambio, si el sensor izquierdo esta más lejos de la pared que el derecho, hacemos que el robot se pare cuando queremos alcanzar una referencia lejana y que avance cuando queremos acercarnos a la pared.

```
if(abs(err) > 5 ){
  pwm=Kp1*err;
  if(pwm>80){
    pwm=80;
  }
  else if(pwm<-80){
    pwm=-80;
  }
  else if(ang_desvio>(0.1745)){
   if(desvio>0){
    if(distancia_real>ref){
       pwm=-80;
    }
    else{
       pwm=0;
    }
```

```
   }
   else{
     if(distancia_real>ref){
       pwm=0;
     }
     else{
       pwm=80;
     }
   }
 }
 avanza(80+pwm/4,75-pwm/4);
}
```

Si el robot está a una distancia de ±5 de la referencia, se usará un controlador PD para mantenernos recto. Saturaremos la variable 'pwm', que posteriormente se sumará o restará a un valor constante de 80 para el movimiento del robot, a ±10.
Por una parte, si la distancia media de los sensores es menor que la referencia y el ángulo del robot es mayor que 15º, si la diferencia entre las medidas de los sensores es positiva, se hace que la rueda derecha tenga un PWM mayor para que se siga cumpliendo la limitación del ángulo. Si la diferencia entre los sensores es negativa, se va haciendo un giro más lento.
Por otro lado, cuando la distancia media medida por los ejes es mayor que la referencia y el ángulo del robot supera los 15º, si el desvío es positivo se gira hacia la derecha, siendo este giro más brusco cuando la diferencia entre las medidas de los sensores es negativa.

```
else{
  err=desvio;
  err_d=desvio-desvio_ant;
  if(abs(desvio)>2){
    pwm=Kp*err+Kd*err_d;
  }
  if(pwm>10){
    pwm=10;
  }
  if(pwm<-10){
    pwm=-10;
  }
  if(distancia_med<ref){
    if(ang_desvio>PI/12){
      if(desvio>0){
        pwm=0;
      }
      if(desvio<0){
        pwm=10;
      }
      avanza(80+pwm/2,75);
      pwm_izq=80+pwm/2;
      pwm_der=75;
    }
  }
```

```
  else if(distancia_med>ref){
    if(ang_desvio>PI/12){
      if(desvio>0){
        pwm=10;
      }
      if(desvio<0){
        pwm=0;
      }
      avanza(75,80+pwm/2);
      pwm_izq=75;
      pwm_der=80+pwm/2;
    }
  }
}
```

Por último, podemos encontrar todas las funciones de los movimientos básicos del robot junto con los filtros de Kalman.

## MODO 5

En este modo volvemos a cambiar el montaje del robot, en el que eliminamos los sensores de ultrasonido e instalamos sensores de velocidad. Esto lo hacemos ya que el objetivo de este modo es realizar un control de velocidad angular de cada rueda para que estas respondan adecuadamente a los cambios de velocidad. Estos sensores de velocidad son los encoders, que detectan el movimiento de un disco, instalado en el robot, mediante cambios en la dirección del campo magnético de este. Al detectar este cambio, los encoders envían esta información en forma de corriente a los pines de interrupción del Arduino, que hace que se puedan contar los giros (en nuestro caso, los 'ticks').

Hemos conectado dos pines nuevos, en este caso de interrupción. El primero es el del encoder izquierdo en el pin 21, y el segundo es el del encoder derecho en el pin 20. Los conectamos a estos pines para que, posteriormente, cambie el valor de las variables ordenadas cada vez que ocurra una interrupción.

```
void countTicks_iz() {
  tick_iz++;
}
void countTicks_de() {
  tick_de++;
}
```

Para el cálculo de las revoluciones por minuto de las ruedas usamos una fórmula que hemos encontrado por internet cuya expresión general es la siguiente:

$$rpm = \frac{(60 * pulsos)}{resolucion}$$

Esta expresión es válida cuando el recalculado de las variables se hace cada segundo. En nuestro caso, como el recalculado lo hacemos cada 100 ms, tenemos que usar la siguiente formula:

$$rpm = \frac{(600 * pulsos)}{resolucion}$$

Estando los pulsos relacionados con las interrupciones, y siendo la resolución fija de 384.

Los errores en este modo los calculamos como la diferencia entre las revoluciones de referencia y la calculada de la forma anterior. El control que usamos en este modo es un proporcional en el que, viendo si los errores son positivos o negativos se suma o resta, respectivamente, un valor de 5 hasta llegar a la referencia pedida. Como venimos haciendo, se saturan los PWM de los motores a 220 para que no se produzcan daños.

En el siguiente código se puede observar la parte donde actúa el control:

```
rpm_iz = ((float)tick_iz / 384) / 0.00166667; //Cálculo de las rpm de las ruedas, midiendo los ticks incrementales
rpm_de = ((float)tick_de / 384) / 0.00166667; //desde la última medición
tick_iz = 0;
tick_de = 0;

error_rpm_iz = rpm_ref - rpm_iz; //Calculamos el error entre la medicion real de rpm de cada rueda y la referencia
error_rpm_de = rpm_ref - rpm_de; //Este error se lo restamos o sumamos al pwm de modo que es proporcional al error.

if (error_rpm_iz > 0) pwm_iz = pwm_iz + 5; //Vamos sumando o restando un offset de 5 a los pwm a medida que se requiere de aumentar o disminuir las rpm
else pwm_iz = pwm_iz - 5;

if (error_rpm_de > 0) pwm_de = pwm_de + 5;
else pwm_de = pwm_de - 5;

if (abs(pwm_iz) > 220) pwm_iz = 220; //Saturamos los valores pwm del motor
if (abs(pwm_de) > 220) pwm_de = 220;

avanza(abs(pwm_iz), abs(pwm_de)); //Avanza
```

## MODO 6

Este modo es una modificación del anterior ya que se nos pide los escalones de velocidad, pero avanzando en línea recta en una superficie.

En este caso hemos implementado la referencia por Bluetooth y se han ajustado una serie de parámetros para que la ejecución sea mejor y con menos errores. Ahora el recalculado de las variables se hace cada 50 milisegundo, dejando el cálculo de las revoluciones por minuto de la misma forma. También se va sumando o restando un valor de 2.5, en vez de 5, a los PWM de los encoders.

Al hacer pruebas nos dimos cuenta que se nos iba un poco hacia los lados ya que el control de cada rueda es individual y distinto. Para ello se nos ocurrió calcular un error de la velocidad, haciendo la diferencia entre las revoluciones de la rueda izquierda y la derecha. Si este error es mayor que 3, se le suma al PWM de la rueda derecha el valor de la multiplicación del error por la constante de velocidad Kvel=0.2 usada en

este control proporcional. En cambio, si el error es menor que -3, se le suma al PWM de la rueda izquierda el valor absoluto de la multiplicación del error por la constante de velocidad Kvel=0.2 usada en este control proporcional.

El siguiente código muestra la parte en la que actúan los controles:

```
rpm_iz = ((float)tick_iz/384)/0.00166667; //Cálculo de las rpm de las ruedas, mid
iendo los ticks incrementales
rpm_de = ((float)tick_de/384)/0.00166667; //desde la última medición
tick_iz = 0;
tick_de = 0;

error_rpm_iz = rpm_ref - rpm_iz; //Calculamos el error entre la medicion real de
rpmde cada rueda y la referenca
error_rpm_de = rpm_ref - rpm_de; //Este error se lo restamos o sumamos al pwm de
modo que es proporcional al error.
error_vel = rmp_iz - rpm_de;//error de las velocidades para mantener recto

if(error_vel>3) pwm_de = pwm_de + error_vel*Kvel;
else if(error_vel<-3) pwm_iz = pwm_iz + abs(error_vel*Kvel);

if (error_rpm_iz > 0) pwm_iz = pwm_iz + 2.5; //Vamos sumando o restando un offset
 de5 a los pwm a medida que se requiere de aumentar o disminuir las rpm
else pwm_iz = pwm_iz - 2.5;

if (error_rpm_de > 0) pwm_de = pwm_de + 2.5;
else pwm_de = pwm_de - 2.5;

if (abs(pwm_iz) > 220) pwm_iz = 220; //Saturamos los valores pwm del motor
if (abs(pwm_de) > 220) pwm_de = 220;

avanza(abs(pwm_iz), abs(pwm_de)); //Avanza
```

## MODO 7

En este modo se nos pide que estimemos la posición del robot basándonos en la odometría y teniendo en cuenta el modelo cinemático del vehículo. En este caso en concreto vamos a hacer que el robot realice un cuadrado de un metro de lado, acabando en la posición de inicio.

Para la realización de este modo tomaremos como ejemplo el código empleado en el modo 6. En este caso vamos a introducir dos nuevas medidas. La primera de ellas es la distancia recorrida en centímetros de cada rueda, y lo haremos siguiendo la siguiente fórmula:

$$cm = \frac{(2\pi r * pulsos)}{resolucion}$$

Siendo r el radio de las ruedas (3.25 cm) y la resolución 384, que se corresponde con la de los encoders. Vamos a trabajar con la distancia recorrida del centro, por lo que tendremos que hacer la media entre la

rueda derecha y la izquierda.
La segunda medida será el factor de escala medido experimentalmente, el cual sirve para hacer el cálculo correcto de los centímetros recorridos y que sacamos con las siguientes formulas:

$$x = x + dist_{centro} * \cos(\varphi) * 1.3$$

$$y = y + dist_{centro} * \sin(\varphi) * 1.3$$

$$\varphi = \varphi + \frac{dist_{der} - dist_{izq}}{b}$$

Siendo b=10 cm la separación entre las ruedas del robot.

Seguidamente podemos observar el código donde se efectúa el control:

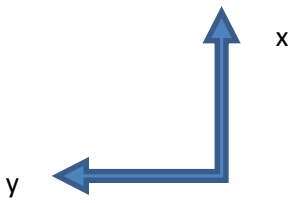
```
if(error_vel>3) pwm_de = pwm_de + error_vel*Kvel;
else if(error_vel<-3) pwm_iz = pwm_iz + abs(error_vel*Kvel);

if (error_rpm_iz > 0) pwm_iz = pwm_iz + 2.5; //Vamos sumando o restando un offset
de5 a los pwm a medida que se requiere de aumentar o disminuir las rpm
else pwm_iz = pwm_iz - 2.5;

if (error_rpm_de > 0) pwm_de = pwm_de + 2.5;
else pwm_de = pwm_de - 2.5;

if (abs(pwm_iz) > 220) pwm_iz = 220; //Saturamos los valores pwm del motor
if (abs(pwm_de) > 220) pwm_de = 220;
```

Otro aspecto a destacar es que hemos creado una sentencia switch case para que cuando el robot llegue a la distancia deseada haga el giro deseado. Esto lo hacemos siguiendo el sistema de coordenadas del robot, que hemos deducido que es:




Por tanto, hacemos que el robot avance un metro y luego que pivote 90 grados en sentido antihorario (excepto en el último caso, que no gira), debido a la disposición de los ejes. Esto lo hacemos teniendo en cuenta en que posición y ángulo está en cada caso. Como podemos observar, el robot siempre trabajará en las pares positiva de los ejes.

Este bucle descrito quedaría de la siguiente forma:

```
switch (caso) {
  case 0:
    if (x < 90) {
```

```
      avanza(abs(pwm_iz), abs(pwm_de)); //Avanza
    }
    else {
      pivote_izq();
      if (phi >= PI / 2) {
        para();
        delay(1000);
        caso = 1;
        y=0;
      }
    }
    break;
case 1:
    if (y < 90) {
     avanza(abs(pwm_iz), abs(pwm_de)); //Avanza
    }
    else {
      pivote_izq();
      if (phi >= PI) {
        para();
        delay(1000);
        caso = 2;
        x=100;
      }
    }
    break;
 case 2:
    if (x > 0) {
      avanza(abs(pwm_iz), abs(pwm_de)); //Avanza
    }
    else {
      pivote_izq();
      if (phi >= 1.5 * PI) {
        para();
        delay(1000);
        caso = 3;
        y=100;
      }
    }
    break;
 case 3:
    if (y > 0) {
      avanza(abs(pwm_iz), abs(pwm_de)); //Avanza
    }
    else {
      para();
      delay(1000);
      caso = 4;
```

```
    }
    break;
  case 4:
    para();
    break;
}
```

MODO 8

En este modo libre hemos decidido hacer un control introduciendo dos fotoresistores. Nuestro objetivo, el cual se ha conseguido, era instalar estas dos nuevas piezas junto con la configuración del modo 7 para que cuando estos sensores reciban la luz de la linterna el coche gire. Hemos instalado dos piezas, una en cada lateral frontal del robot. Si se recibe la luz por el sensor derecho, el robot girará hacia la derecha y, si se recibe la luz por el sensor izquierdo, el robot gira hacia este sentido. Para el correcto funcionamiento de este modo libre se deberán de apagar las luces de la sala donde se encuentre el robot, ya se ha programado con unos límites específicos para esto.

Para poder establecer la comunicación con los fotoresistores los hemos conectado a dos entradas analógicas de la placa Arduino. La entrada A0 corresponde con el pin de la izquierda y la entrada A1 con el derecho. Además, hemos inicializado el valor de las resistencias eléctricas de ambos sensores a cero.

```
int ldrPinizq = A0;              // LDR pin izq
int ldrizq = 0;                 // Valor LDR izq
int ldrPinder = A1;              // LDR pin der
int ldrder = 0;                 // Valor LDR der
```

Una vez declaradas las variables que vamos a usar, configuramos los pines como salida y arrancamos los puertos series en el 'setup'. Posteriormente, en el 'loop' empezamos a escribir el programa. Lo primero que hacemos es obtener los valores analógicos de los fotoresistores pasándolos por los filtros Kalman previamente comentados. Una vez obtenida esta información, calculamos el error con la diferencia de los valores derecho menos izquierdo.

```
ldrizq = kalmanizq(analogRead(ldrPinizq)+50); // Read the analog value of the LDR
ldrder = kalmander(analogRead(ldrPinder));    // Read the analog value of the LDR
error = ldrder-ldrizq;
```

Posteriormente saturamos los valores de PWM de cada rueda en el avance (220) y en el retroceso (-80) para que no se produzcan daños en el robot.

El código de control y ejecución principal lo hacemos con un 'switch', ya que nos permite distinguir entre los diferentes casos que se pueden dar.
En el primer caso, el 0, el robot se encuentra parado esperando que el error supere los siguientes límites. Si el error es mayor que 25, se está recibiendo prácticamente la luz de igual formas por ambos sensores y el robot pasa al caso 1 donde avanza con un PWM=100 para cada rueda. Si el error es mayor que 150 significa que la luz se recibe por el sensor derecho y nos vamos al caso 2, donde el robot pivota hacia la derecha con un PWM=80 de la rueda izquierda. Si el error es menor que -80 significa que se recibe la luz por el sensor izquierdo y se pasa al caso 3, donde se pivota hacia la izquierda con un PWM=80 para la rueda derecha.
En el segundo caso, el 1, el coche está avanzando y si el error es menor que 30 es que no está llegando luz

a los fotoresistores y volveremos al caso 0 para que el robot se pare. Si el error es mayor que 250, se pasara al caso 2 para que pivote a la derecha. Si el error es menor que -80, se pasa al caso 3 para que el robot pivote a la izquierda.
En el tercer caso, el 2, se produce el pivotamiento a la derecha. Si el error es menor que 250, se pasa al caso 1 de nuevo y el robot avanzaría recto.
Por último, en el caso 3, el robot estaría pivotando a la izquierda. Si el error se encuentra entre los valores de 0 y 250, se pasa al caso 1, y si el error supera los 250, se pasa al caso 2.
Estos valores límites para ir pasando entre los diferentes casos del 'switch' se han obtenido experimentalmente.

La parte del código donde se aplica este control de ejecución se puede observar:

```
switch(caso){
case 0:
  para();
  if(error>25){
    caso=1;
    break;
  }
  if(error>150){
    caso=2;
    break;
  }
  if(error<-80){
    caso=3;
    break;
  }
  else{
    caso=0;
  }
  break;
case 1:
  avanza(100, 100); //Avanza
  if(error<30){
    caso=0;
    break;
  }
  if(error>250){
    caso=2;
    break;
  }
  if(error<-80){
    caso=3;
    break;
  }
  break;
case 2:
  pivote_der(); //Pivota a la derecha
  if(error<250){
```

```
    caso=1;
    break;
  }
  break;
case 3:
  pivote_izq(); //Pivota a la derecha
  if(error>0 && error<250){
    caso=1;
    break;
  }
  if(error>250){
    caso=2;
    break;
  }
  break;}
```

# IV. TELEMETRÍA Y VÍDEOS

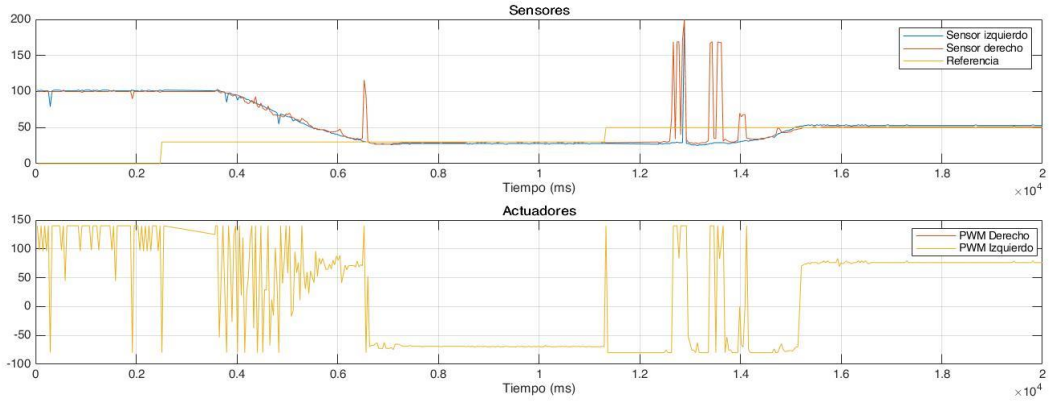
Para ejecutar la telemetría hemos tenido que exportar los datos sacados de la ejecución de cada modo a partir de Google Drive y guardarlo con extensión txt. Posteriormente, desde Matlab, ejecutamos 'telemetria(nombre.txt)' y obtenemos las gráficas presentadas en este apartado.

## MODO 1

Para el modo 1 hemos sacado las variables del tiempo transcurrido, las distancias leídas por los sensores, la referencia y los PWM de ambas ruedas como:

*"Serial2.println(String(elapsedTime) + " " + String(distanciaizq) + " " + String(distanciader) + " " + String(ref) + " " + "1" + " " + String(pwmder) + " " + String(pwmder));"*

De esta forma, hemos obtenido la siguiente gráfica:




Podemos comprobar picos en las medidas de los sensores, que se deben al ruido que hay en ellos. Quitando esta observación, se puede comprobar como el robot sigue a la referencia correctamente. Al hacer un PD, el efecto derivativo provoca una lentitud en el seguimiento de la referencia, pero nos asegura que el PWM aumente de forma paulatina y que se elimine en cierta medida la inercia del movimiento.

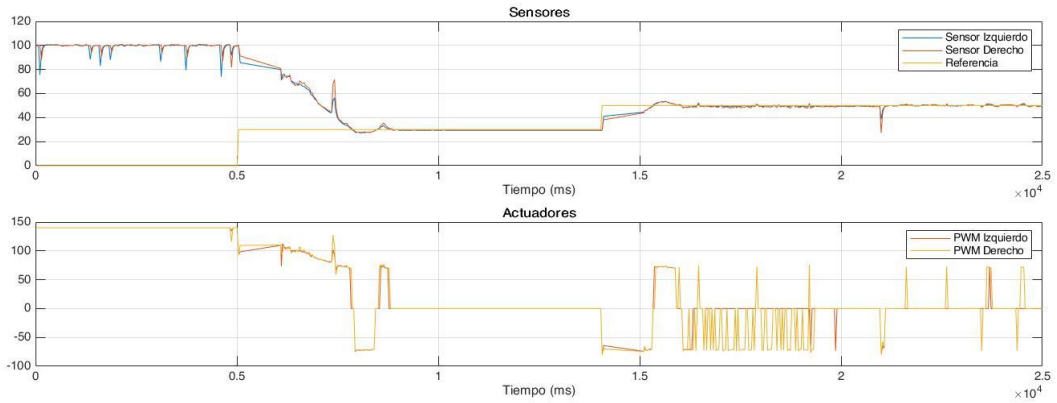
El vídeo correspondiente a este experimento se encuentra en la carpeta “P4G09” y se llama “P4G09MODO1.mp4”.

## MODO 2

Para el modo 2 hemos sacado las variables del tiempo transcurrido, las distancias leídas por los sensores, la referencia y los PWM de ambas ruedas como:

*“Serial2.println(String(elapsedTime) + " " + String(distanciaizq) + " " + String(distanciader) + " " + String(ref) + " " + "2" + " " + String(pwmizq) + " " + String(pwmder));”*

De esta forma, hemos obtenido la siguiente gráfica:




En este modo podemos ver medidas suavizadas gracias al efecto del filtro de Kalman. Tal y como se nos pide, ponemos el robot a un metro de la pared, mandamos la referencia por Bluetooth de 30 cm y vemos como progresivamente el robot se va acercando a la referencia y se corrige en los últimos instantes. En la gráfica vemos perfectamente como el robot se queda parado cuando llega a la referencia. Posteriormente le mandamos la orden de irse a los 50 cm de referencia y vemos como el robot retrocede hasta llegar a la misma.

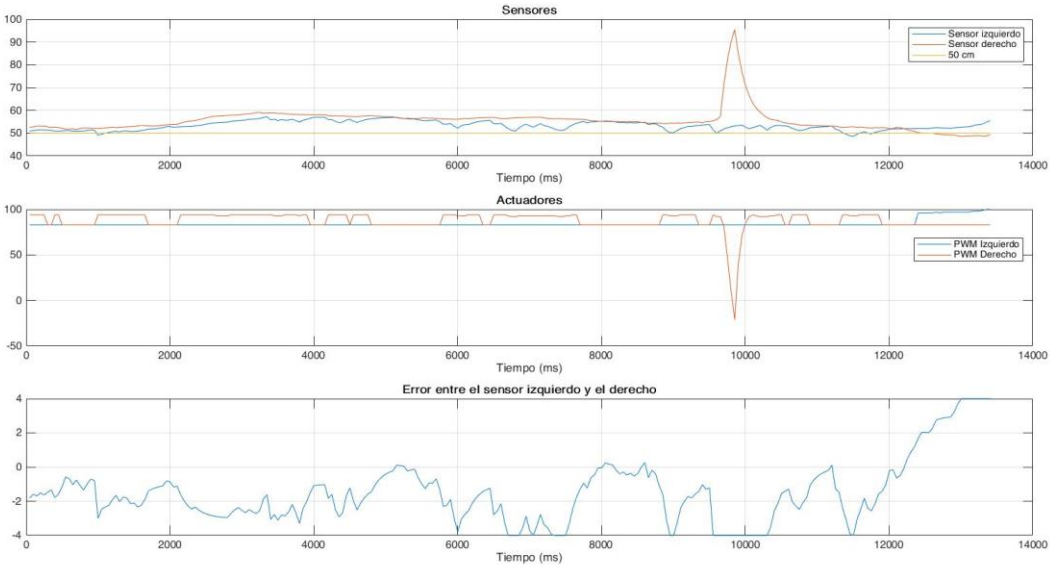
El vídeo correspondiente a este experimento se encuentra en la carpeta “P4G09” y se llama “P4G09MODO2.mp4”.

## MODO 3

Para el modo 3 hemos sacado las variables del tiempo transcurrido, las distancias leídas por los sensores, la referencia, los PWM de ambas ruedas y el error como:

*“Serial2.println(String(elapsedTime) + " " + String(distanciaizq) + " " + String(distanciader) + " " + "50" + " " + "3" + " " + pwmizq + " " + pwmder + " " + String(err));”*

De esta forma, hemos obtenido la siguiente gráfica:




Tras esta telemetría, observamos cómo se mantiene la referencia de 50 cm tal y como nos piden en el enunciado. El error se corrige muy lentamente debido al controlador que hemos diseñado, que ha sido el que nos ha parecido más oportuno, ya que otros controles probados llevaban hacia una dirección diagonal a la pared. Podemos observar como a los 10 segundos se pierde el control y posteriormente se corrige a la perfección, aunque tenemos que parar la prueba por falta de espacio.

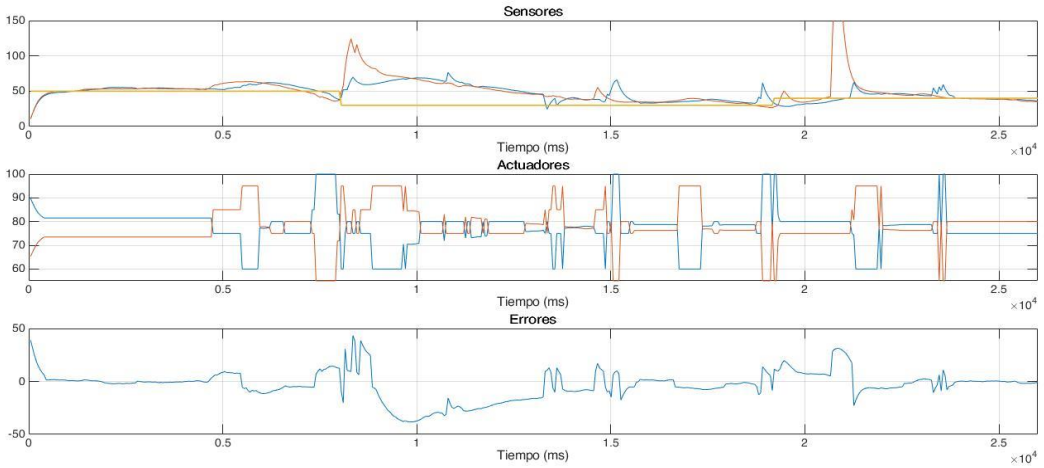
El vídeo correspondiente a este experimento se encuentra en la carpeta “P4G09” y se llama “P4G09MODO3.mp4”.

## MODO 4

Para el modo 4 hemos sacado las variables del tiempo transcurrido, las distancias leídas por los sensores, la referencia, los PWM de ambas ruedas y los errores como:

*“Serial2.println(String(elapsedTime) + " " + String(distanciaizq) + " " + String(distanciader) + " " + String(ref) + " " + "4" + " " + String(pwm_izq) + " " + String(pwm_der) + " " + String(err_iz) + " " + String(err_de));”*

De esta forma, hemos obtenido la siguiente gráfica:



Podemos comprobar como los sensores de ultrasonidos intentan llegar a la referencia, consiguiéndolo en todo momento, aunque se desestabilicen un poco. En la segunda parte de la gráfica se puede observar el PWM que se aplica en cada instante, llegando a tenerse que saturar en los cambios de referencia, sobre todo. Por último, se observa el error entre la distancia media de los sensores y la referencia. Podemos comprobar como el error aumenta con los cambios de referencia y posteriormente tiende a ser cero.

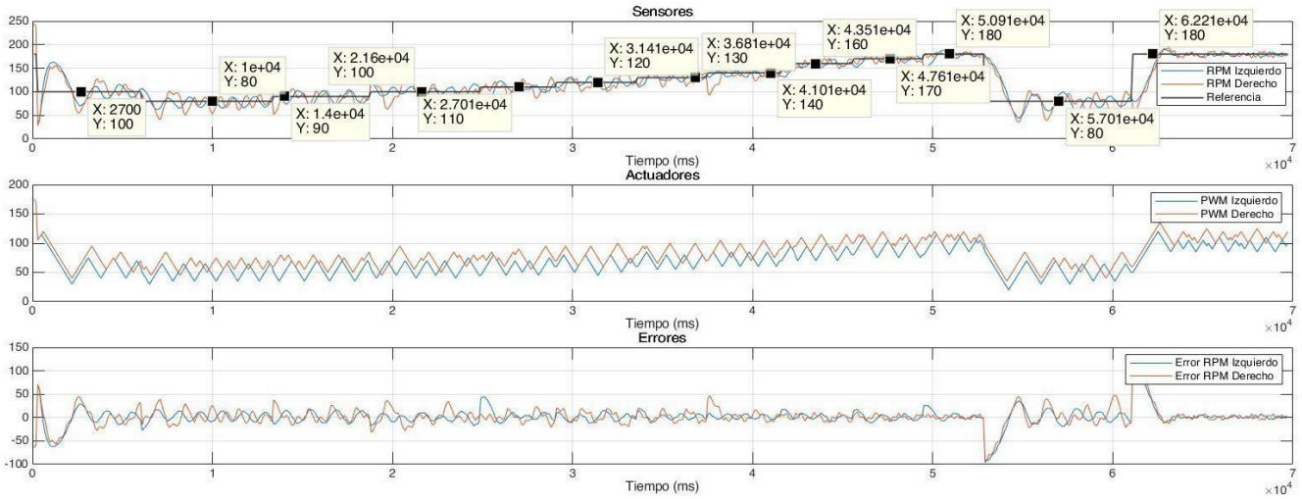
El vídeo correspondiente a este experimento se encuentra en la carpeta “P4G09” y se llama “P4G09MODO4.mp4”.

## MODO 5

Para el modo 5 hemos sacado el tiempo de recalculado, las revoluciones por minuto de las ruedas y de la referencia, los PWM de ambas ruedas y los errores de las revoluciones por minuto como:

*“Serial.println(String(muestreo - muestreo_ant) + " " + String(rpm_iz) + " " + String(rpm_de) + " " + String(rpm_ref) + " " + "5" + " " + String(abs(pwm_iz)) + " " + String(abs(pwm_de)) + " " + String(error_rpm_iz) + " " + String(error_rpm_de));”*

De esta forma, hemos obtenido la siguiente gráfica:




Aquí podemos observar el correcto funcionamiento de los encoders cuando aumentamos o disminuimos la referencia. Se puede observar que cuando el cambio de referencia es mayor, mayor será el error y mayor será el tiempo que tarda el robot en llegar a la misma sobreoscilando y oscilando de forma tenue.

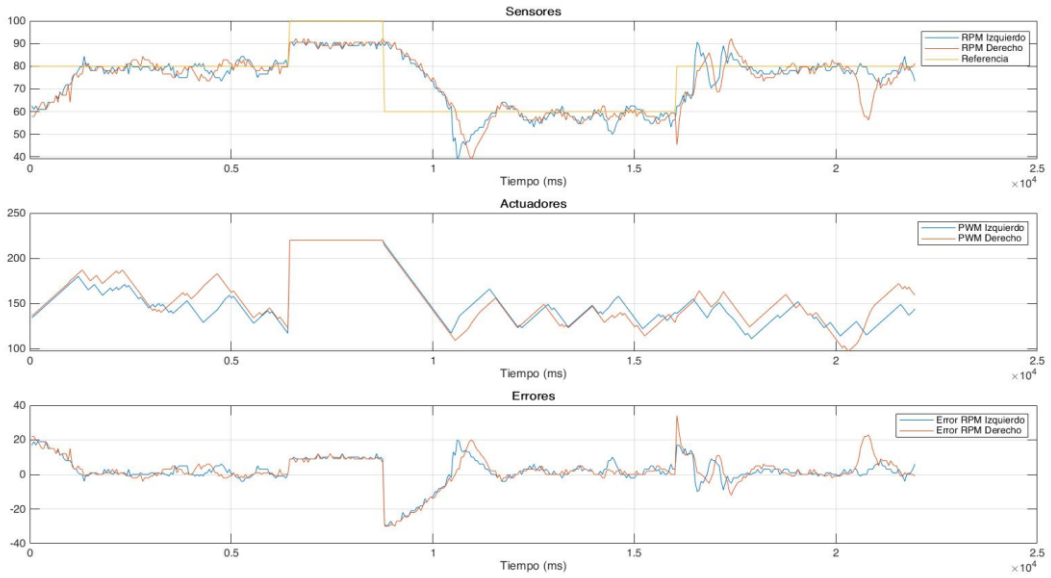
El vídeo correspondiente a este experimento se encuentra en la carpeta “P4G09” y se llama “P4G09MODO5.mp4”.

## MODO 6

Para el modo 6 hemos sacado el tiempo de recalculado, las revoluciones por minuto de las ruedas y de la referencia, los PWM de ambas ruedas y los errores de las revoluciones por minuto como:

*“Serial2.println(String(muestreo - muestreo_ant) + " " + String(rpm_iz) + " " + String(rpm_de) + " " + String(rpm_ref) + " " + "6" + " " + String(abs(pwm_iz)) + " " + String(abs(pwm_de)) + " " + String(error_rpm_iz) + " " + String(error_rpm_de));”*

De esta forma, hemos obtenido la siguiente gráfica:




En esta prueba hemos dado las siguientes referencias: 80-100-60-80 rpm. Se puede observar como el robot sigue a la referencia en cada cambio de la misma excepto cuando es de 100 rpm. Esto ocurre porque tenemos saturado con un PWM=220, que se da cuándo el robot tiene una velocidad de 90 rpm.

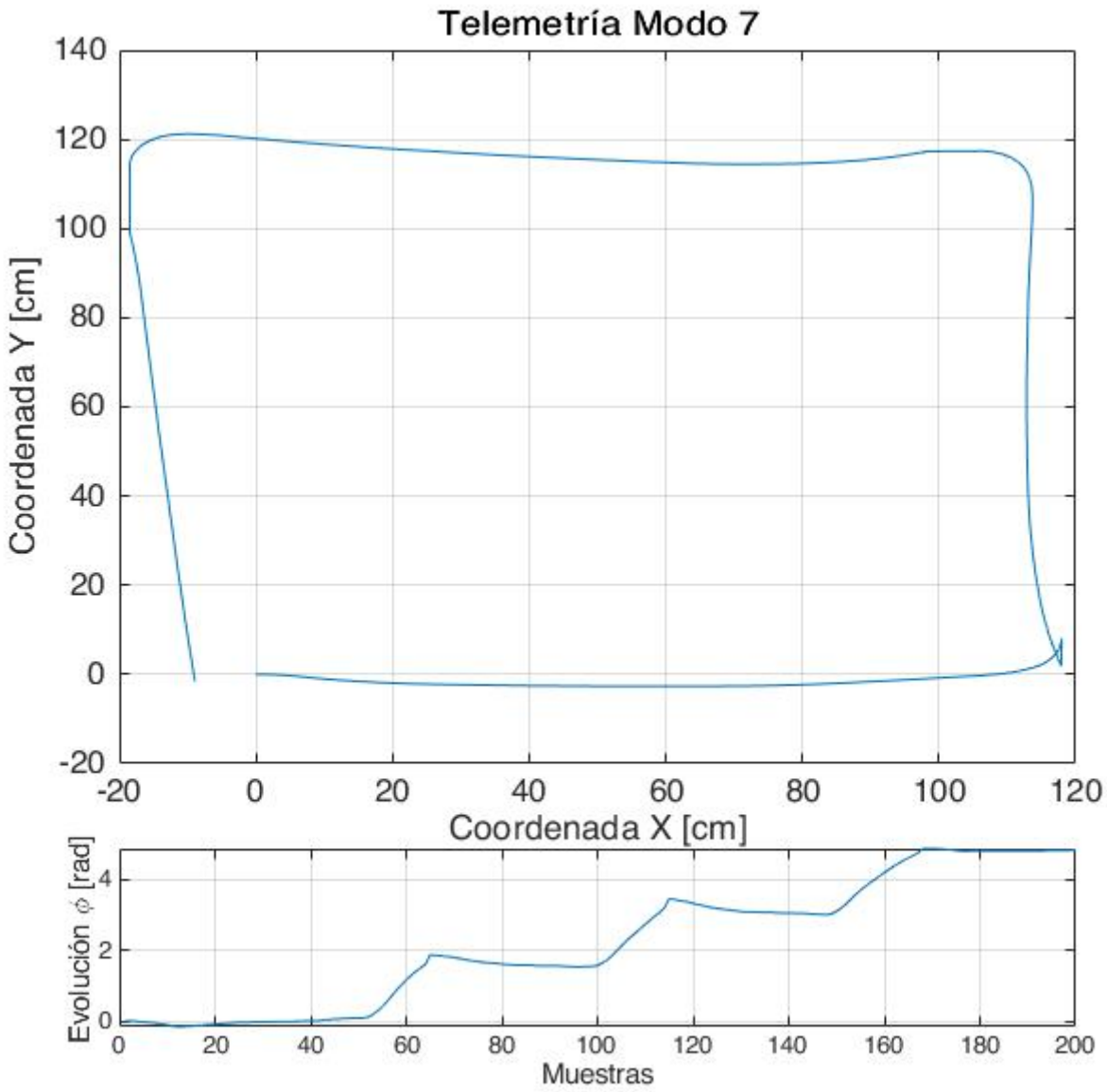
El vídeo correspondiente a este experimento se encuentra en la carpeta "P4G09" y se llama "P4G09MODO6.mp4".

## MODO 7

Para el modo 7 hemos sacado el tiempo de recalculado, las posiciones 'x' e 'y', y el angulo 'phi' como:

*"Serial2.println(String(muestreo - muestreo_ant) + " " + String(x) + " " + String(y) + " " + String(phi));"*

De esta forma, hemos obtenido la siguiente gráfica:




Podemos comprobar con esta telemetría que el robot realiza bien los giros, pero se pasa un poco del metro debido a la inercia.

El vídeo correspondiente a este experimento se encuentra en la carpeta "P4G09" y se llama "P4G09MODO7.mp4".

## V. DIFICULTADES

Para el modo 1 nos hemos encontrado la dificultad de no saber qué control utilizar, ya que era la primera vez que nos enfrentábamos a este ejercicio con un robot de este tipo. Tras prueba y error decidimos usar un PD que, aunque lento, ha sido efectivo.

En el modo 2 nos encontramos con que los sensores de ultrasonidos nos daban muchas medidas erróneas, así que decidimos introducir un filtro de Kalman. Además, en un primer momento, estábamos intentando

hacer el modo con un único control, hasta que vimos que con uno por rueda si funcionaba.

En el modo 3 nos no tuvimos problemas para programar el funcionamiento, aunque sí para la elección del controlador, debido a que, tras muchas pruebas y errores, no encontrábamos las constantes necesarias.

En el modo 4 ha sido dónde más dificultades nos hemos encontrado. No sabíamos de qué forma hacer el control para que no se nos pusiera el robot en diagonal a la pared y empezara a dar medidas muy grandes, lo que conllevaba a un mal control del modo. Tras muchas vueltas conseguimos que no se desestabilizara cuando se acercaba a hacia la pared, aunque cuando se alejaba daba picos de PWM que eran incontrolables. Finalmente, con un control en cascada casi que lo conseguimos. No hemos podido lograr el objetivo completamente, aunque sí se acerca mucho a lo pedido.

En el modo 5 no sabíamos cómo pasar de los pulsos de los encoders a las revoluciones por minuto. Una vez que lo entendimos, no tenía mayor dificultad este modo.

En el modo 6 no hemos encontrado problemas destacables.

En el modo 7 nos encontramos con que hemos tenido que jugar mucho con los parámetros del factor de escala y la cantidad de desplazamiento del robot. Tras haber intentado arreglarlo de todas las formas posibles no hemos conseguido que el robot recorra el metro pedido, sino 1,20m.

En el modo 8 hemos tenido que buscar la declaración y como usar los fotoresistores, pero más allá de esto no hemos encontrado dificultades destacables.