Trabajo de curso

Robótica móvil

Laboratorio de Robótica

Pablo Carmona Benito
4º GIERM

Introducción

En este primer cuatrimestre, en la asignatura de Laboratorio de Robótica, una de las partes ha consistido en trabajar con un robot móvil de dos ruedas fijas y una rueda loca. El robot ha sido controlado con un Arduino Mega 2560, y para el conexionado nos hemos servido de una breadboard, todo proporcionado por el departamento.

La alimentación del sistema se realiza mediante unas pilas AA convencionales, por lo que ha sido necesario añadir un portapilas al montaje, añadiendo, de paso, el peso que conllevan 6 pilas.

El trabajo estaba dividido en 8 modos de funcionamiento, en los que el robot desarrollaba distintas funcionalidades. Para ellos nos hemos servido, además del material ya mencionado, de dos sensores de ultrasonidos y un módulo de comunicación Bluetooth.

Dicho esto, procedemos a explicar cada uno de los modos trabajados. En nuestro grupo (formado por otros dos alumnos además de yo mismo: Daniel Ríos Navarro y Juan José Calderón) hemos llegado con éxito total al modo 3. El modo 4 lo hemos desarrollado sin llegar a su pleno funcionamiento, y en el modo 5 hemos escrito completamente el código, y creemos que está todo correcto, pero no obtenemos resultados coherentes. Más adelante lo analizamos con mayor detenimiento.

Tareas a realizar (I)

Este es un apartado previo al resto del trabajo, en el que hemos montado físicamente el robot, y desarrollado un código básico para las acciones principales que más tarde hemos necesitado.

En primer lugar, los pines utilizados para el control de las ruedas son pines del puerto PWM del Arduino. Es decir, las ruedas de este robot funcionan con el método PWM. Hemos escogido los pines 10 y 5 para las entradas de las ruedas izquierda y derecha respectivamente (según nuestra configuración, pues la rueda loca la hemos dejado atrás). Por tanto, los pines 9 y 8 son los de avance y retroceso de la rueda izquierda, y otro tanto para los pines 7 y 6 respecto de la dererecha. Todos los pines descritos han sido configurados como OUTPUT, pues por ellos solo mandamos órdenes a la placa.

Para este primer paso, hemos desarrollado unas funciones que accionan las ruedas hacia delante o hacia atrás, o hacen al robot girar o pivotar, con distintas configuraciones de las ruedas. Explicaremos el funcionamiento basándonos en la función pivote_izda():

```
void pivote_izda () //gira en el sitio hacia la derecha
{
 digitalWrite (IN1, LOW);
 digitalWrite (IN2, HIGH);
 analogWrite (ENA, 200);
 digitalWrite (IN3, HIGH);
 digitalWrite (IN4, LOW);
 analogWrite (ENB, 200);
}
```

En las entradas IN1 e IN2, correspondientes a los pines 9 y 8, escribimos valores digitales 0 y 1. Por tanto, la rueda izquierda gira hacia atrás. Y lo hace con “velocidad” 200, según especificamos en ENA (pin 10) de modo analógico. Se puede deducir que la rueda derecha girará, entonces, hacia delante, con la misma velocidad. Como resultado, el robot “pivota” o, más bien, gira sobre sí mismo. En el bucle ponemos las funciones encadenadas para probar que todas funcionen.

```
void loop ()
{
palante ();                parar ();
 delay (5000);             delay (4000);
 patras ();                }
 delay (5000);
 pivote_dcha ();
 delay (3000);
 pivote_izda ();
 delay (3000);
 giro_dcha ();
 delay(5000);
 giro_izda ();
 delay(5000); …
```

Modo 1

En este modo hemos desarrollado un programa mediante el cual, a través de Bluetooth, el robot recibía una referencia y, ayudándose de los sensores de ultrasonidos colocados en la parte frontal, se posicionaba a la distancia requerida de la pared.

Primeramente establecimos la comunicación Bluetooth con el ordenador. Para ello, la placa ha de estar alimentada, y hay que configurar el BT. Incluimos la librería de comunicación serie del Arduino, le indicamos una baud rate y lanzamos la comunicación. La información recibida por BT la guardamos en un fichero .txt para luego representar su telemetría y analizar el funcionamiento del sistema.

Podemos pasarle la referencia por BT al robot. Para ello, la cadena que enviamos, la convertimos de caracter a entero, para poder tratarlo y controlar el robot:

```
if(Serial3.available())   // Si llega un dato por el monitor serial se envía al
puerto BT
  {
     Serial3.println("Referencia: "); //indica que le pasamos referencia
     cad = Serial3.readString();//-48;
       if(cad.length()>3 || dbt>300) Serial3.println("Referencia demasiado grande.
Introduzca una menor que 300");
     else{
      for(i=0;i<cad.length();i++)
      {
        if(cad.length()==3){
        dbt = (cad[0]-48)*100+(cad[1]-48)*10+cad[2]-48;
        }
        else if(cad.length()==2){
        dbt = (cad[0]-48)*10+cad[1]-48;
        }
        else dbt = (cad[0]-48);
      }
    }
    BT.write(dbt);
```

El control ideado es un control todo o nada, que activa las ruedas cuando hay una discrepancia entre la posición actual y la referencia.

```
if(distanciad>dbt && distanciai>dbt)
 {
  if(errd==0 || erri==0){
  parar();
   }
  else if(errd <=5 || erri <=5){
   palante(75);
   delay(10);
  }
  else{
```

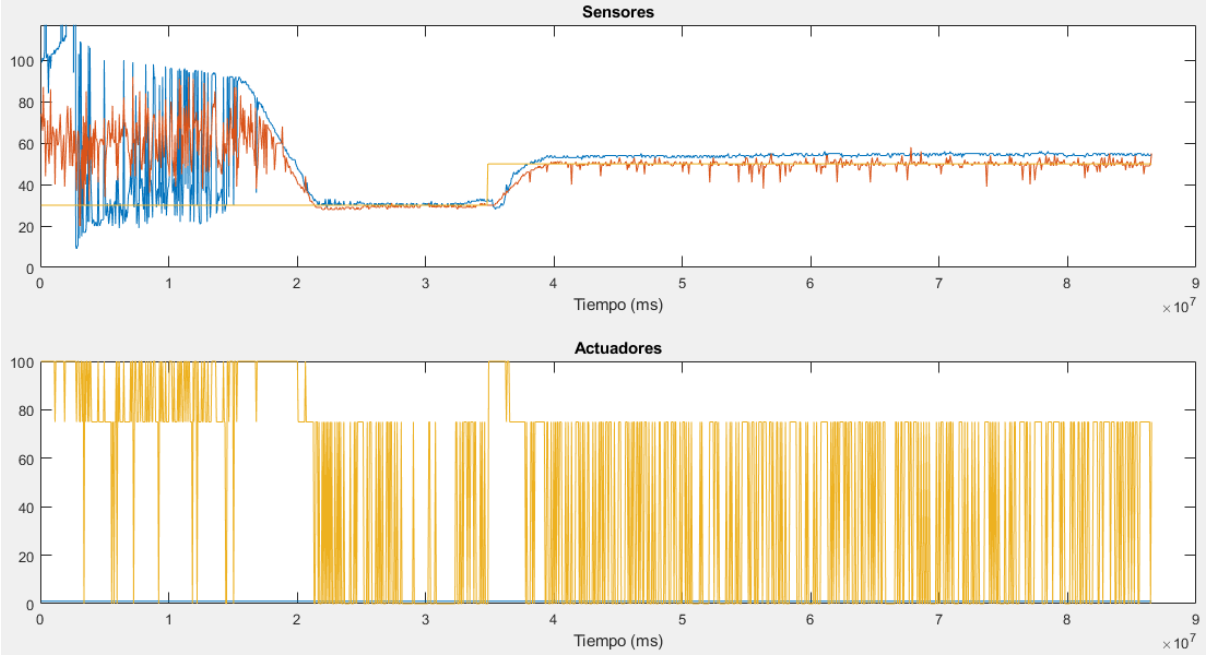
```
    palante(100);
    delay(10);
  }
 }

 else if(distanciad<dbt || distanciai<dbt)
 {
   if(errd==0 || erri==0){
  parar();
   }
  else if(errd<=5 || erri <=5){
   patras(75);
   delay(10);
  }
  else{
    patras(100);
    delay(10);
  }
 }
 else{
  parar();
 }
```

Hemos incluido unos márgenes para reducir la velocidad del robot, de manera que se reduzca también el error, y no desemboque en un sistema inestable.

Por último explicamos el funcionamiento de los sensores de ultrasonidos: se envían ondas en un momento determinado, registrando el tiempo, y se recoge después el rebote de esas ondas, registrando asimismo el tiempo transcurrido, de manera que se puede deducir la distancia a la que se encuentra el obstáculo contra el que han rebotado.

```
//activar y leer ultrasonidos
  digitalWrite(trigd,LOW);
  delay(0.002);
  digitalWrite(trigd,HIGH);
  delay(0.01);
  digitalWrite(trigd,LOW);
  delay(0.01);
  duraciond = pulseIn(ecod,HIGH);

  digitalWrite(trigi,LOW);
  delay(0.002);
  digitalWrite(trigi,HIGH);
  delay(0.01);
  digitalWrite(trigi,LOW);
  delay(0.01);
  duracioni = pulseIn(ecoi,HIGH);
```

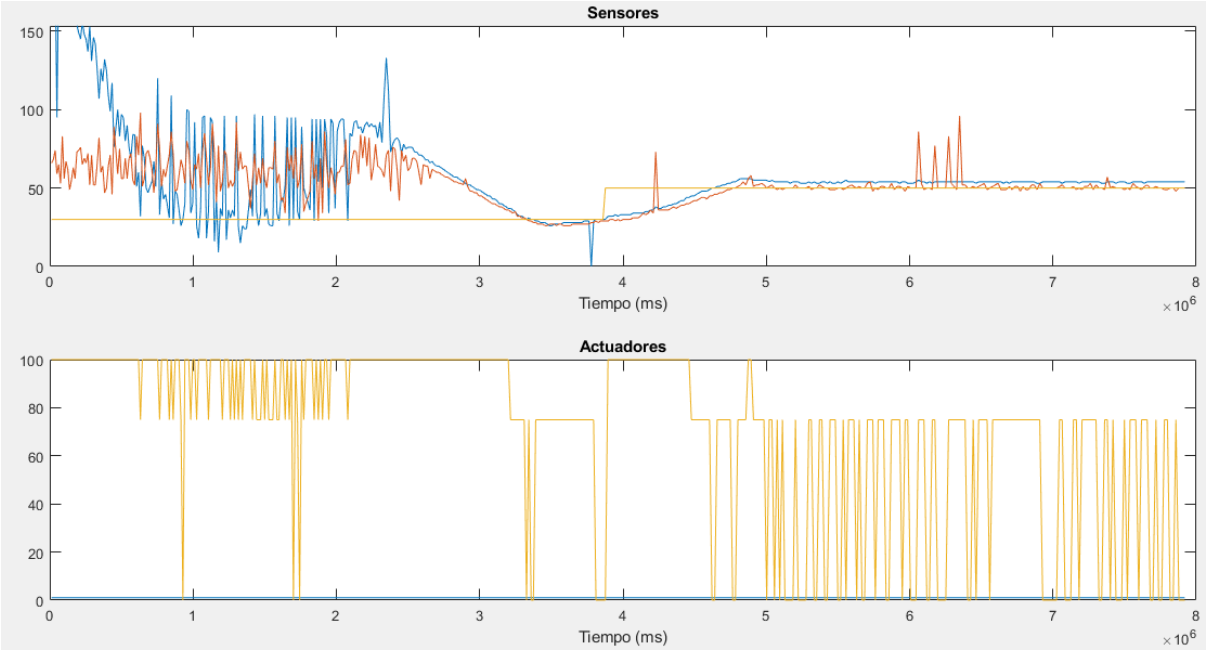
```
distanciad = duraciond/58;
distanciai = duracioni/58;

//leo tiempo del sistema
mytime = millis()-mytime;

delay(10);
errd = abs(dbt-distanciad);
erri = abs(dbt-distanciai);
```



Funcionamiento del robot en Modo 1. Se aprecia el control todo-nada

Modo 2

En este modo el objetivo era sumar al anterior que el robot se posicionase lo más perpendicular posible a la pared. Las actuaciones son más lentas para facilitar la perpendicularidad. Por lo demás, no añadimos nada al modo anterior, pues, sin querer, ya está programado de manera que se queda perpendicular a la pared.

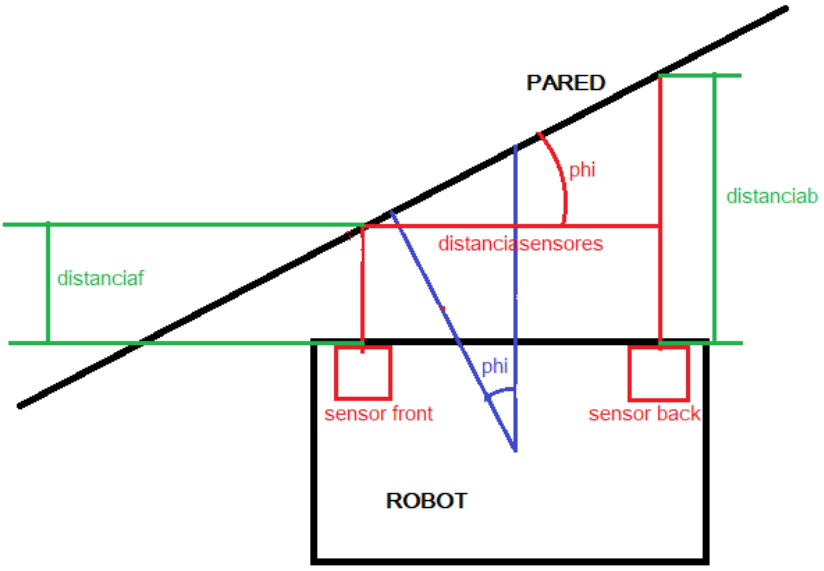


Los sensores tienen un error, por el cual se hace necesario incluir umbrales en los cálculos

Modo 3

En este modo se ha pretendido que el robot se desplace paralelo a una pared. Para ello hemos debido modificar la configuración del robot, colocando la breadboard con los sensores en un lateral, y por tanto desplazando la tarjeta de Arduino al lateral opuesto.

Tras intentarlo varias veces sirviéndonos de nuevo de las distancias medidas por los sensores, estableciendo umbrales, dimos con otro método mejor: el control del ángulo de inclinación respecto de la pared. Tomando como referencia (ángulo 0º) la perpendicular del robot con la pared, que pasa por medio de los sensores de ultrasonidos, hemos calculado el ángulo como:



phi = atan2((distanciab - distanciaf), distanciasensores)

Y por tanto la acción de control está destinada a hacer que phi = 0º.

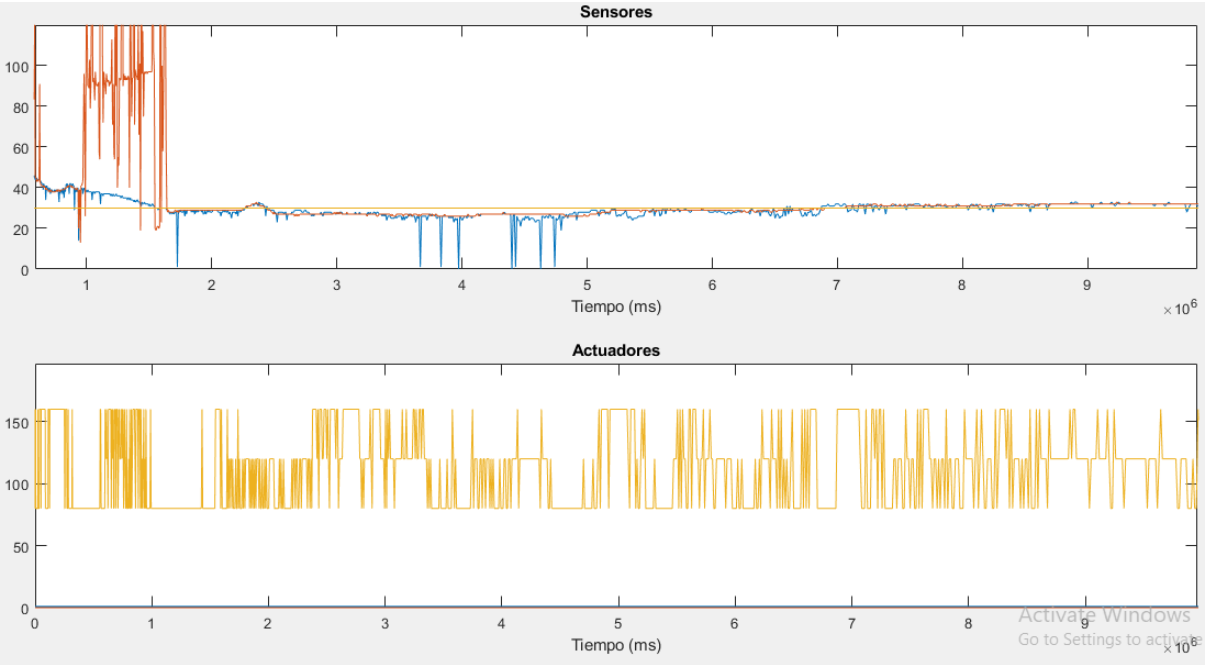
Para realizar esto hemos incluido la librería matemática, math.h. La distancia entre los sensores, colocados como los teníamos, es de 15cm. Para trabajar más cómodamente con el ángulo, lo hemos pasado a grados hexadecimales (aunque en la práctica, para este apartado no hemos necesitado esa conversión, pues el único valor de phi que hemos manejado ha sido 0º)

```
phi = phi*180/3.141592;

if(phi>0) //se esta acercando a la pared, quiero que gire a la izda=> larueda dcha tiene que girar mas rapido
    {
      velf = 80;
      digitalWrite(IN3,HIGH);
      digitalWrite(IN4,LOW);
      analogWrite(ENB,125);
      digitalWrite(IN1,HIGH);
```

```
    digitalWrite(IN2,LOW);
    analogWrite(ENA,velf);
    delay(5);}
  else if(phi<0){
    velf = 160;
    digitalWrite(IN3,HIGH);
    digitalWrite(IN4,LOW);
    analogWrite(ENB,125);
    digitalWrite(IN1,HIGH);
    digitalWrite(IN2,LOW);
    analogWrite(ENA,velf);
    delay(5);}
  else{
    velf = 120;
  digitalWrite(IN3,HIGH);
  digitalWrite(IN4,LOW);
  analogWrite(ENB,125);
  digitalWrite(IN1,HIGH);
  digitalWrite(IN2,LOW);
  analogWrite(ENA,velf);
  delay(5);}
```

En el desarrollo del programa hemos prescindido de las funciones creadas en inicio, con tal de manejar con más libertad los valores de las velocidades imprimidas en las ruedas. Como se ve, la condición es simplemente que si el ángulo es menor o mayor que 0º, el robot gira hacia un lado u otro. Da bastante buen resultado.

Nótese que la velocidad imprimida en la rueda derecha es normalmente algo mayor que la de la rueda izquierda, cuando se quiere que el robot vaya recto. Esto es porque está descompensado, una rueda gira más lento que la otra, y se añade una diferencia para tratar de compensarlo.



Modo 4

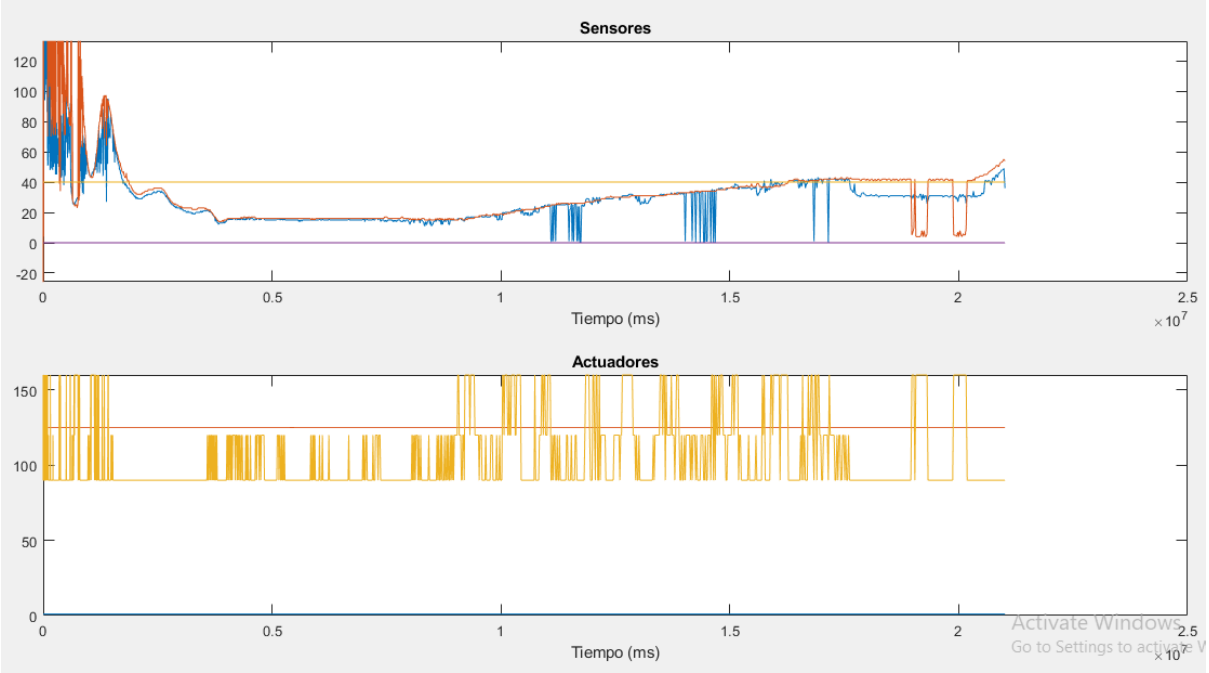
En este modo se añadía al anterior la posibilidad de cambiar la referencia a la que el robot se desplaza de la pared. Para ello hemos tenido que volver a incluir cálculos con distancia. Incluimos un pseudocódigo de lo realizado:

```
  if(((distanciaf+distanciab)/2 < dbt-6 || (distanciaf+distanciab)/2 > dbt+6))
    {
      if((distanciaf+distanciab)/2 < dbt-6) // && phi>-30)      {
       Giro a la izquierda;
      }
      else if((distanciaf+distanciab)/2 > dbt+6) // && phi>-30)
      {
       Giro a la derecha;
      }
      else
      {
        Hacia delante;
      }
    }
       else  if(((distanciaf+distanciab)/2  <  dbt-3  ||  (distanciaf+distanciab)/2  >
dbt+3))
    {
      if((distanciaf+distanciab)/2 < dbt-3) // && phi>-30)
      {
        Giro a la izquierda, más suave;
      }
      else if((distanciaf+distanciab)/2 > dbt+3) // && phi>-30)
      {
        Giro a la derecha, más suave;
      }
      else
      {
        Hacia delante;
      }
    }
```

Este código logra que el robot se sitúe a la distancia requerida de la pared. Después, incluimos el mismo código del modo anterior, que lo mantiene paralelo, con una condición para que sólo se ejecute una vez que ya esté a la distancia requerida: un flag que se pone a 1 cuando ya se da esa condición, y que se vuelve a poner a 0 cada vez que recibe una nueva referencia por BT.

El resultado es que, para referencias mayores que la distancia a la que ya se encuentra, lo hace con éxito. Sin embargo, para referencias menores, con frecuencia gira demasiado en la aproximación a la referencia, y pierde el contacto de los sensores con la pared, provocando una inestabilidad en el control. Lo hemos intentado solucionar

poniendo limitación de ángulo (de modo que no pueda girar más de cierto ángulo) pero no hemos conseguido darle arreglo.



Se observa cómo funciona. El tiempo de subida es lento. Hacia el final de la gráfica se daba un cambio de referencia, que desemboca en fallo del control. No lo hemos incluido.

Modo 5

Este ha sido el último modo en que hemos logrado algo. En los modos 6 y 7 no hemos podido desarrollar código útil, y por ello no los incluimos en la memoria. En este caso no hemos obtenido resultados aceptables, por fallos en la lectura de los encoders: dan valores sin sentido, posiblemente por un error en nuestra manera de leerlos en el programa o por tenerlos mal colocados. Aquí hemos incluido un controlador distinto, de tipo PID, incluyendo la librería PID_v1.h para Arduino.

Adjuntamos el código de lectura e interpretación de los encoders:

```
if(millis() - Time >=1000)//medimos el numero de vueltas en un segundo,
  {

    pulsosi = conti;
     RPMi = 60 * pulsosi/PPR;//calculamos rpm sabendo el número de vueltas en un
segundo y su reducción(rueda izquierda)
    pulsosd = contd;
     RPMd = 60 * pulsosd/PPR;//calculamos rpm sabendo el número de vueltas en un
segundo y su reducción(rueda derecha)


    conti = 0;         //reseteamos todas las variables
    pulsosi = 0;
    contd = 0;
    pulsosd = 0;
    Time = millis();

  }
```

Y las llamadas a los controladores para cada rueda:

```
//derecha-------------------
  Input = RPMd;
  controladord.Compute();
  Outputd = Output;

  if (Outputd < 0)
  {
    digitalWrite (IN1, LOW);
    digitalWrite (IN2, HIGH);
    analogWrite (ENA, abs(Outputd));
  }
  else if (Outputd == 0)
  {
    digitalWrite (IN1, LOW);
    digitalWrite (IN2, LOW);
    analogWrite (ENA, Outputd);
  }
  else
```

```
{
  digitalWrite(IN1, HIGH);
  digitalWrite(IN2, LOW);
  analogWrite(ENA, Outputd);
}
//-------------------------

//izquierda----------------
Input = RPMi;
controladori.Compute();
Outputi = Output;

if (Outputi < 0)
{
  digitalWrite (IN3, LOW);
  digitalWrite (IN4, HIGH);
  analogWrite (ENB, abs(Outputi));
}
else if (Outputi == 0)
{
  digitalWrite (IN3, LOW);
  digitalWrite (IN4, LOW);
  analogWrite (ENB, Outputi);
}
else
{
  digitalWrite (IN3, HIGH);
  digitalWrite (IN4, LOW);
  analogWrite (ENB, Outputi);
}
//-------------------------
```

Nota

En muchos de los ficheros .txt, de la telemetría, se nos ha pasado cambiar la variable modo, de manera que aparece como modo 1 cuando realmente (y claramente) no lo es.