MEMORIA ROBOT MÓVIL:

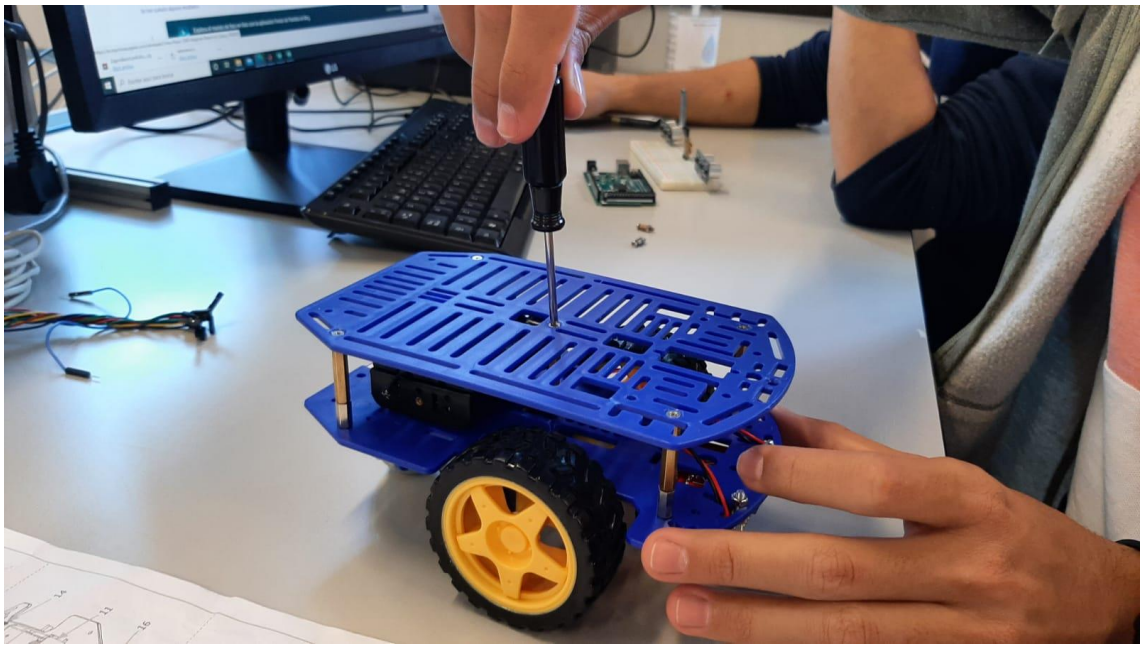
JUAN DE DIOS HERRERA HURTADO

La primera tarea que se tuvo que realizar en la parte de robot móvil fue el montaje.

Montaje

Comenzamos colocando el dispositivo que contiene las 6 pilas AA, este alimenta a la placa Arduino, que a su vez alimenta a la parte de electrónica de potencia, la cual alimenta a los motores de las ruedas.

Posteriormente se pasó a montar la segunda superficie donde irán la placa Arduino junto con la protoboard con sus sensores.

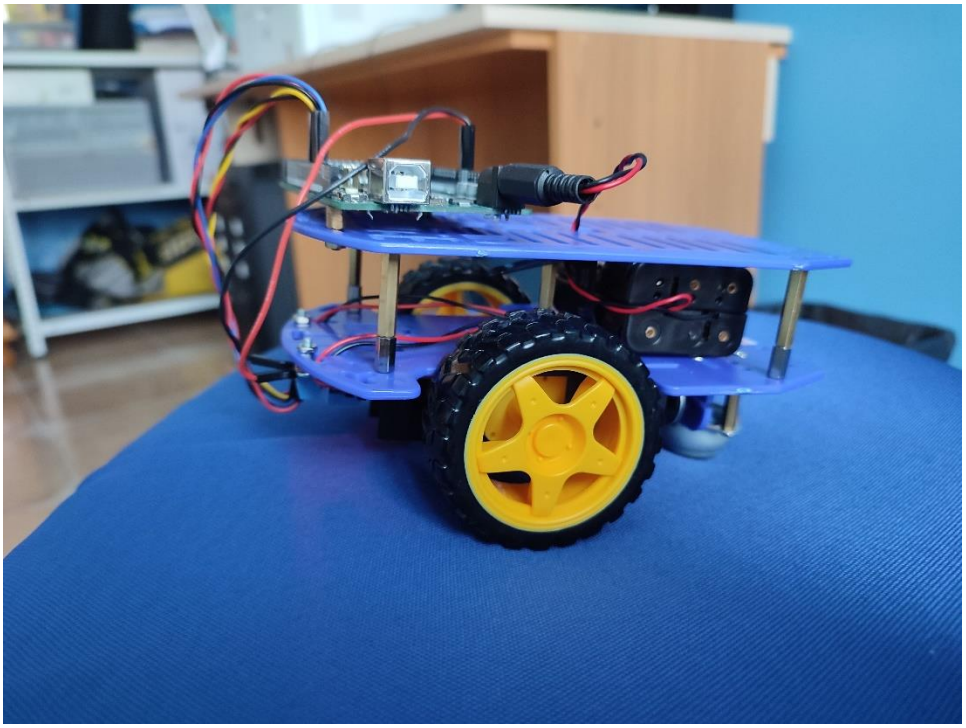


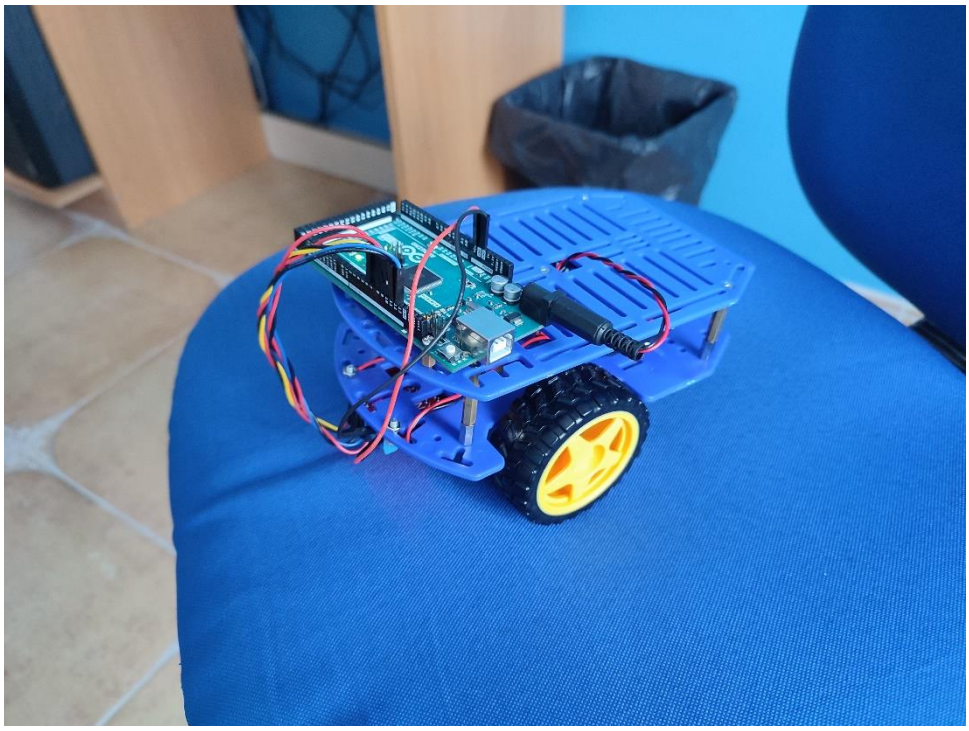
Una vez conseguido, se colocó la placa Arduino. El profesor nos dio varios consejos sobre el montaje de manera que fuese más segura para evitar problemas futuros. Entre estos consejos se puede destacar:

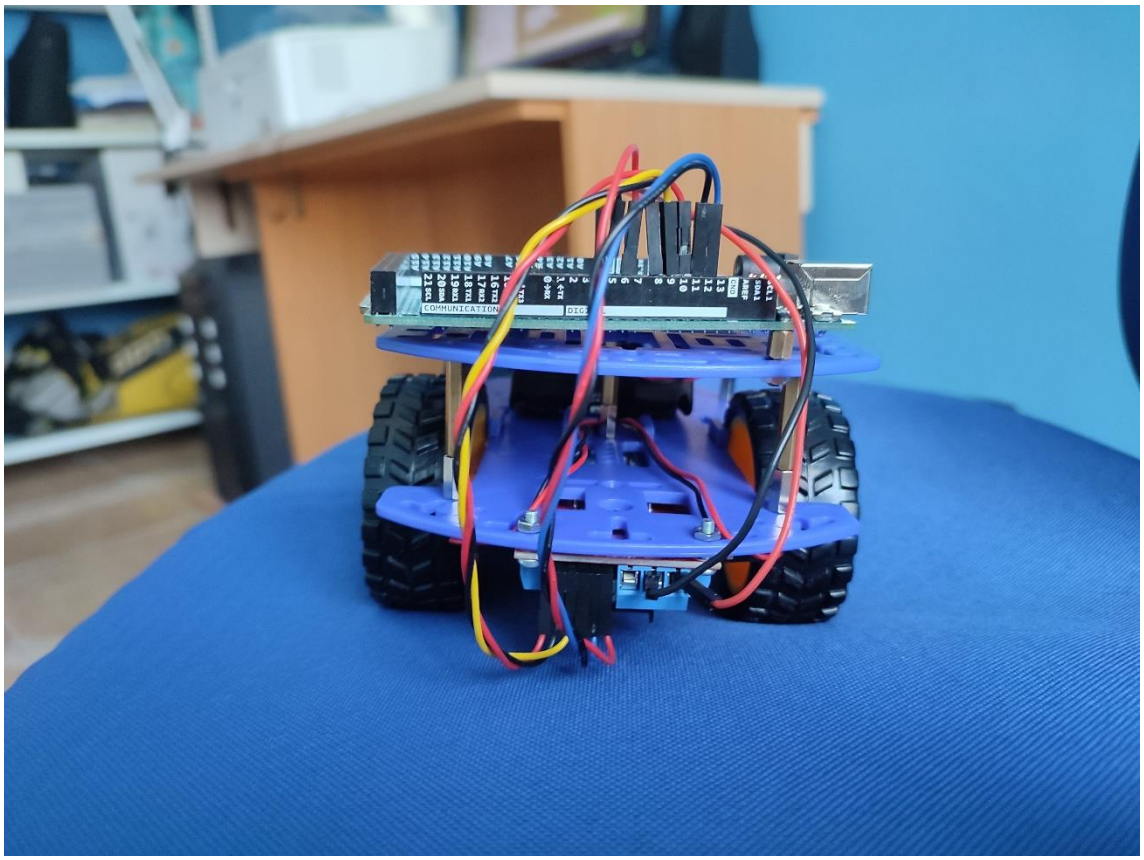
- Evitar empalmar dos cables (ya tuvimos varios problemas debido a que se nos soltaban las uniones).
- Trenzar varios cables para que no estén tan sueltos y para que sean más visibles las cosas a la hora de buscar algo sin tanto cable de por medio.
- La placa nos bailaba un poco, es decir, no estaba sujeta del todo, debíamos solucionarlo ya que podría provocar un cortocircuito.

- Ayudarnos de los colores de los cables para separar cosas, por ejemplo, hemos puesto un cable rojo para cada una de las señales “ENA” y “ENB”, las que regulan la velocidad de los motores.

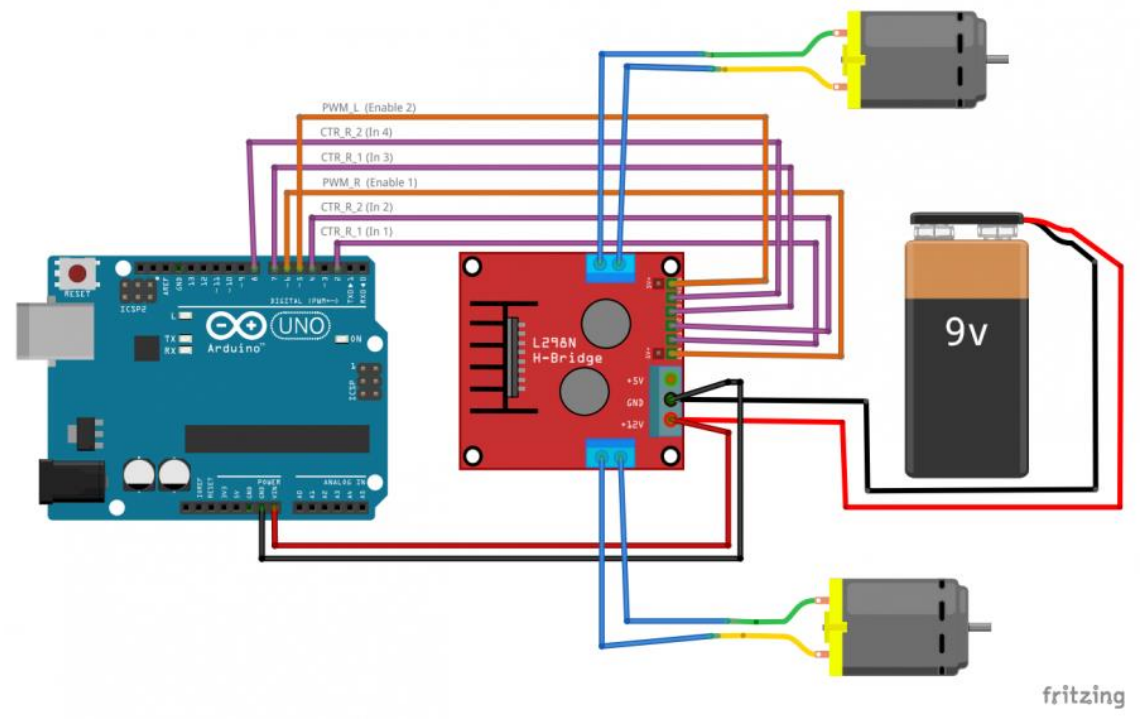
El resultado final se muestra en las siguientes imágenes:







Para el conexionado de los cables nos apoyamos en la siguiente imagen:



Fuente: https://bricolabs.cc/wiki/guias/control_de_motores

Modo 0

Comenzaré explicando un “Modo 0” en el que actuamos directamente sobre el sentido de giro de las ruedas y la velocidad (PWM) de ambas ruedas. Para esta programación, se han recurrido a varias funciones con un nombre indicativo de su función:

- void Adelante ()
- void AdelanteDer ()
- void AdelanteIzq ()
- void Atras ()
- void AtrasDer ()
- void AtrasIzq ()
- void Rotar_Derecha ()
- void Rotar_Izquierda ()
- void Parar ()
- void PararIzq ()
- void PararDer ()

La programación del LN298 y las funciones anteriormente usadas se han obtenido del siguiente enlace: https://www.prometec.net/l298n/.

El fundamento para la rueda derecha es sencillo:

IN1	IN2	Resultado
LOW	LOW	No se mueve
LOW	HIGH	Giro hacia atrás
HIGH	LOW	Giro hacia delante
HIGH	HIGH	No se mueve

Esto es análogo con la rueda izquierda pero con “IN3” e “IN4”, es decir:

$$\mathrm{IN1} = \mathrm{IN3}$$

$$\mathrm{IN2} = \mathrm{IN4}$$

Con “ENA” establecemos la velocidad, este valor vendrá de un pin PWM del Arduino en un rango [-255 , 255]. Lo mismo con “ENB”.

De este modo no hay un código como tal, simplemente se incluye en el resto de modos todas las funciones anteriormente mencionadas.

Ahora explicaré cómo se usan los ultrasonidos, ya que será de gran importancia para el desarrollo de los modos 1, 2, 3 y 4.

El código necesario para realizar una medida es el siguiente:

```
//Ultrasonidos 1:
digitalWrite(TRIG1, HIGH);
delayMicroseconds(10);
digitalWrite(TRIG1, LOW);
DURACION1 = pulseIn(ECO1, HIGH);
DISTANCIA1 = DURACION1 / 58.2;
```

Las señales “TRIG1” y “ECO1” son pines PWM de la placa Arduino, lo mismo para el otro ultrasonido con las señales “TRIG2” y “ECO2”.

Los pasos son:

- Activamos la señal “TRIG1” para que el ultrasonido comience a emitir ondas.
- La desactivamos para dejar de emitir.
- Finalmente usamos la función “pulseIn” junto con la señal “ECO1”. Esta función en nuestro caso, espera a que “ECO1” tome el valor ‘HIGH’, una vez esto, comienza a contar hasta que la señal cambie a valor ‘LOW’.

El tiempo que devuelve son microsegundos, para pasarlo a centímetros debemos dividir entre 58.2. Sabiendo que la velocidad del sonido son 343,2 m/s, que la distancia recorrida por la onda ha sido dos veces la distancia del ultrasonido a la pared (ida y vuelta), y que la velocidad es igual a distancia entre tiempo, el razonamiento es el siguiente:

$$343{,}2\,\frac{\cancel{\text{m}}}{\cancel{\text{s}}} * \frac{1\,\cancel{\text{s}}}{10^6\,\mu\text{s}} * \frac{100\ \text{cm}}{1\,\cancel{\text{m}}} = \frac{2 * \text{Distancia (cm)}}{\text{Tiempo } (\mu s)}$$

$$Distancia(cm) \approx 0.034\,\frac{cm}{\cancel{\mu s}} * \frac{Tiempo\ (\cancel{\mu s})}{2} \approx \frac{Tiempo\ (\mu s)}{58.2}$$

Adicionalmente, explicaré cómo se ha procedido para llevar a cabo la recepción de las referencias introducidas por el usuario. Esta tarea se ha llevado a cabo a partir de un buffer. El código tiene la siguiente estructura:

```
if(Serial.available() > 0) //Para procesar los diferentes caracteres que hay en el buffer del serial, debemos quedarnos aquí hasta que llegue un \n
  {
    //Array/buffer para almacenar el mensaje entrante
    static char Buffer[LongitudBuffer];
    static unsigned int IndiceBuffer = 0;

    //Leer el siguiente byte disponible en el buffer del Serial
    char byte_temporal = Serial.read();

    //Comprobar si el siguiente mensaje es caracter terminador
    if(byte_temporal != '\r')
    {
      //Añadir el byte entrante al array:
      Buffer[IndiceBuffer] = byte_temporal;
      IndiceBuffer++;
    }

    //Mensaje completo recibido:
    else
    {
      //Añadir caracter nulo al array:
      Buffer[IndiceBuffer] = '\0';

      //Convertir el mensaje a entero:
      sscanf(Buffer, "%lu", &velocidad_buffer); //Se formatea como unsigned long para que funcione la separación de los números

      //Resetear índice del buffer:
      IndiceBuffer = 0;
    }
  }
```

La función *“Serial.available()”*, nos devuelve el número de caracteres disponibles para leer desde el puerto serie, es decir, son datos que ya han llegado y que se almacenan en un buffer del Arduino que tiene una capacidad de 64 caracteres.

Luego creamos un vector tipo char (nuestro buffer) con tamaño inicialmente de 50 caracteres.

Mediante *“Serial.read()”* leemos un byte (carácter) y comprobamos si es distinto a “\r”, lo que significaría que todavía tenemos datos por leer. En ese caso guardamos el byte en nuestro buffer. Si el carácter era “\r”, significa que hemos leído todo el contenido del buffer, ante este caso se le añade el byte de terminador “\0” a nuestro buffer y se resetea el índice del mismo.

Finalmente se convierte a entero para poder manipular la cadena luego.

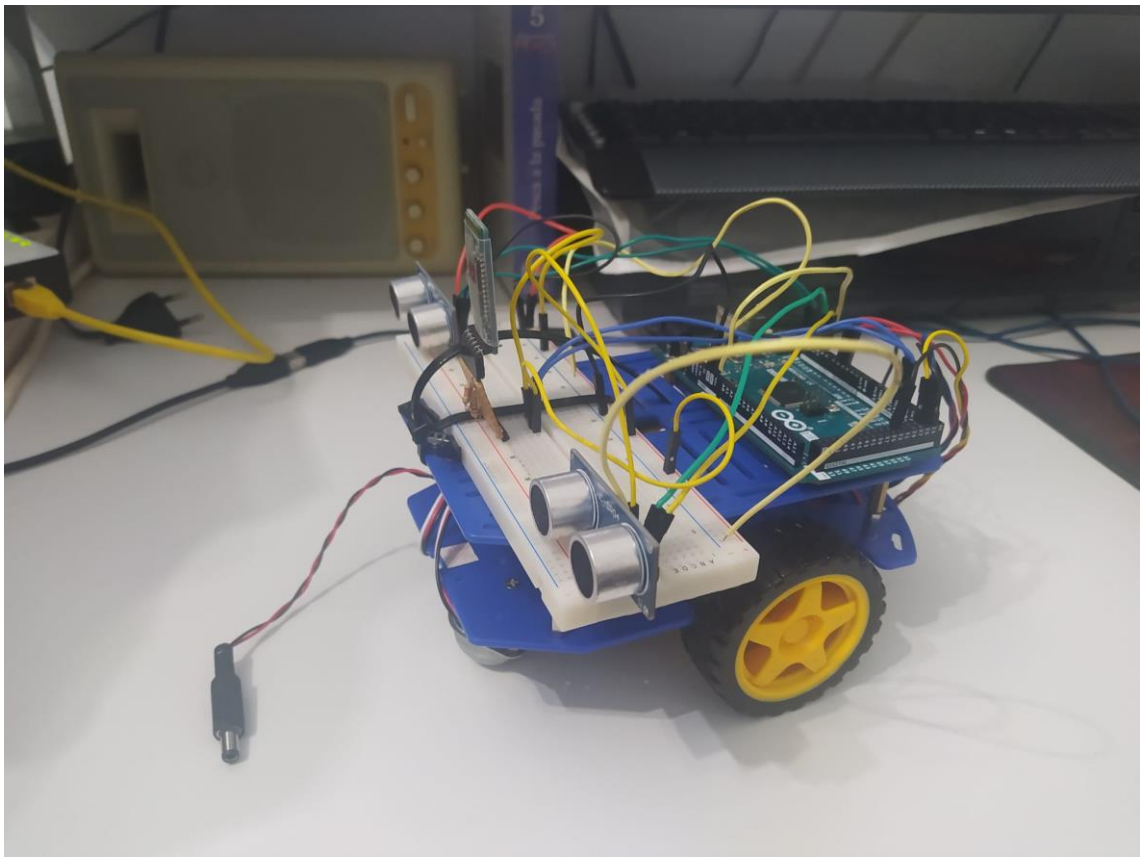
Trabajando de esta manera, tan solo se procesa un carácter por cada ciclo de loop, lo que nos garantiza que el tiempo de ejecución de esta parte del código será similar en cada ciclo.

Tal y como establa planteado al principio, se procesaban todos los caracteres en un solo ciclo de loop, lo que provocaría tiempos de ejecución muy dispares de esta parte del código para distintos ciclos de loop. El problema de este primer planteamiento es que en el loop que se procesaban los datos, tardaría mucho más tiempo que los otros ciclos, un tiempo que estaría nuestro robot moviéndose a ciegas sin ser controlado.

Luego, asegurándonos de tener tiempos más pequeños y parecidos en este código, nos aseguramos a su vez controlar el robot de manera más robusta.

Modo 1 – 2

Para estos modos, se han dispuesto los dos sensores ultrasonido en la parte frontal de nuestro robot tal y como se muestra en la imagen:



Es lógico pensar que, si queremos situar nuestro robot a una cierta distancia frontal con respecto a la pared y de manera lo más perpendicular posible, la mejor manera de hacerlo es con los ultrasonidos en la posición en la que se han dispuesto.

Así, podremos controlar la distancia hacia la pared y, si por ejemplo el robot está girado un poco, un ultrasonido detectará que está en la posición correcta y su rueda correspondiente no se moverá, pero el otro ultrasonido detectará que se encuentra a una distancia ligeramente inferior/superior a la referencia y, se actuará sobre esa rueda levemente hasta conseguir alcanzar la referencia deseada en ese ultrasonido también.

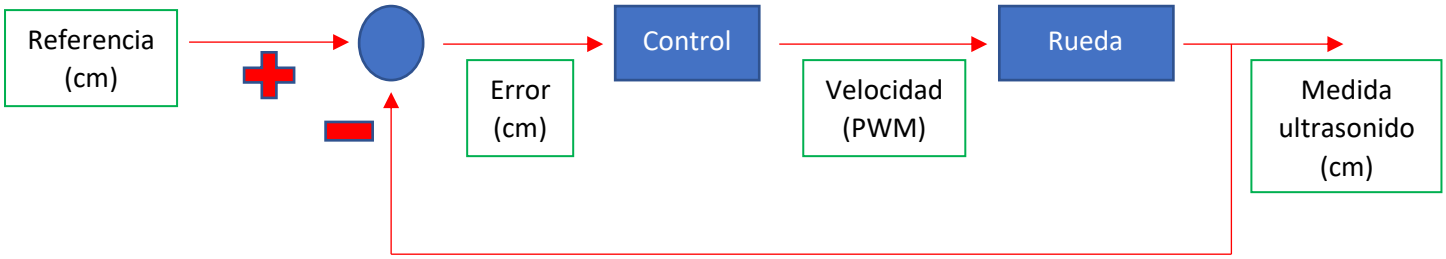
Comentado el montaje pasamos a explicar a rasgos generales, el código:

- Se recurrió a un control PI (proporcional + integral). La parte integral será la que se encargue de terminar de ajustar la medida del sensor a la referencia cuando estemos cerca de esta referencia.
- Como ya se comentó, tenemos un ultrasonido asociado a cada rueda luego, se realizarán dos controles PI, una para cada rueda.
- A cada control le llega como referencia la distancia solicitada por el usuario, que se recibirá a través de “Putty” o “Tera Term”.

- La salida será la medida del ultrasonido, luego el error será:

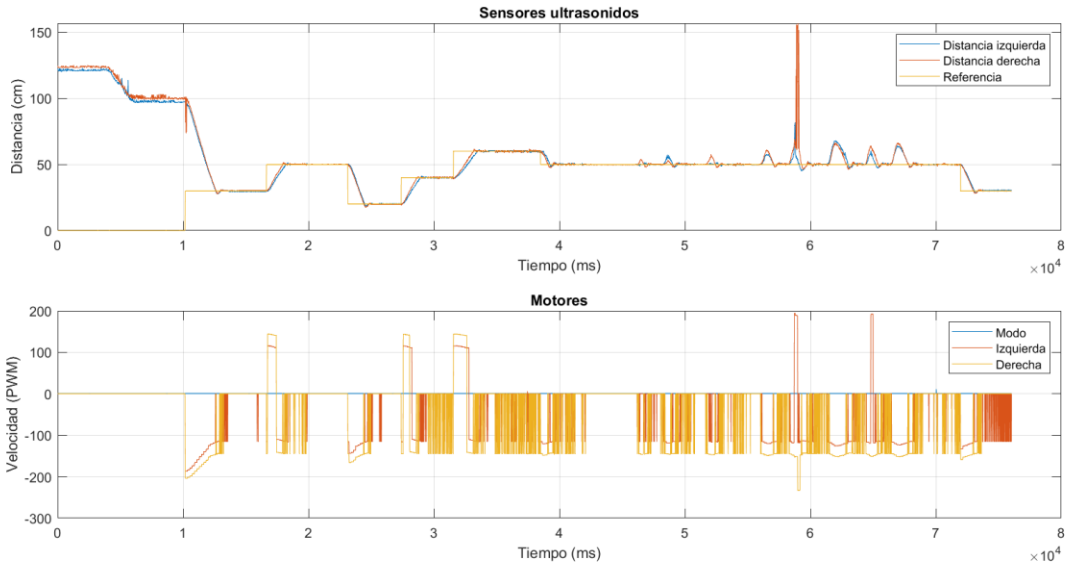
$$\text{Error} = \text{Referencia} - \text{Medida}_{\text{ultrasonido}}$$

- Este error entrará al control y la salida se aplicará sobre las ruedas de la siguiente manera:



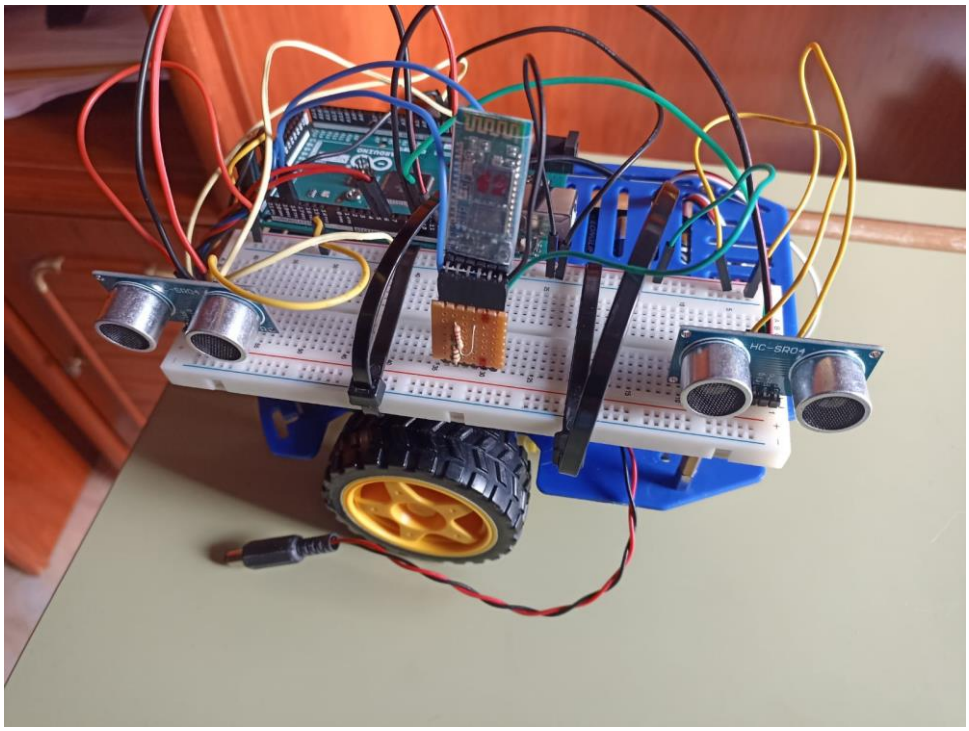
- La salida del PWM se mueve en el rango [-255 , 255], por ello, se han debido de hacer dos cosas:
  - Saturar la señal al rango [-255 , 0] o al rango [0 , 255] según si el valor obtenido por el control es positivo o negativo.
  - Mapear esos rangos para transformarlos a los rangos [-255 , -80] o al rango [80 , 255], debido a que un valor en el rango [-80 , 80] no es un valor suficiente de PWM para provocar un movimiento en las ruedas.
- Adicionalmente se ha saturado el aporte de la parte integral del control con el fin de evitar “el efecto windup”.
- Por último, comentar que la señal de control (PWM de las ruedas), se anula cuando el error entre referencia y medida de ultrasonido es menor a 0.5 cm, ya que es muy improbable obtener “Error = 0.0 cm”, debido a que el sensor introduce ruidos en la medida entre otros motivos.

La gráfica obtenida al realizar el experimento es la siguiente:



Modo 3

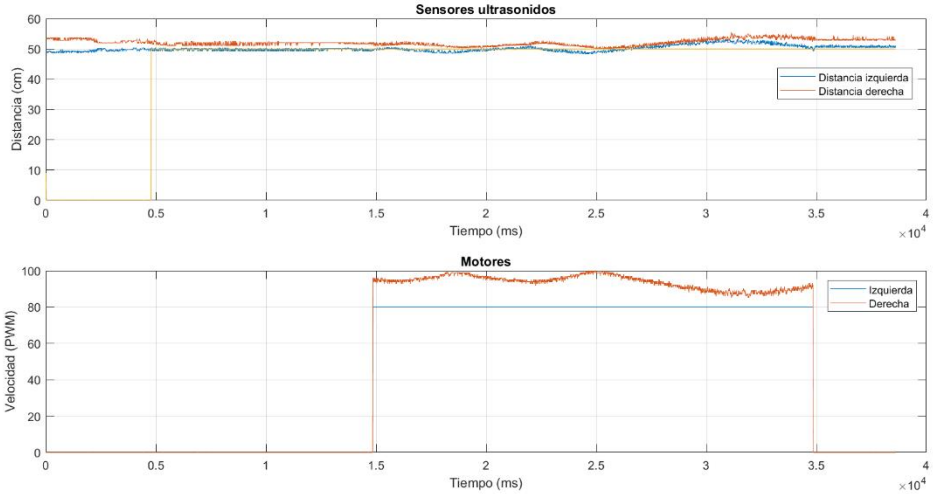
Para este modo y para el modo 4, los sensores se han dispuesto en el lateral derecho tal y como se muestra en la imagen:



El código consiste en lo siguiente:

- Básicamente es lo mismo que el control para los modos 1 y 2, con la diferencia que esta vez lo que queremos es mantenernos paralelos a la pared, a 50 cm concretamente.
- Para ello, mediante un control PI (proporcional + integral), generamos una pequeña diferencia de velocidades entre las ruedas con el fin de que se acerque o aleje nuestro robot lateralmente con respecto a la pared.
- Es importante mencionar, que se ha optado por dejar la velocidad de la rueda izquierda constante con valor 80 PWM y controlar solo la rueda derecha.
- La velocidad de la rueda derecha tendrá un valor constante de 98 PWM más un aporte que procede del control PI.
- Se vuelve a saturar la acción de control.
- También se satura la parte integral para evitar “el efecto windup”.

Los resultados que se obtuvieron se muestran en la siguiente gráfica:



Modo 4

Se dispone del mismo montaje que pare el modo 3.

Pasamos a comentar el código:

- Se aplica un control PI a cada rueda.
- Cada señal de control se satura y se mapea.
- Se satura también la parte integral del control para evitar “el efecto windup”.
- Adicionalmente se ha añadido una especie de filtro a la lectura del ultrasonido para que, si ocurre alguno de los siguientes casos, se tome como valor de la lectura el valor obtenido en el ciclo anterior:
    - Si sale una lectura negativa de la distancia medida.
    - Si la diferencia entre la lectura actual y la del ciclo anterior supera los 100 centímetros. Cada ciclo se ejecuta en un tiempo de milisegundos, esto es imposible de que suceda pero si nos fijamos en los archivos .log, a veces se ven algunos valores muy altos que con este diseño se han tratado de ignorar.

  El fragmento de código que aplica esto es el siguiente:

```
if(DISTANCIA1 < 0 || abs(DISTANCIA1-distancia1_ant) >= 100){          //Si sale la DISTANCIA1 < 0 : se queda con el valor anterior
 DISTANCIA1 = distancia1_ant;
  }
else{          //Solo coge el valor si DISTANCIA1 > 0
distancia1_ant = DISTANCIA1;
}
```

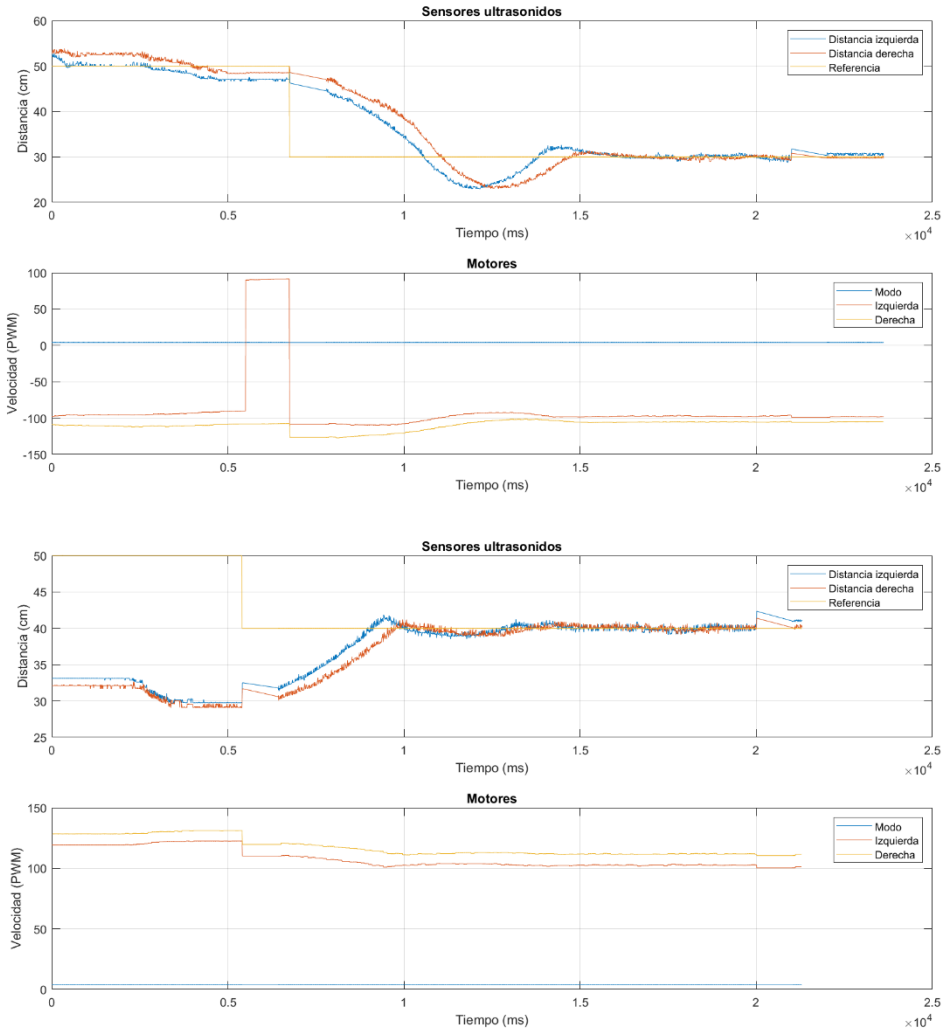
Ahora explicaré los diferentes casos que se han planteado en el código para llevar a cabo este modo:

Ultrasonido 1 delantero (cm)	Ultrasonido 2 trasero (cm)	Vel izquierda (PWM)	Vel derecha (PWM)	Caso
>= 35	>= 35	Vel izquierda	Vel derecha	Paso de 50 a 30
< 35	< 35	Vel izquierda	1.10 * Vel derecha	Paso de 30 a 40
Casos en las cercanías a la referencia				
Error 1 <= 0.5	Error 2 <= 0.5	Vel izquierda	Vel derecha	En la referencia
Error 1 > 0.5	Error 2 < -0.5	Vel izquierda	1.30 * Vel derecha	Robot apuntando a la pared
Error 1 < -0.5	Error 2 > 0.5	1.40 * Vel izquierda	1.20 * Vel derecha	Robot apuntando hacia fuera

“Vel izquierda” y “Vel derecha” hacen referencia a los valores PWM obtenidos de los controladores PI.

Cabe resaltar que, en nuestro caso, para un mismo valor de PWM la rueda izquierda iba más rápido, de ahí que poniendo la misma velocidad se logre pasar de 50 centímetros a 30 centímetros.

Los resultados obtenidos se muestran a continuación:



Se tiene además otro código llamado “Modo4_generico.ino” que trataba de hacer este primer código que se ha comentado algo más genérico, ya que, el primer código está bastante enfocado para que funcione bien para los valores concretos del test (50 cm – 30 cm – 40 cm).

Este otro código es lo mismo para los tres casos del robot en las cercanías a la referencia, pero los dos primeros casos admiten que se le den cambios de referencia siempre que sean mayores a 5 centímetros. Se muestra ahora el código de estos dos últimos casos:

```
if(movimiento==2 && (Setpoint1 - Input1) <= -5 && (Setpoint2 - Input2) <= -5){   //Si estamos más alejados de la referencia, la izquierda debe ir más rápido
//(movimiento==1 && referencia==30 && abs(Setpoint1 - Input1) >= 15.0)               //Error = referencia - distancia

     AdelanteIzq(abs(Out1)*1.10);
     AdelanteDer(abs(Out2));
}

if(movimiento == 2 && (Setpoint1 - Input1) >= 5 && (Setpoint2 - Input2) >= 5){  //Si estamos más cerca de la pared que la referencia, la derecha debe ir más rápido
//(movimiento==1 && referencia==40 && abs(Setpoint1 - Input1) >= 5.0)
     AdelanteIzq(abs(Out1));
     AdelanteDer(abs(Out2)*1.10);
}
```

Aparte del cambio en esos dos casos, todo el resto del código es muy similar y aplica la misma idea.

Antes de pasar a los siguientes modos, al igual que ocurrió con los sensores ultrasonido, voy a explicar previamente cómo se trabajará con los encoders de velocidad.

Primero necesitaremos dos pines de interrupción en nuestra placa Arduino, uno para cada encoder, en nuestro caso hemos usado los pines 2 y 3, estos pines serán los que llamen a las interrupciones.

A dichas interrupciones se accede cuando el encoder detecta el paso de un polo del disco que es solidario al eje de la rueda.

En dichas interrupciones, tan solo se incrementa un contador (nº de tics). Esta variable nos permite saber cúantos tics se dan en una revolución de la rueda, dato importante para el cálculo de la velocidad de la rueda. Además, se calcula cuánto tiempo ha pasado entre la vez anterior que se entró a la interrupción y la actual. A este valor se le hace la inversa obteniendo unidades de s^1, es decir, frecuencia. Finalmente calculamos la velocidad en rpm.

Ahora pasaré a explicarlo en pasos todo lo mencionado anteriormente:

- Encoder detecta polo del disco y salta la interrupción.
- Incremento contador de tics y calculo la frecuencia:

$$f = \frac{1}{\text{Instante}_{\text{actual}} - \text{Instante}_{\text{anterior}}}$$

- Calculo la velocidad de la rueda como:

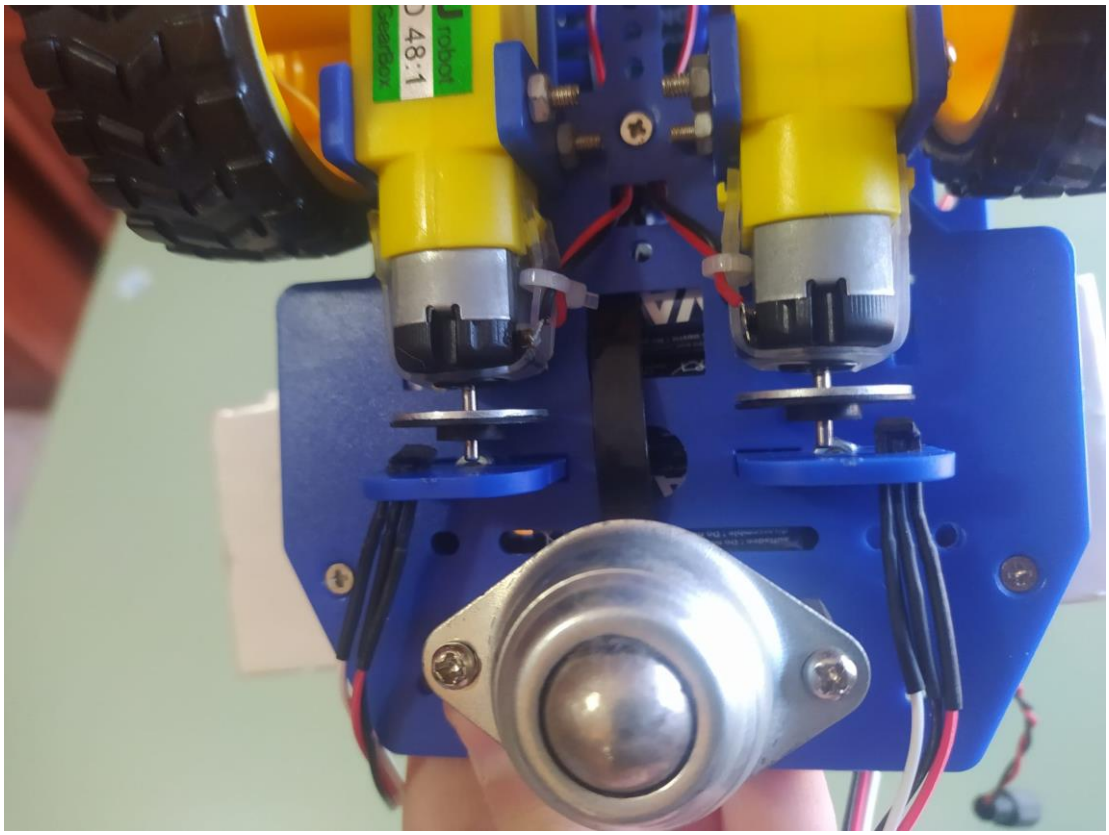
$$\text{Velocidad (rpm)} = f\left(\frac{1}{\text{s}}\right) * \frac{60\ (\text{s})}{\text{min}} * \frac{1}{\text{nº tics por vuelta}}$$

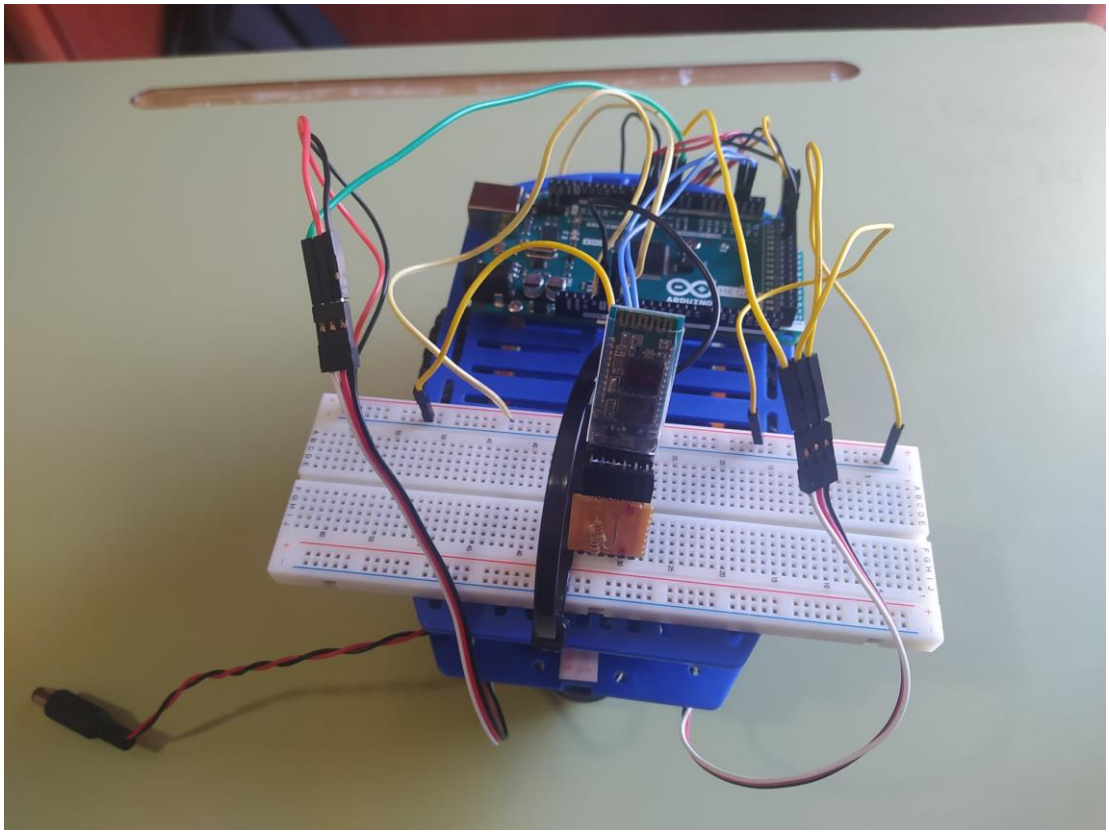
En nuestro caso el valor de los tics difiere según la rueda:

Nº tics por vuelta rueda izquierda	Nº tics por vuelta rueda derecha
182.5	198

Modo 5

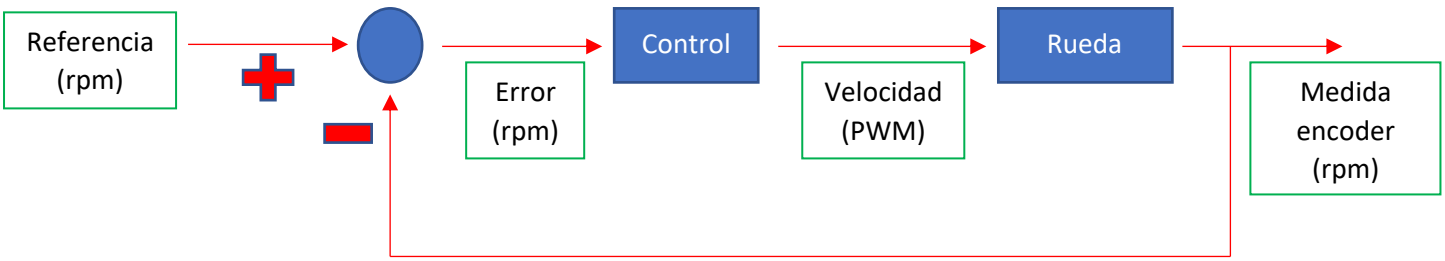
En este modo se introducen ya los encoders en el montaje de nuestro robot. En la imagen se ve cómo se han colocado:



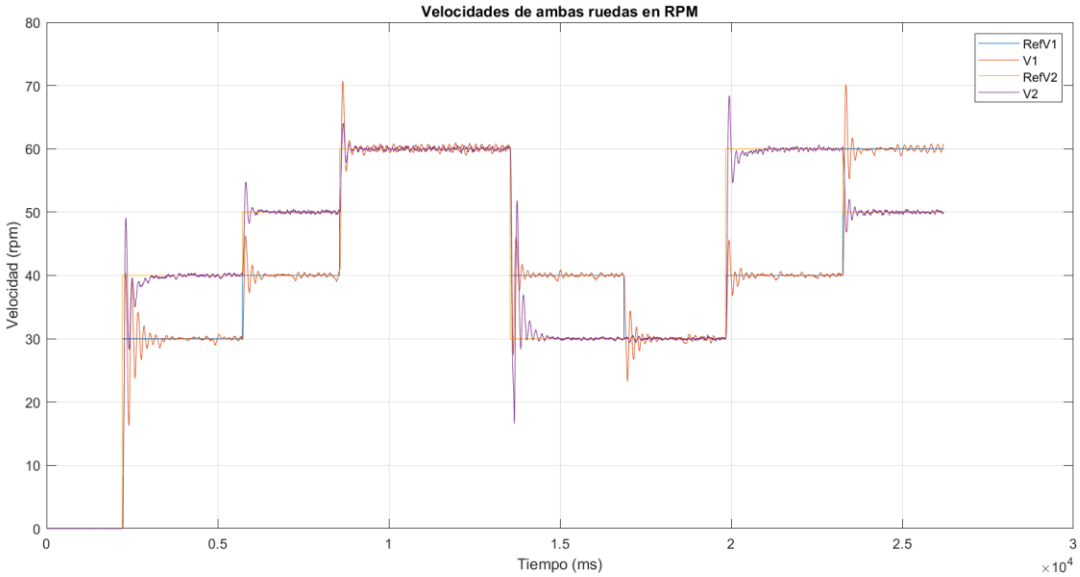


Una vez obtenida la velocidad de las ruedas, se aplica un control PI (proporcional + integral) a cada rueda como en los modos previos, un control a cada rueda. Se saturan también las señales PWM y se satura la parte integral del control para evitar “el efecto windup”.

El esquema de control ahora es el siguiente:



La gráfica obtenida al realizar el experimento es la siguiente:



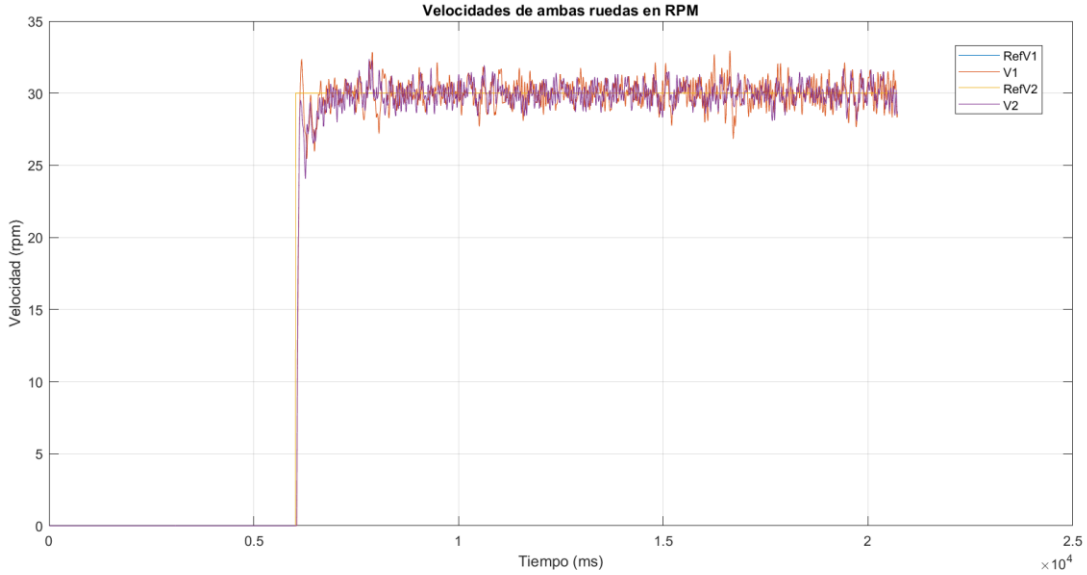
Adicionalmente se ha aplicado un filtro paso de baja al valor de velocidad obtenido a partir de la lectura de los encoders, con el fin de hacerla menos ruidosa. El código para implementarlo es el siguiente:

```
// Filtro LP(Low-pass filter)  (25 Hz cutoff) para limpiar la señal ruidosa de la velocidad que se entrega a los controladores de ambas ruedas
VIzqFilt = 0.854*VIzqFilt + 0.0728*Vizq + 0.0728*VizqPrev;
VizqPrev = Vizq;
VDerFilt = 0.854*VDerFilt + 0.0728*Vder + 0.0728*VDerPrev;
VDerPrev = Vder;
```

Modo 6

Este modo simplemente consiste en usar el modo 5, estableciendo en nuestro caso una referencia de 30 rpm en ambas ruedas. El hecho de que sea un valor bajo de referencia, ayuda a que sea más fácil que permanezca en línea recta el robot.

La gráfica obtenida al realizar el experimento es la siguiente:



Modo 7

En este modo se hace uso de la odometría, esta se encarga de estimar la posición en la que se encuentra el robot a partir de la lectura de los encoders, en nuestro caso son tres medidas (x, y, φ) las dos primeras corresponden a la posición y, la última a la rotación del robot.

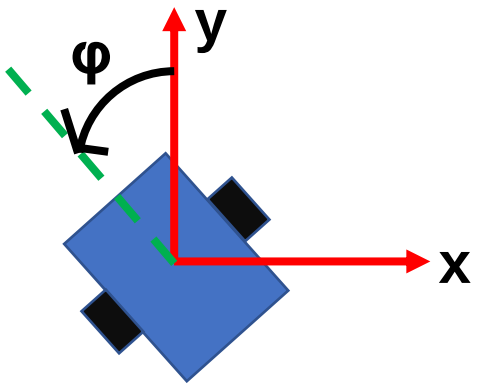
La función de odometría es la siguiente:

```
void Odometria(int TicsIzq, int TicsDer){
  resolucion_izq = 182.5;                    //Tics/vuelta rueda izquierda
  resolucion_der = 198.0;                    //Tics/vuelta rueda derecha
  diametro_rueda = 6.9;                      //Diámetro de la rueda en [cm]
  longitud_base = 8.0;                       //Longitud entre ambas ruedas en [cm]
  fi = pi * diametro_rueda/resolucion_izq;   //Factor de conversion entre tics y desplazamiento para la rueda izquierda
  fd = pi * diametro_rueda/resolucion_der;   //Factor de conversion entre tics y desplazamiento para la rueda derecha
  Li = fi * TicsIzq;                         //Longitud recorrida por la rueda izquierda
  Ld = fd * TicsDer;                         //Longitud recorrida por la rueda derecha
  Lc = (Li + Ld)/2;                          //Longitud recorrida por el centro del robot
  yaw = (Ld - Li)/longitud_base;             //Ángulo de orientación del robot
  yaw_rad = yaw*pi/180;                        //Paso a radianes
  y = Lc * cos(yaw_rad);                       //Coordenada y del robot
  x = Lc * sin(yaw_rad);                       //Coordenada x del robot (cuando avanza lm en línea recta el robot tendría (x,y)=(0,1))
```

Esta función a partir de los tics contados por las funciones de interrupción de los encoders, se encarga de calcular las tres magnitudes requeridas. Para ello simplemente se ha aplicado las ecuaciones mostradas en la siguiente fuente:

https://www.youtube.com/watch?v=XWApJxz9laM&ab_channel=TutosIngenieria

El esquema del robot con los ejes es el siguiente:

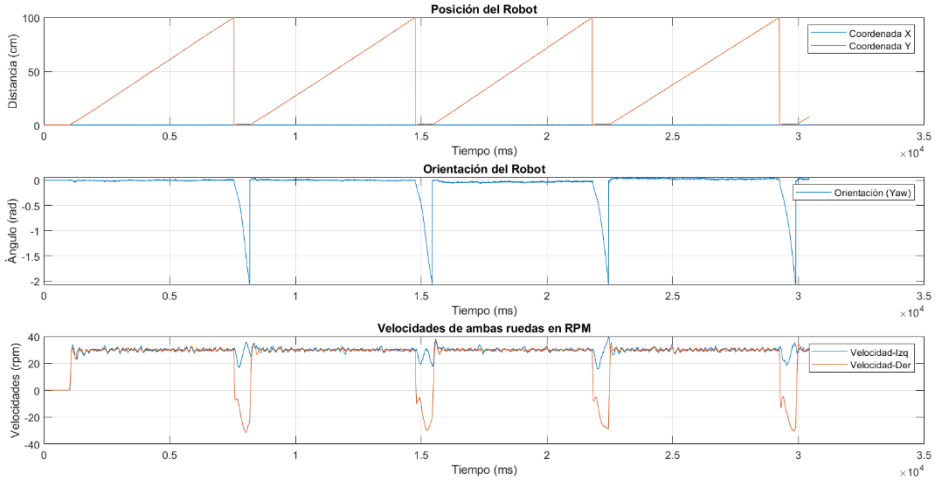


Con todo esto se ha creado una máquina de estados con dos estados, el primero para avanzar 100 cm hacia adelante (realizando una trayectoria en línea recta) y el segundo para girar 90º en sentido horario:

```
switch(estado){
  //En el estado 0:
  case 0:
  tiempo_anterior = millis();
    codigo_main();         //Se realiza el código main, que incluye control de velocidad de las 2 ruedas + contr
     delay(5);
      tiempo_actual = millis();
      delta_tiempo = tiempo_actual-tiempo_anterior;
      Prints();
    if(y >= 100){          //Una vez se llega a 1m (100cm):
      Parar();             //Se para el robot
      ContTicsIzq = 0;     //Volvemos a estar en "(0,0)", lo cual se realiza reseteando ambos contadores de tics
      ContTicsDer = 0;
      estado = 1;          //Se pasa al estado 1
    }
      break;
```

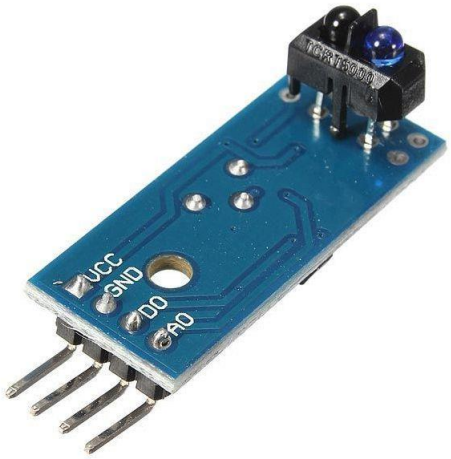
```
case 1:
  tiempo_anterior = millis();
  ContTicsDerPivote = -ContTicsDer;                //Se invierte el signo del contador de tics del encoder derecho (emula una velocidad negativa, necesaria para el cálculo del ángulo girado)
  Odometria(ContTicsIzq, ContTicsDerPivote);       //Se cálcula la odometría teniendo en cuenta la "velocidad negativa de la rueda derecha"
  AdelanteIzq(95);                                 //Se hace avanzar la rueda izquierda y retroceder la rueda izquierda a la vez.
  AtrasDer(115);                                   //Con estos valores de PWM el pivotado se produce de manera correcta.

  float frecuenciaIzq = 0.0;  //Frecuencia con la que se entra en la interrupción del encoder izquierdo
  float frecuenciaDer = 0.0;  //Frecuencia con la que se entra en la interrupción del encoder derecho
  ATOMIC_BLOCK(ATOMIC_RESTORESTATE){
    frecuenciaIzq  = frecuencia_izq;   //Se recogen los valores procedentes de las interrupciones dentro de un bloque ATOMIC, para evitar errores
    frecuenciaDer  = frecuencia_der;
  }

  // Convertir tics/s a RPM (cálculo de velocidad de cada rueda en [rpm])
  float Vizq = frecuenciaIzq/182.5*60.0;
  float Vder = -frecuenciaDer/198.0*60.0;

  // Filtro LP(Low-pass filter)  (25 Hz cutoff) para limpiar la señal ruidosa de la velocidad que se entrega a los controladores de ambas ruedas
  VIzqFilt = 0.854*VIzqFilt + 0.0728*Vizq + 0.0728*VizqPrev;
  VizqPrev = Vizq;
  VDerFilt = 0.854*VDerFilt + 0.0728*Vder + 0.0728*VDerPrev;
  VDerPrev = Vder;
  delay(5);
  tiempo_actual = millis();
  delta_tiempo = tiempo_actual-tiempo_anterior;
  Prints();

  if(yaw <= - 2.05){                                //Sabiendo el yaw (orientación del robot), una vez se ha girado 90º (debería ser -pi/2, pero no giraba 90º con ese valor):
    Parar();                                        //Se para el robot
    ContTicsIzq = 0;                                //Volvemos a estar en "(0,0)", lo cual se realiza reseteando ambos contadores de tics
    ContTicsDer = 0;
    estado = 0;                                     //Se pasa al estado 0. De esta manera, el robot estará realizando cíclicamente ambos procesos, realizando el cuadrado de 1m de lado
  }
    break;
```

Para el movimiento en línea recta (estado 1), se aplica el mismo control que para los modos 5 y 6. Se obtuvieron los siguientes resultados:



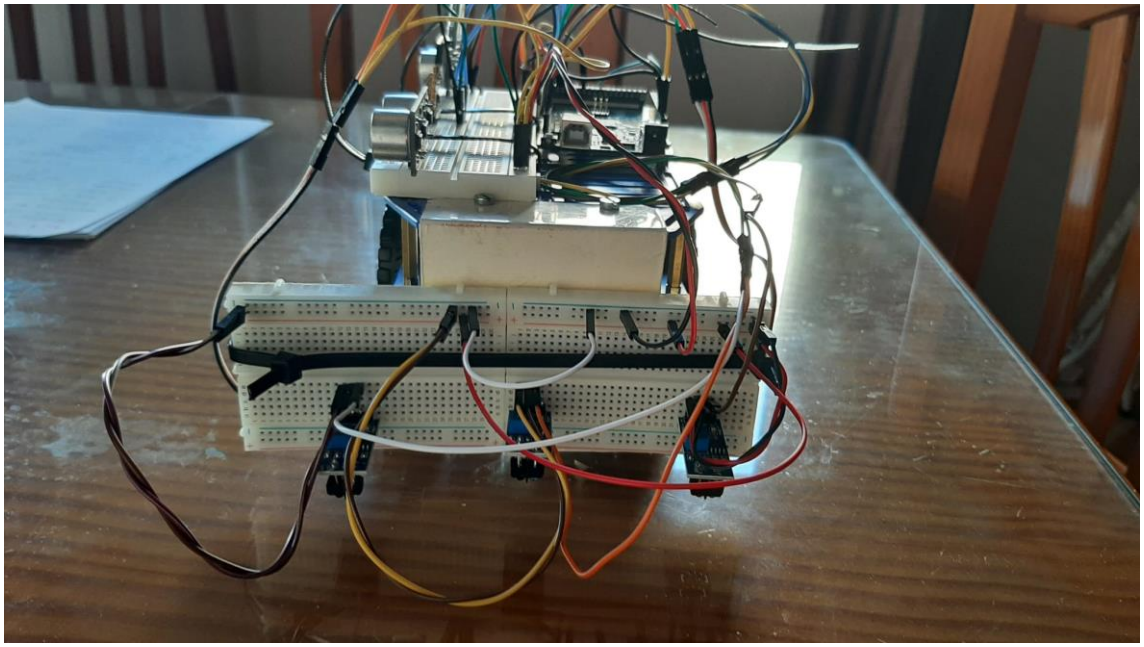
Modo libre

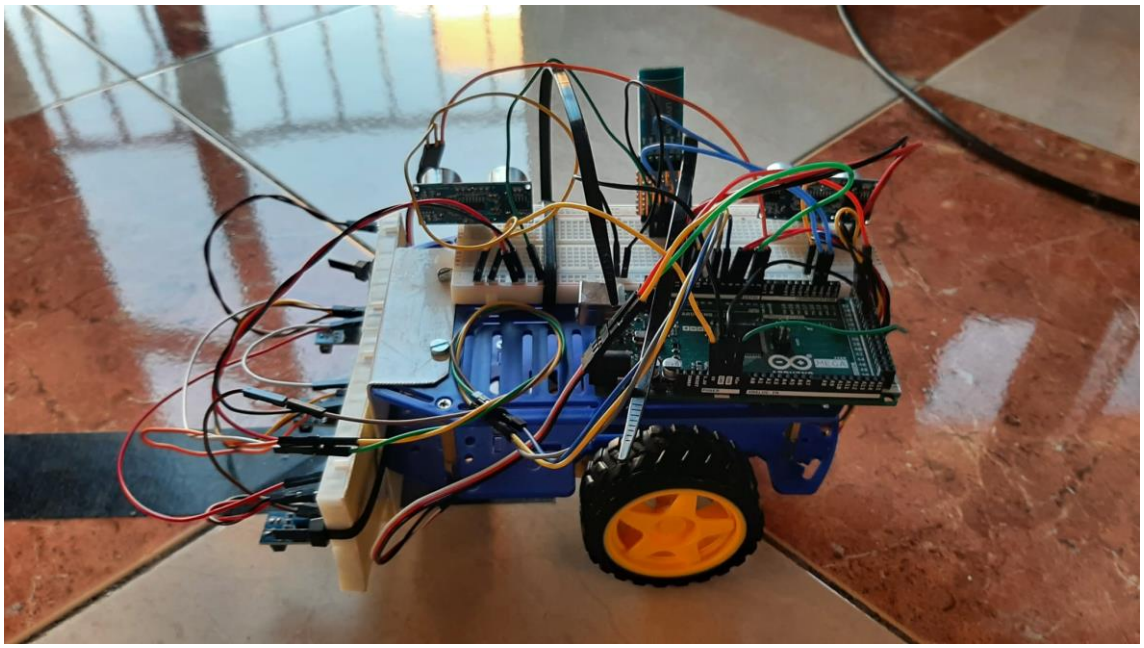
Para el modo libre, se propone la realización de un siguelíneas haciendo uso de tres sensores infrarrojos tipo TCRT5000, uno centrado en el robot, y otros dos situados uno a cada lado. A continuación, se muestra una imagen de estos sensores:

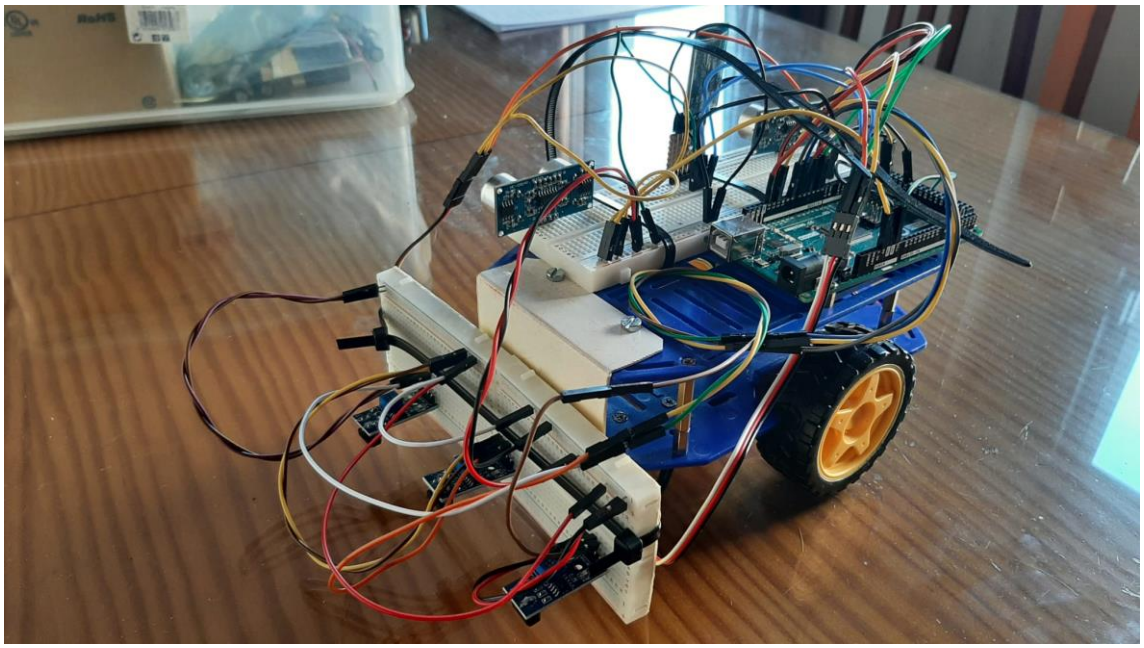


Se necesita conectar la pata de alimentación a 5 voltios, la pata de GND a la tierra del circuito y, la señal D0 será una señal booleana que valdrá ‘1’ cuando el sensor detecte y ‘0’ cuando no detecte. La señal A0 da la misma información que D0 pero en analógico.

Montaje de los sensores en el robot móvil:







El uso de este sensor es muy cómodo, puesto que no hay que estar por ejemplo como ocurre con el ultrasonido enviando pulsos y esperando su recepción, basta con tener las patas de alimentación y tierra conectadas, para poder leer la señal digital que indica si detecta o no.

Fuente de dónde se ha extraído la información para trabajar con estos sensores:

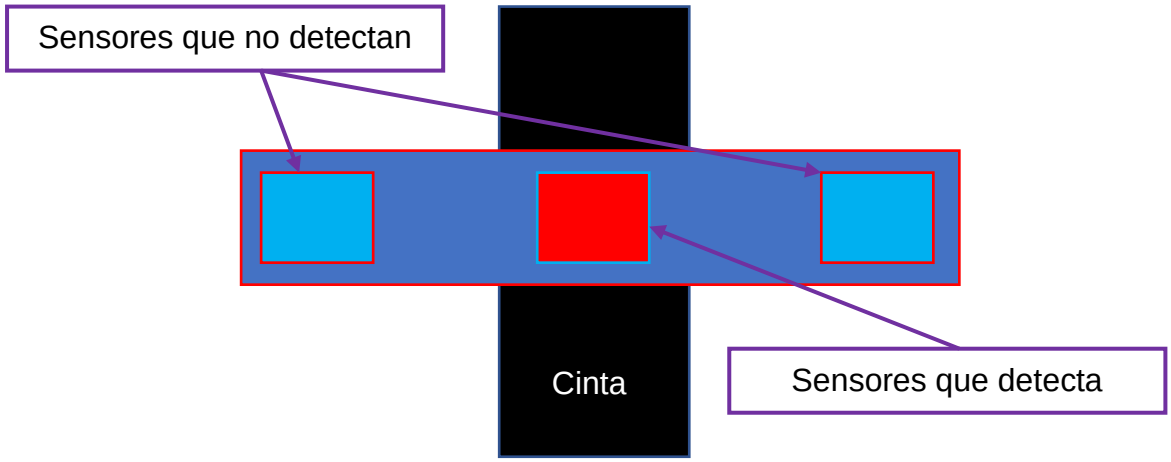
https://www.luisllamas.es/arduino-detector-lineas-tcrt5000l/

El circuito está descrito con cinta americana negra sobre un suelo rojo y blanco. Cuando un sensor infrarrojo esté sobre la cinta, se pondrá a ‘1’ su salida D0, mientras que los otros dos que no estén sobre la línea estarán a ‘0’.

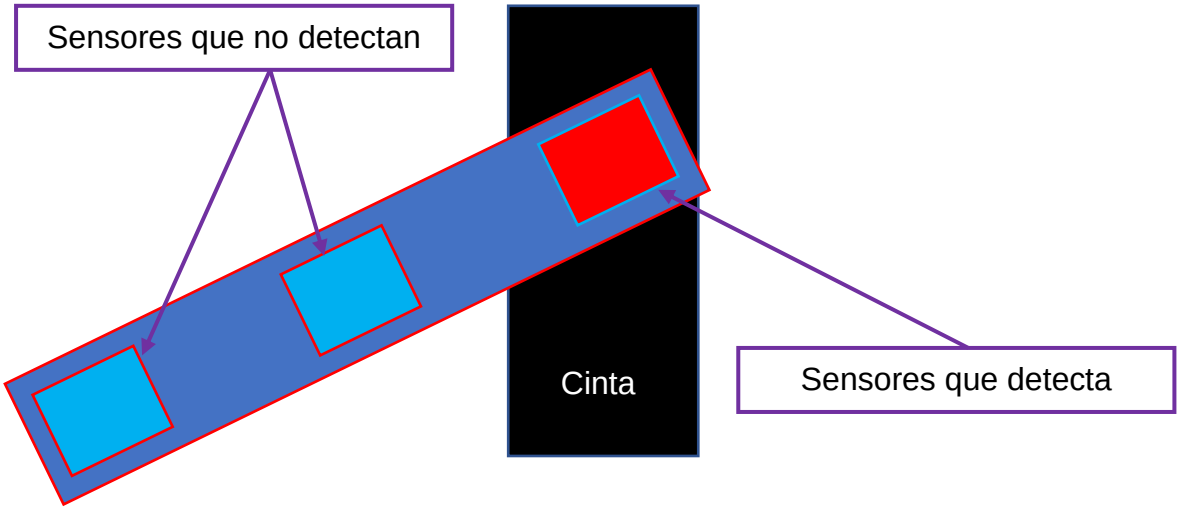
El código para este modo es básicamente el control en velocidad del modo 5 y 6, al que se le dan unos valores de velocidad en la referencia según qué sensores están detectando. Para ilustrarlo incluiré la parte del código donde se realiza:

```
if(flag_girobrusco==0){
    if(Lectura_Sensor_Izq==0 && Lectura_Sensor_Centro==1 && Lectura_Sensor_Dcha==0){ //Si detecta el sensor central significa que vamos sobre la línea, luego seguimos recto (misma velocidad en ambas ruedas)
      velocidad_izq_ref=35;
      velocidad_der_ref=35;
      flag_salido=0;
    }
    else if(Lectura_Sensor_Izq==0 && Lectura_Sensor_Centro==0 && Lectura_Sensor_Dcha==1){ //Si detecta el sensor derecho significa que nos estamos saliendo por la izquierda, luego la rueda izquierda debe ir más rápido
      velocidad_izq_ref=40;
      velocidad_der_ref=25;
      flag_salido=0;
    }
    else if(Lectura_Sensor_Izq==1 && Lectura_Sensor_Centro==0 && Lectura_Sensor_Dcha==0){ //Si detecta el sensor izquierdo significa que nos estamos saliendo por la derecha, luego la rueda derecha debe ir más rápido pa
      velocidad_izq_ref=25;
      velocidad_der_ref=40;
      flag_salido=0;
    }
    else if(Lectura_Sensor_Izq==1 && Lectura_Sensor_Centro==1 && Lectura_Sensor_Dcha==0){ //Si detecta el sensor izquierdo y el central significa que tenemos un ángulo agudo o recto hacia la izquierda, por ello la rued
      //Tengo que girar con más fuerza a la izquierda
      velocidad_izq_ref=25;
      velocidad_der_ref=60;
      flag_salido=0;
      flag_girobrusco=1;    //Activamos el flag de giro brusco
    }
    else if(Lectura_Sensor_Izq==0 && Lectura_Sensor_Centro==1 && Lectura_Sensor_Dcha==1){ //Si detecta el sensor derecho y el central significa que tenemos un ángulo agudo o recto hacia la derecha, por ello la rueda iz
      //Tengo que girar con más fuerza a la derecha
      velocidad_izq_ref=60;
      velocidad_der_ref=25;
      flag_salido=0;
      flag_girobrusco=1;    //Activamos el flag de giro brusco
    }
```

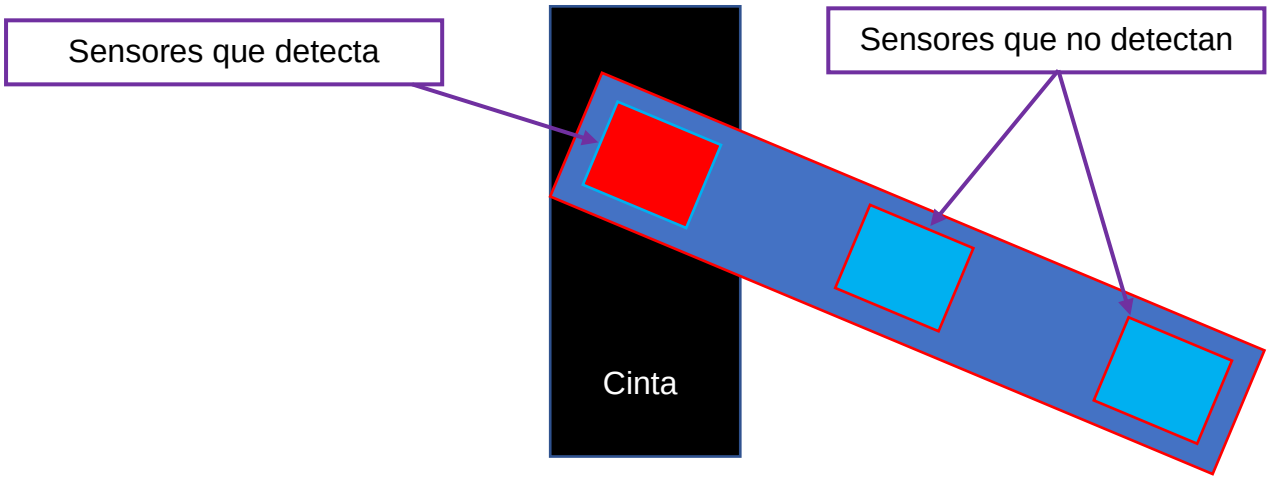
Si por ejemplo detecta el sensor central, significa que el robot está centrado encima de la línea, luego debemos seguir recto. Imagen para explicarlo mejor:



En el segundo caso, el sensor derecho es el que detecta, luego nos estamos yendo hacia la izquierda, debemos hacer que la rueda izquierda vaya más rápido. Imagen para explicarlo mejor:

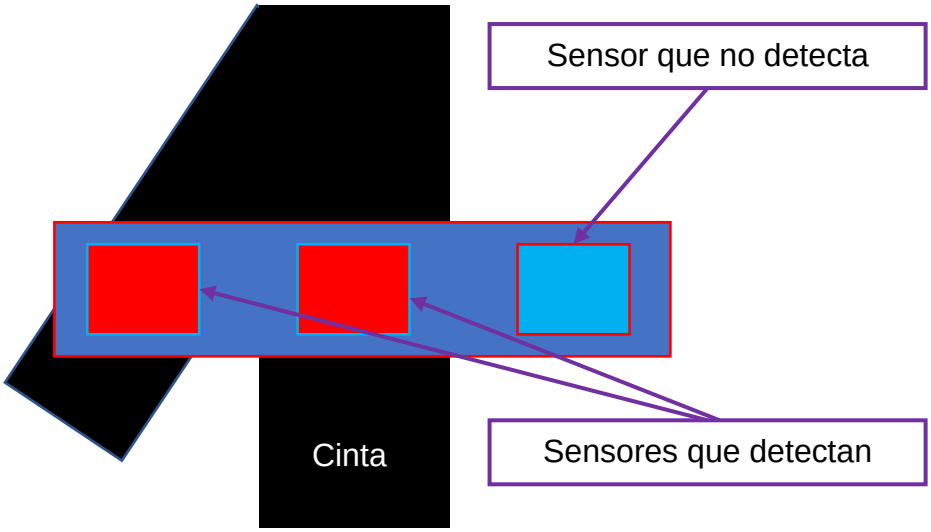


El siguiente caso es el contrario a este anterior, es el sensor izquierdo el que detecta porque nos estamos yendo hacia la derecha, luego debemos hacer que la rueda derecha vaya más rápido. Imagen para explicarlo mejor:

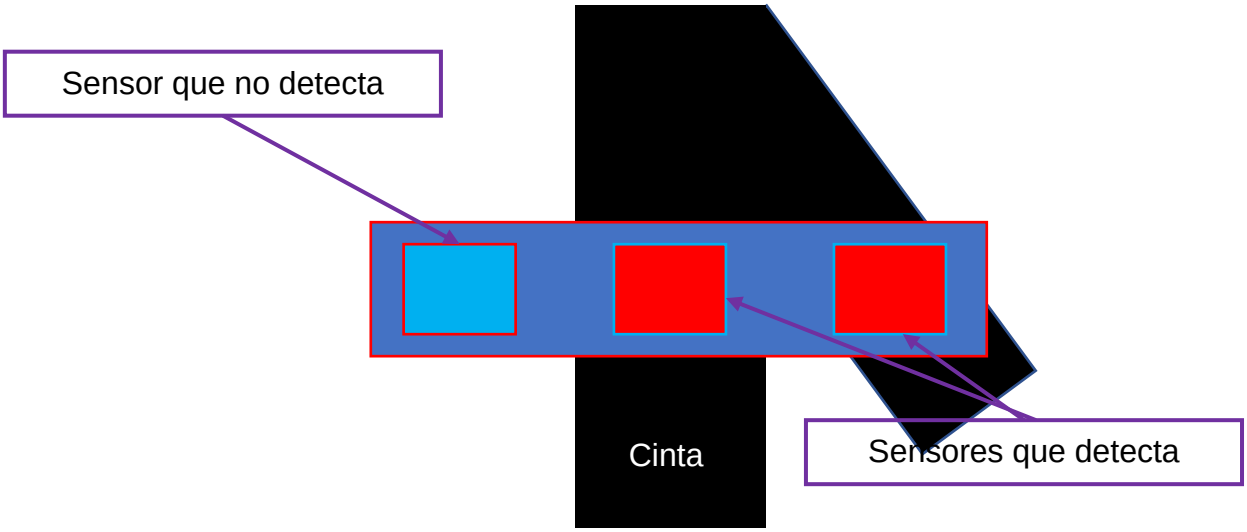


Los dos próximos casos se usan para realizar giros cerrados, ya sean ángulos agudos o ángulos rectos.

El caso en el que detecte el sensor central y el izquierdo, significa que debemos hacer un giro cerrado a la izquierda, luego la rueda derecha debe ir bastante más rápido. Imagen para explicarlo mejor:

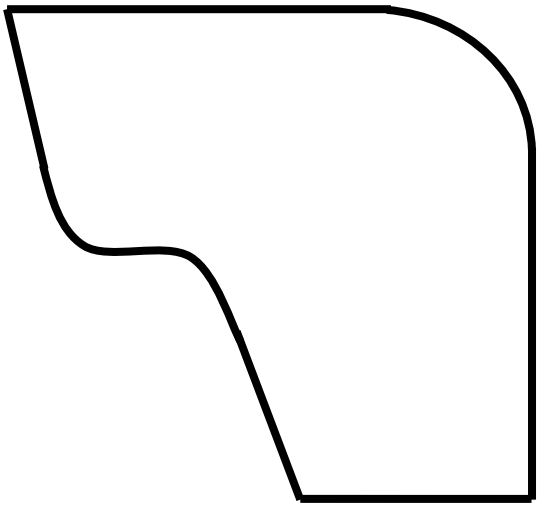


El último caso es el contrario a este anterior. Detecta el sensor central y derecho, debiendo girar la rueda izquierda bastante más rápido para lograr un giro a la derecha.

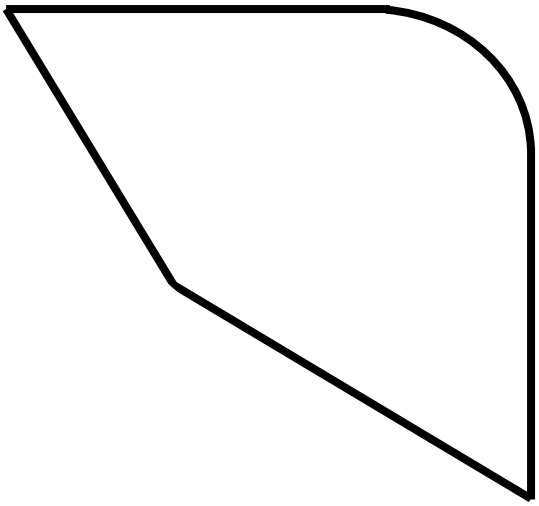


Como mecanismo de seguridad, se propone parar el robot en caso de que pasen 3 segundos sin que detecte ningún sensor.

Los circuitos que se han utilizado para probar esta funcionalidad tienen la siguiente forma:



Circuito 1



Circuito 2