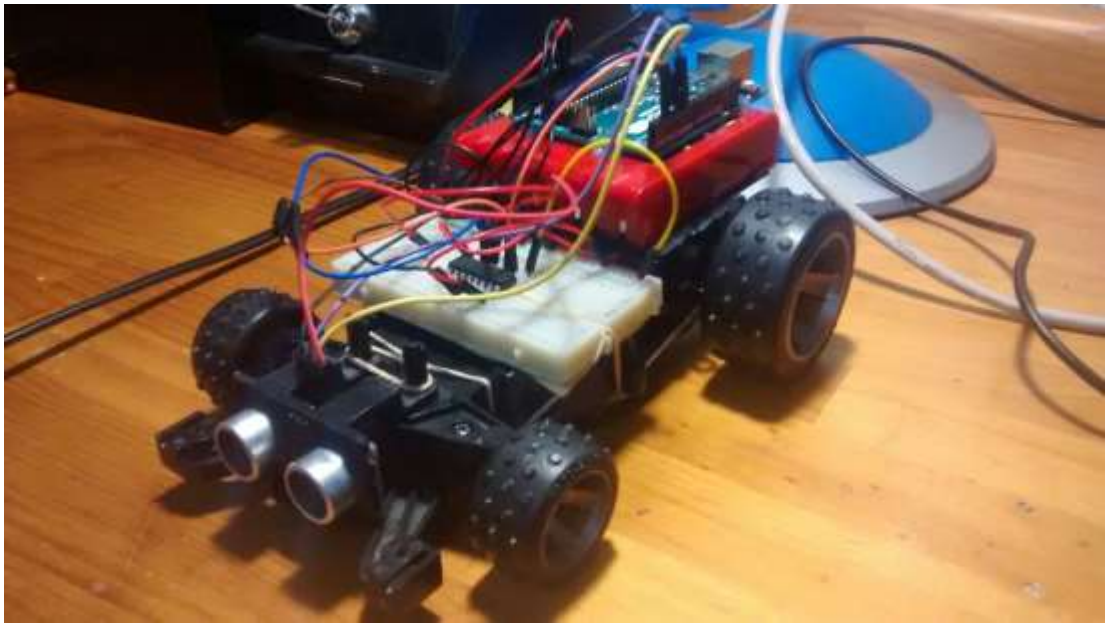
UNIVERSIDAD POLITÉCNICA DE MADRID
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA Y DISEÑO INDUSTRIAL
INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA
REGULACIÓN AUTOMÁTICA

# Control de distancia mediante PID en Arduino



José Luis Grande Morón

51.250

# Índice

# Introducción

El objetivo del presente proyecto es el diseño de un controlador Proporcional-Integral-Derivativo para la distancia entre un obstáculo y un coche dotado de un motor de corriente continua, sin dirección, pudiendo éste avanzar hacia delante y hacia atrás. La distancia de referencia podrá ser introducida por serial o definida en el código y el controlador autorregulará la tensión que precise el motor.

Usaremos el popular microcontrolador *Arduino UNO*. La enorme comunidad existente nos ayudará a salir de los diferentes problemas que se nos planteen. Para proporcionar intensidad suficiente al motor de corriente continua, necesitaremos usar un *puente H* (L293B).

Conseguiremos la realimentación gracias a un *sensor de ultrasonidos* (HC-SR04) que nos proporcionará la distancia con una simple operación.

Para la obtención del modelo de la planta, recurriremos al uso de datos experimentales ya que no disponemos de las características del motor de corriente continua y se nos hace imposible recurrir a un modelado matemático. Para ello, haremos uso de la herramienta *Ident* de Matlab.

Igualmente, el cálculo de las constantes del controlador se realizará vía Matlab con la funcionalidad *pidTuner*, buscando, dentro de lo posible, la mejor respuesta.

# Material

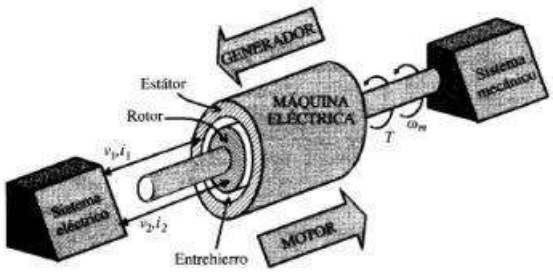
- Arduino UNO
- Circuito integrado L293B (Puente H)
- Sensor de ultrasonidos HC-SR04
- Protoboard
- Jumpers
- Fuente de 7.5V (5 x 1.5V)
- Pistola termofusible

# Descripción de elementos

**Planta**

Nuestra planta se trata de un viejo coche radiocontrol al que se le ha retirado toda la antigua electrónica y se le ha fijado permanentemente la dirección. El hecho de desconocer incluso el nombre del juguete original nos dificulta en exceso, imposibilita en la práctica, la intención de conocer las características internas de los actuadores. Esto nos obliga a tratar la planta como una caja negra, teniendo acceso a, tan solo, los dos cables del motor de corriente continua como entrada y pudiendo observar la distancia como salida.

Un motor de corriente continua es una máquina eléctrica rotativa. Como tal puede funcionar como generador o como motor, siendo reversible:




Cada máquina eléctrica rotativa consta de dos partes:

*Estator*: Parte fija de la máquina.

*Rotor*: Parte giratoria de la máquina.

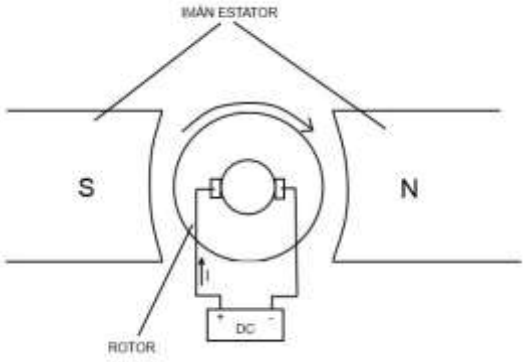
Poseyendo tanto medio mecánico como eléctrico, el entrehierro supone el medio de acoplamiento entre estos. Tanto el rotor como el estator poseen devanados para poder crear el flujo que circula por el entrehierro. Podemos distinguir entre:

*Devanado inductor*: es el que crea un flujo en el entrehierro.

*Devanado inducido*: en él se inducen corrientes por la variación del flujo.

Cuando la corriente eléctrica circula por el devanado del rotor, se crea un campo electromagnético. Este interactúa con el campo magnético del imán del estator. Esto deriva en un rechazo entre los polos del imán del estator y del rotor creando un par de fuerza donde el rotor gira en un sentido de forma permanente.

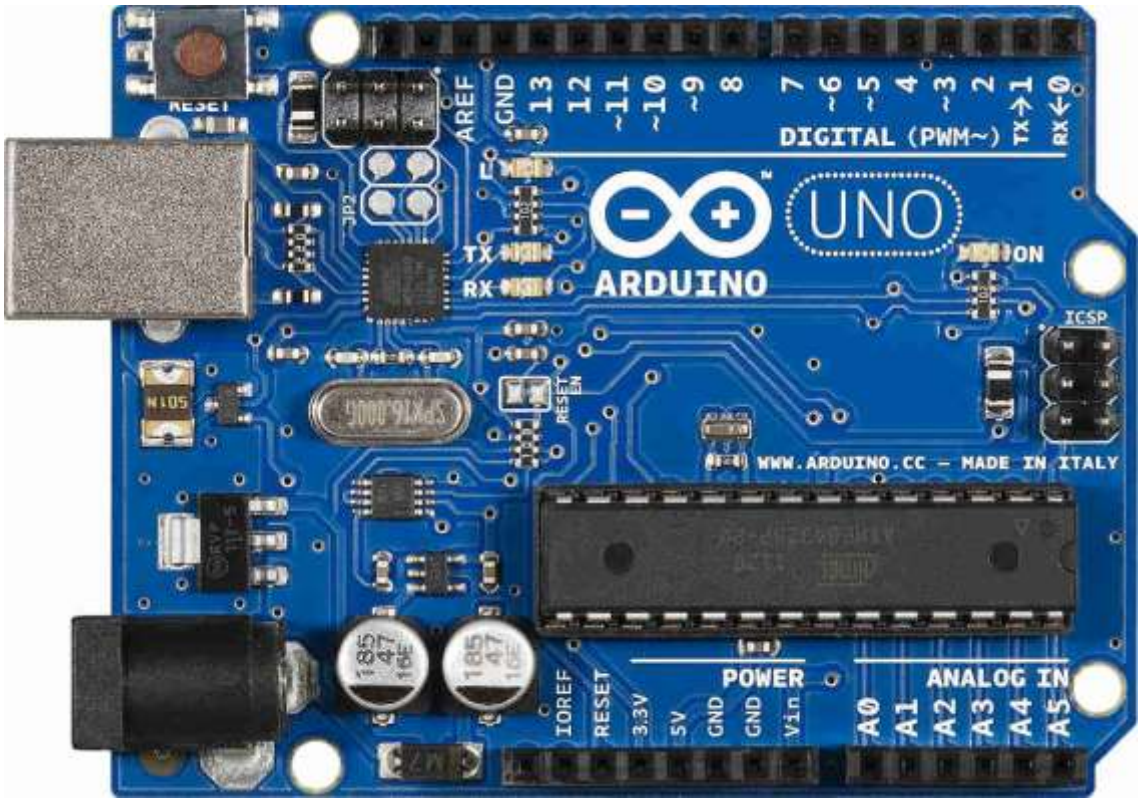
Si queremos cambiar el sentido de giro del rotor, tenemos que cambiar el sentido de la corriente que le proporcionamos al rotor; asique basta con invertir la polaridad de la pila o batería.




**Arduino UNO**

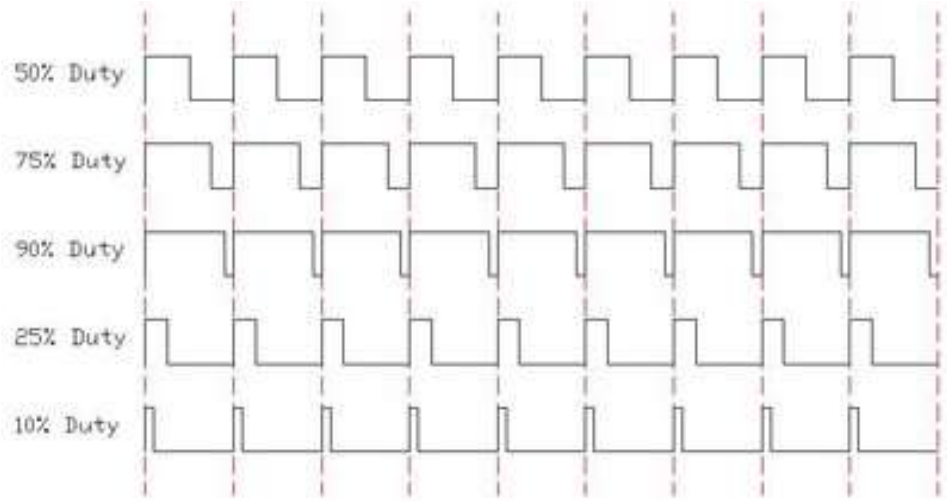
Arduino UNO es un microcontrolador basado en el ATmega328 donde programaremos el controlador.

.



Consta de una serie de pines de entrada-salida digitales y otras entradas analógicas. En mi caso, no usaré las entradas analógicas. Las salidas analógicas se obtienen gracias al PWM que poseen algunos de los pines digitales.

El PWM consiste en la regulación del duty cycle, generando un tren de pulsos digitales equivalente a una señal analógica.



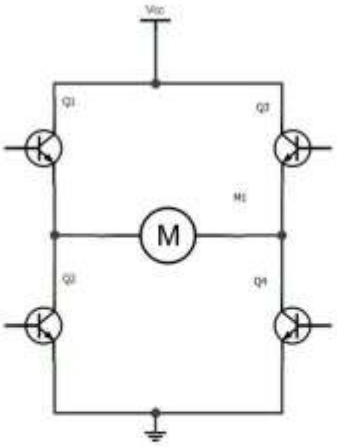

El sensor de ultrasonidos precisará de una salida digital y una entrada digital del Arduino. La salida del controlador es una señal analógica que va al motor usando los pines PWM.

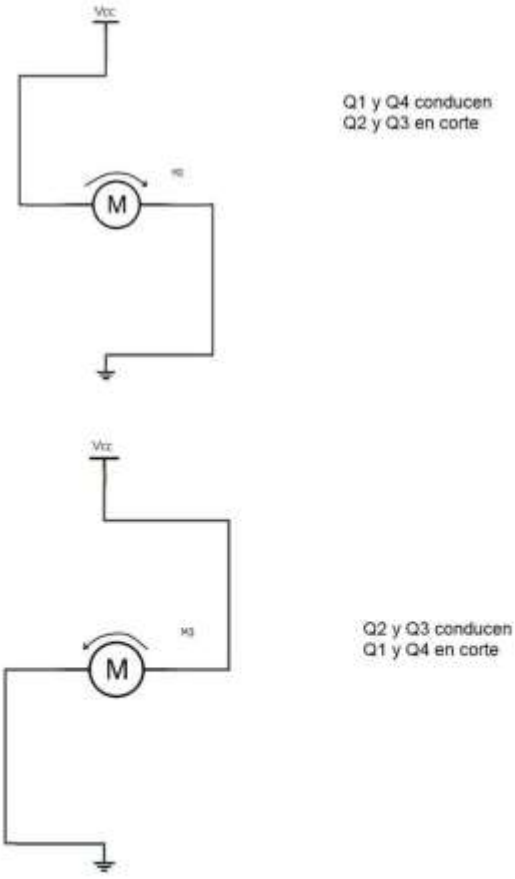
A parte de esto, tan sólo serán necesarios los pines de 5V y GND para alimentar el circuito integrado y el sensor de ultrasonido.

**Circuito integrado L293B**

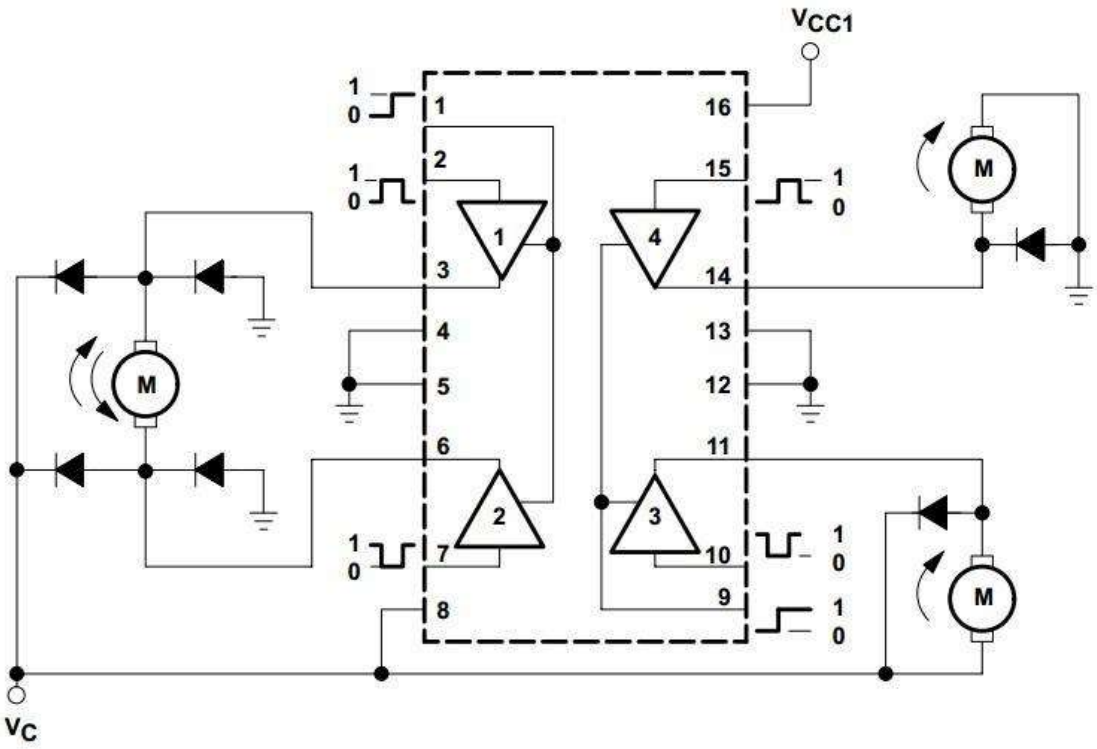
Las salidas del microcontrolador Arduino UNO están limitadas a 40mA, lo que es insuficiente para alimentar el motor DC. Para proporcionar la corriente necesaria, deberemos usar un driver como el L293B junto con una alimentación externa.

El L293B es un circuito integrado para controlar motores DC que usa un sistema puente en H. Este sistema sirve para controlar el sentido de giro de un motor DC usando cuatro transistores. Los transistores se comportan como interruptores siendo activados por la placa dos a dos, para que la corriente pueda circular en un sentido o en otro sobre el motor.




El L293B tiene dos puentes H y, según el datasheet, proporciona como máximo 1000mA de manera continua al motor, soportando un voltaje entre 4,5V y 36.




Utilizaremos la configuración de la izquierda de la última figura si bien los diodos representados fuera del circuito integrado están dentro del mismo a modo de protección, para evitar que al dejarle de llegar tensión, corrientes parasitas salgan de él hacia el integrado y la placa y los quemen.

Los pines 3 y 6 del integrado se conectan al motor, mientras que los pines 2 y 7 serán las entradas donde conectar las salidas PWM del Arduino. Estos dos últimos harán que el motor gire en un sentido o en el otro, dependiendo de la diferencia de tensión entre ambos.

**Sensor de ultrasonidos HC-SR04**

Usaremos un sensor de ultrasonidos para medir la distancia del coche al obstáculo. Este tipo de sensor funciona enviando un pulso sonoro y midiendo el tiempo que dicho pulso tarda en volver al sensor tras ser rebotado en un objeto, de manera similar a la que un murciélago mide distancias. Conociendo la velocidad del sonido, podemos sacar la distancia al objeto con un simple cálculo.

El sensor tiene cuatro patillas. Dos de ellas, Vcc y GND, se dedican exclusivamente a la alimentación del sensor; metiendo 5V a la patilla Vcc. La patilla *trigger* es la que controla la emisión de la onda, lanzándose cuando la misma está en estado HIGH. Por último, la patilla *echo* es la encargada de recibir la onda.

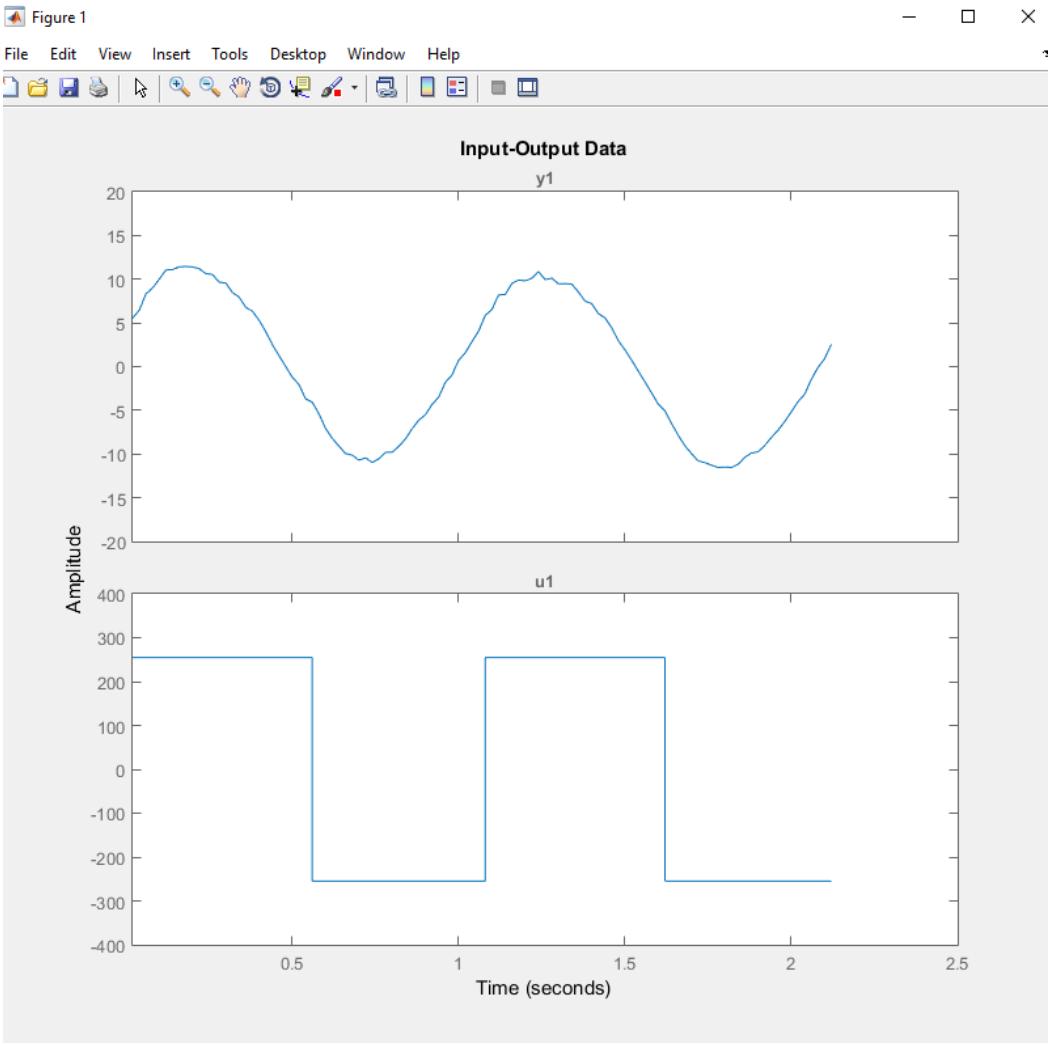

Y1
HC-SR04
Vcc
Trig
Echo
GND
T
J1
R

# Modelado de la planta

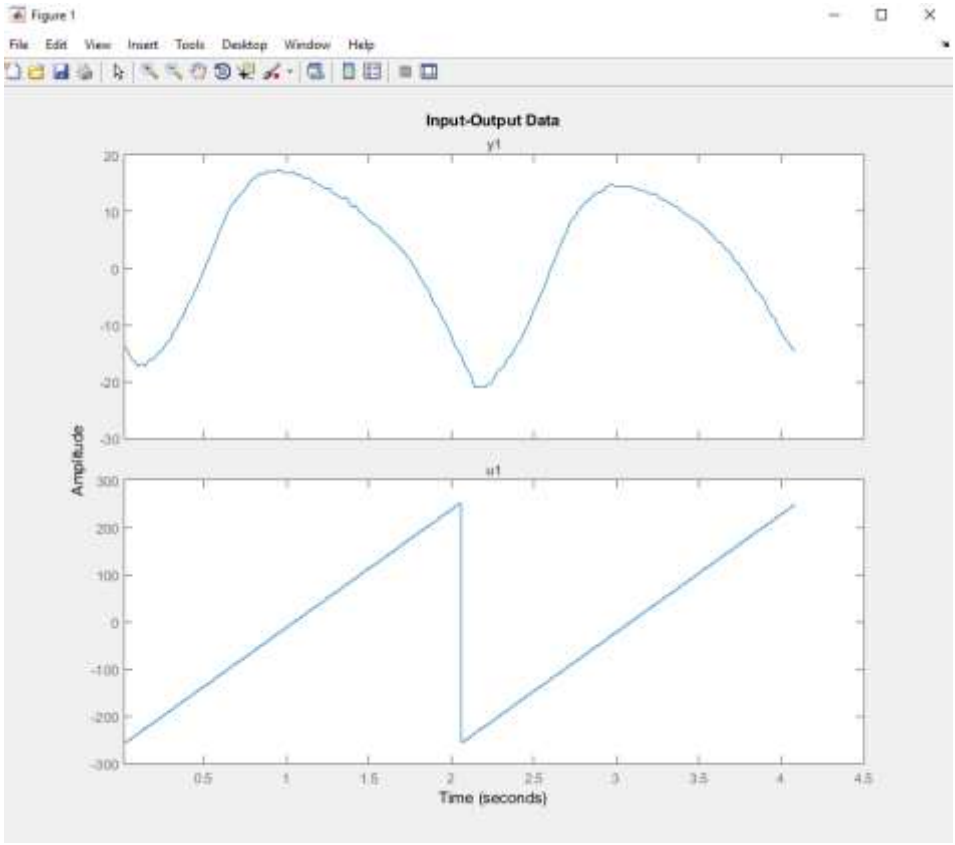
Como se ha mencionado anteriormente, no disponemos de datos físicos del motor y no podemos sacar un modelo teórico de la planta. Debemos, entonces, pensar en sacar el modelo mediante procesos experimentales.

El primer paso será generar varias ondas de entrada en forma de tensión y observar como varía la distancia. Para ello, realizamos varios programas en el IDE de Arduino que impriman por serial la tensión, la distancia y el tiempo transcurrido en cada paso por el bucle *loop*. Esos datos recibidos por el serial se pegan en un documento de Excel, fácilmente importables a Matlab en forma de vectores para su representación gráfica. A continuación las formas de onda generadas tanto de salida como de entrada, una vez eliminado el offset de la medida de la salida.

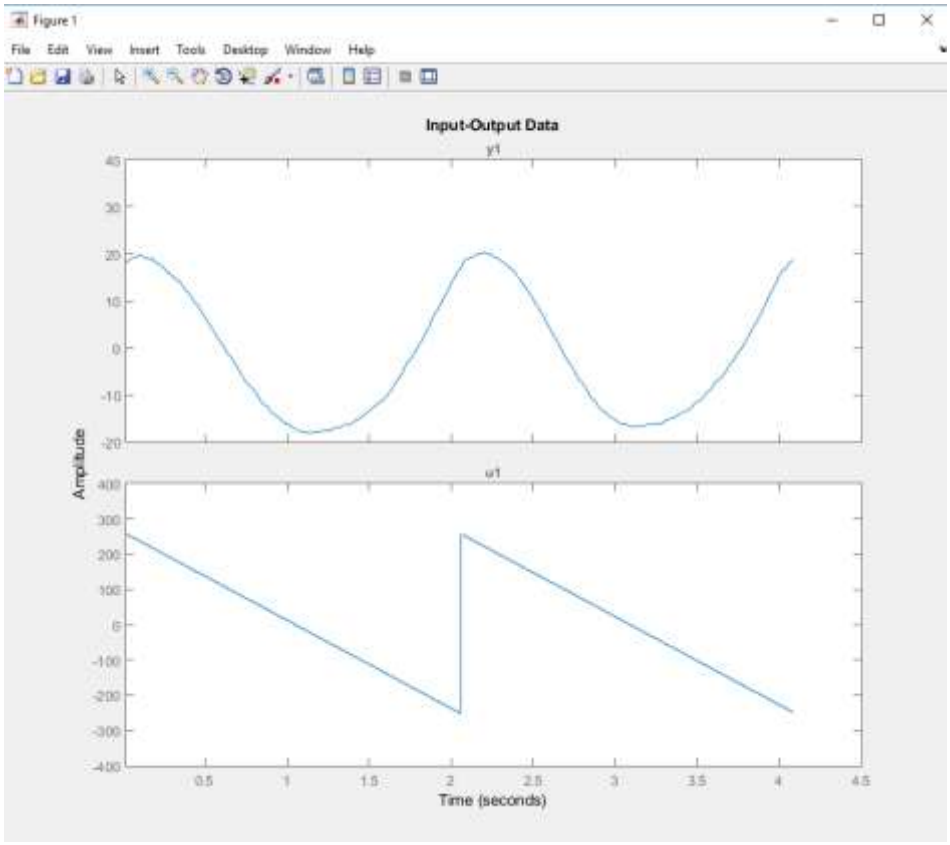
**Cuadrada**



**Rampapos1**




**Rampaneg1**



**Rampa2**

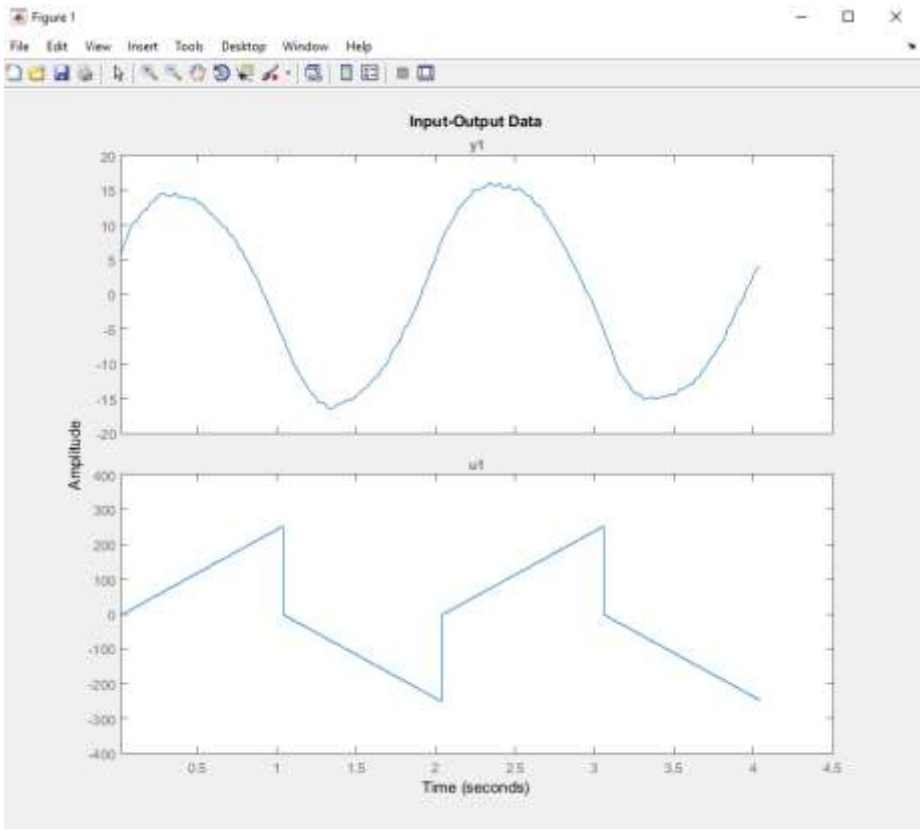


Figure 1
File Edit View Insert Tools Desktop Window Help
Input-Output Data
y1
u1
Amplitude
Time (seconds)

**Seno**

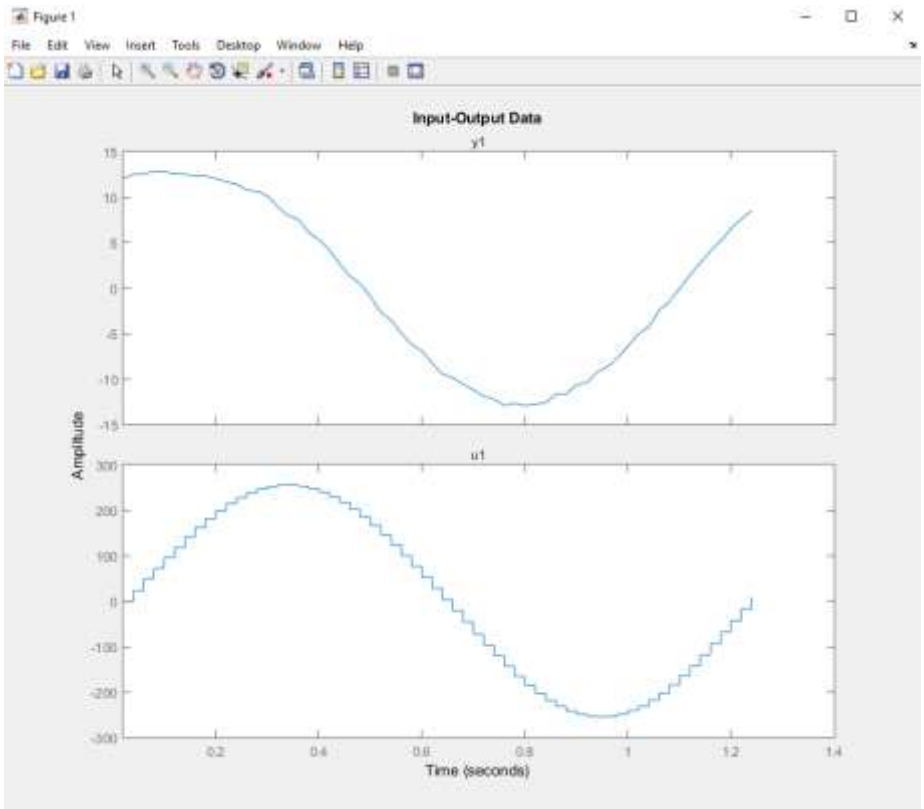


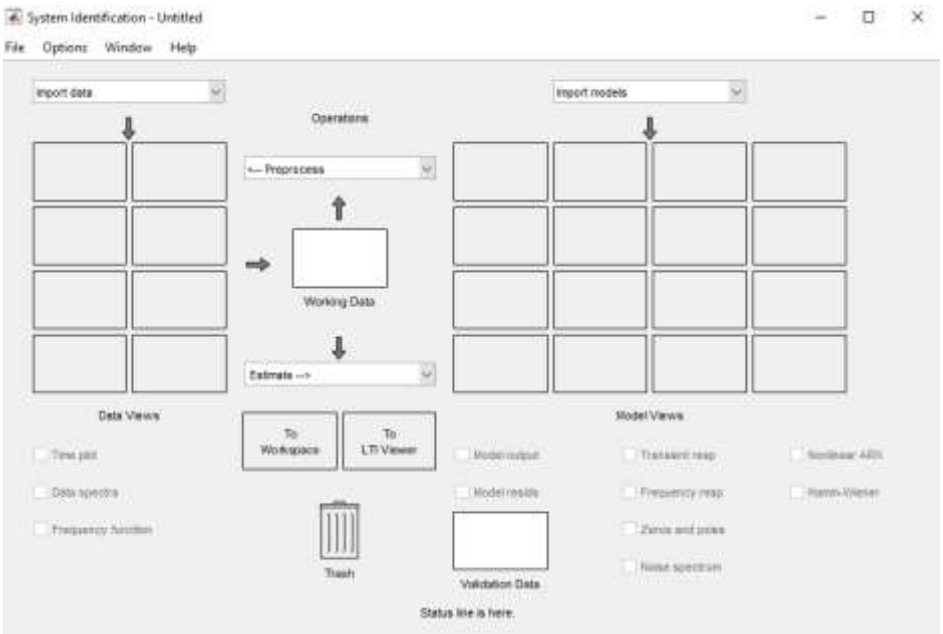
Figure 1
File Edit View Insert Tools Desktop Window Help
Input-Output Data
y1
u1
Amplitude
Time (seconds)

**Proceso**

Un ejemplo de script para eliminar el offset es el siguiente:

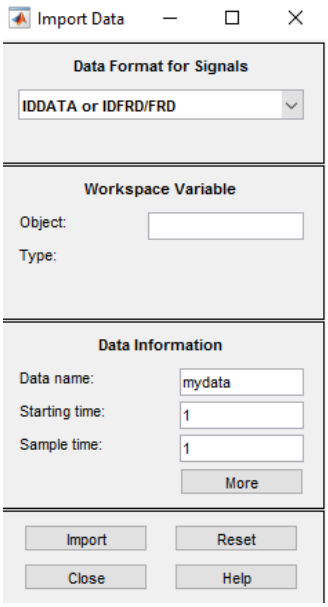
```
cuadrada=iddata(cuadrada_d,cuadrada_v,0.02);
T_cuadrada=getTrend(cuadrada);
T_cuadrada.OutputOffset = 37.46774;
cuadrada_data = detrend(cuadrada,T_cuadrada);
```

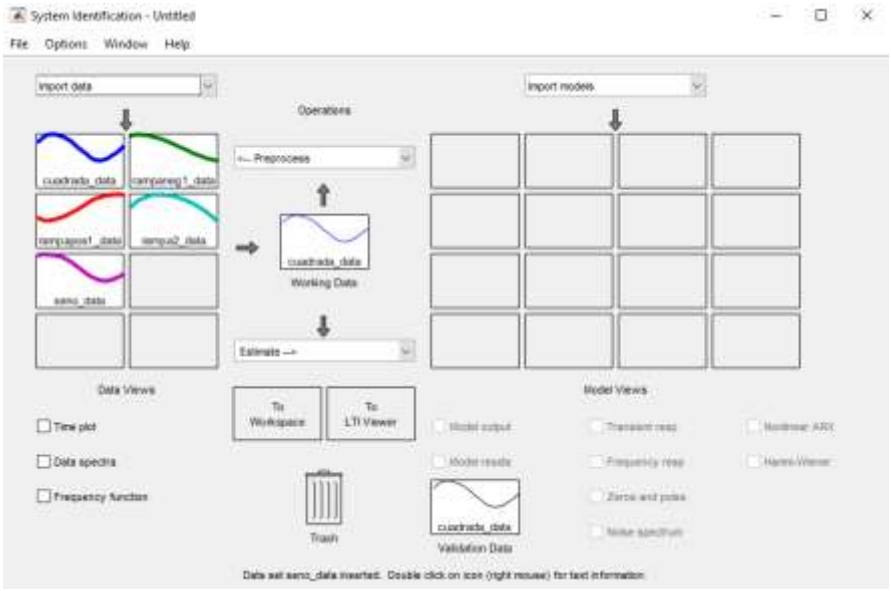
Una vez tenemos todas las señales en objetos *iddata* siendo muy importante que el tiempo de muestreo sea el mismo (0.02 segundos en mi caso) y asumiéndolos correctos debido al escaso ruido registrado, abrimos *ident* desde la ventana de comandos.



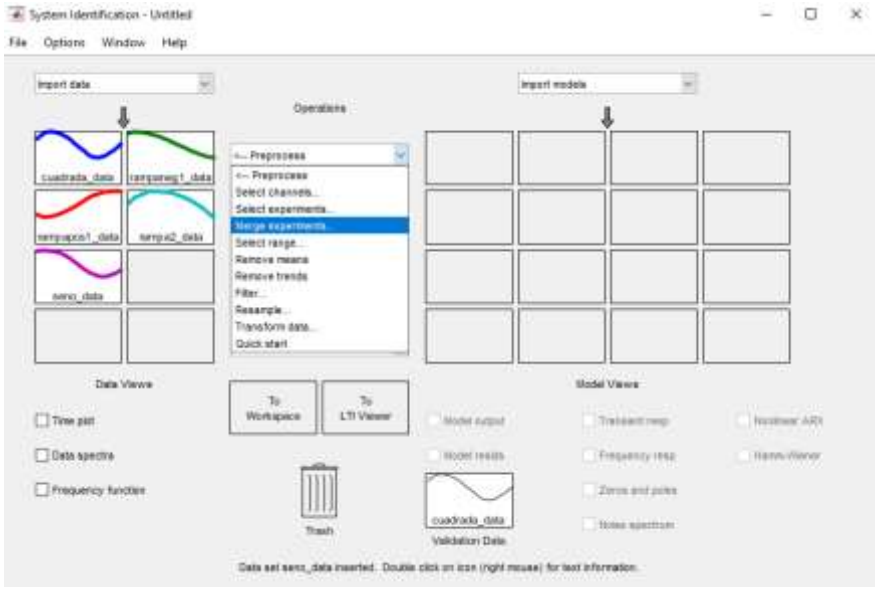

En el desplegable *Import data* seleccionamos *Data Object*.

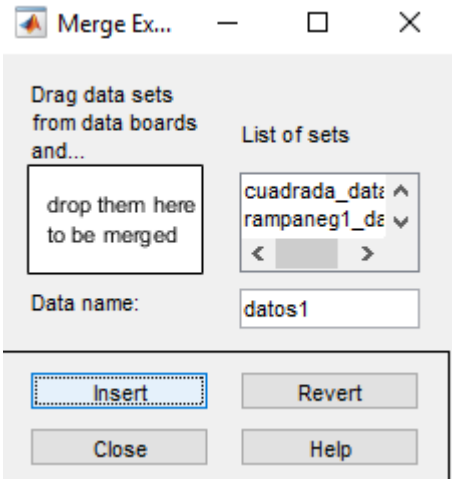
A continuación, vamos introduciendo el nombre de los objetos *iddata* anteriormente generados, comprobamos que el *Sample time* es el correcto y coincide en todos los casos y, los vamos importando uno a uno a *Data Views*.



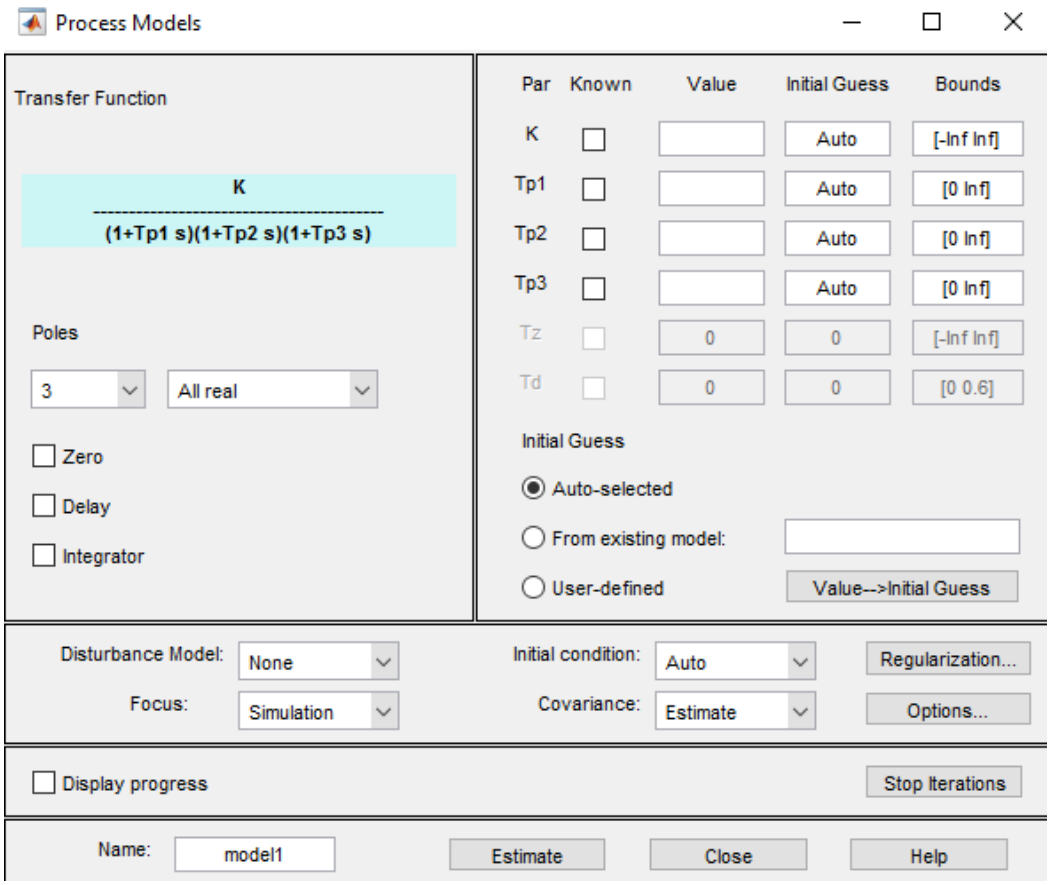

El siguiente paso es combinar todos los datos. Para ello selecciona *Merge experiments* en el desplegable *Preprocess* y vamos arrastrando uno a uno los datos a la nueva ventana. Escribimos el nombre para la combinación de experimentos y presionamos *Insert*.



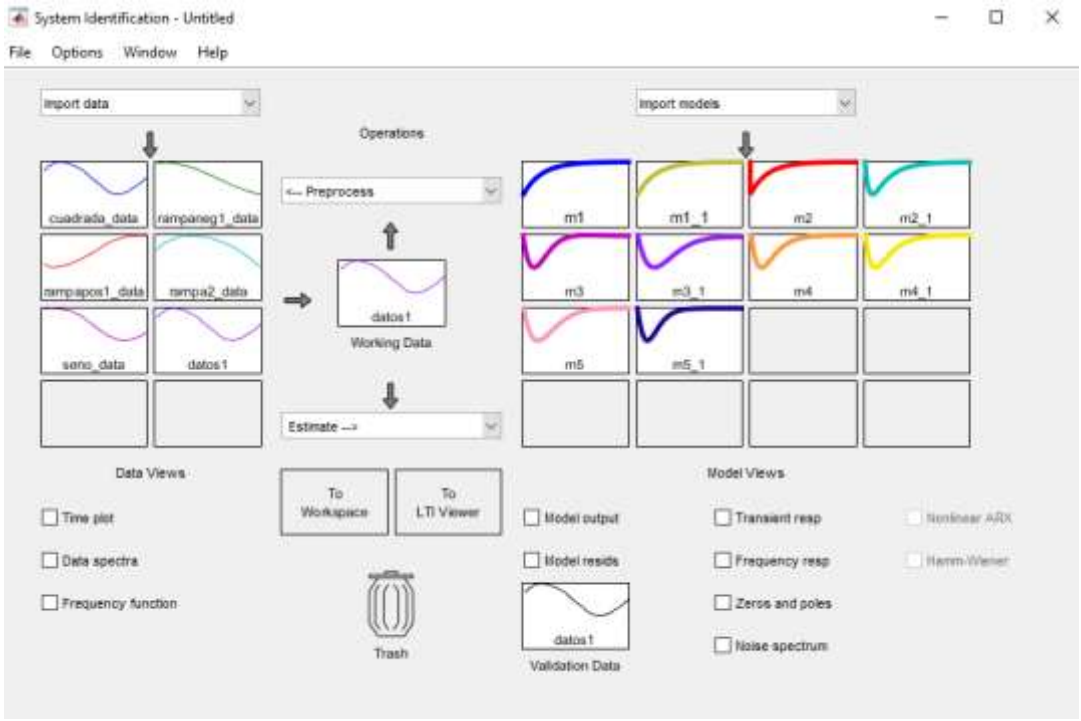

Se generará un nuevo objeto en *Data Views* combinación de todos los anteriores. Continuamos arrastrando el nuevo objeto al *Working Data* y seleccionando *Process Models* en el desplegable *Estimate*.

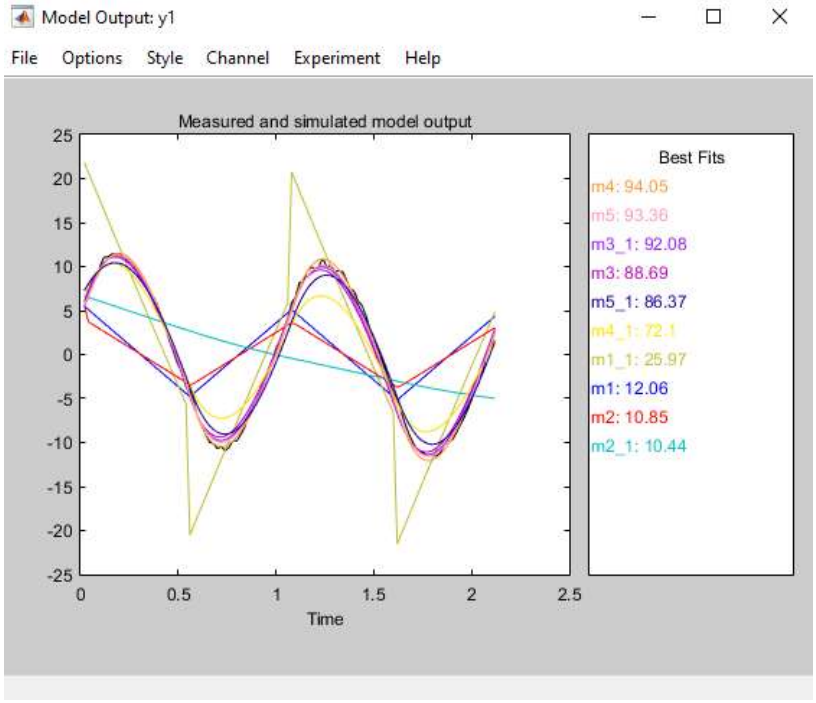

En este punto se nos abrirá la siguiente ventana:



Ahí podemos elegir qué tipo de función de transferencia tendrá nuestra planta. Para ello elegimos el número de polos, si son reales o imaginarios, si debemos añadir un cero o un retardo puro en la respuesta entre otras opciones. Pulsaremos después sobre *Estimate* para enviar el modelo a la sección *Model Views* de la interfaz de *Ident*.

La idea es ir probando diferentes configuraciones para dar con la adecuada, generando varios modelos. Así lo hacemos y, haciendo click en la casilla *model output* con todos los modelos seleccionados, Matlab nos indica un porcentaje de ajuste con el modelo real.






El modelo que más se ajusta, *m4*, corresponde a un modelo de 3 polos reales, sin cero y sin retardo. Tomaremos, entonces, este modelo y lo exportaremos al *Workspace* gracias al bloque destinado a ello en la interfaz.

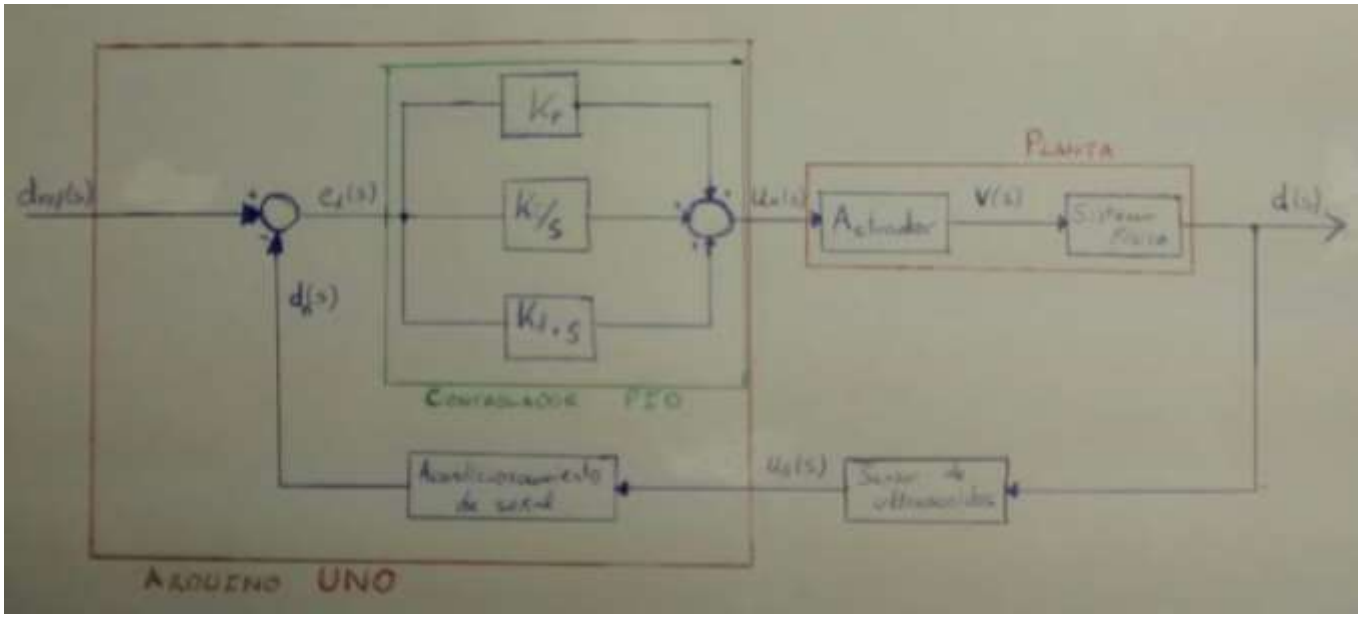
Ya en la ventana de comandos procedemos a sacar la función de transferencia del modelo estimado de la planta:

```
>> modelo = tf(m4)

modelo =

  From input "u1" to output "y1":
            -0.2921
  ---------------------------------------
  0.006489 s^3 + 0.2402 s^2 + 0.9516 s + 1

Name: m4
Continuous-time transfer function.
```

# Diagrama de bloques

Una vez obtenido un modelo válido de la planta, nos podemos poner a pensar en el regulador. Para ello, lo primero es realizar el diagrama de bloques, diferenciando la planta de lo que se programará en el microcontrolador.

Por lo tanto, el diagrama de bloques quedará de la siguiente manera:




La señal de error será la diferencia entre la distancia de referencia y la distancia real y será la señal que entre al controlador PID, el cual sigue la siguiente fórmula:

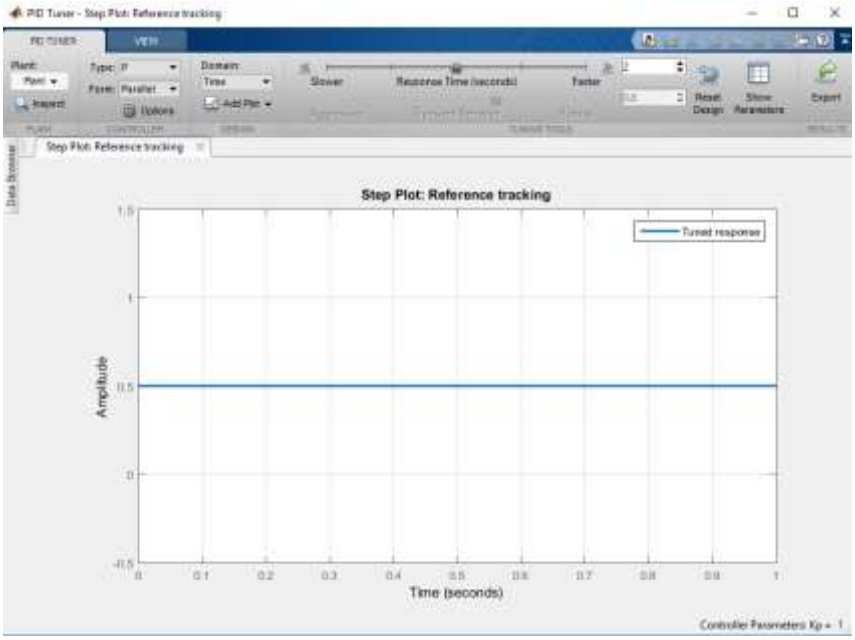
$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{d}{dt} e(t)$$

La salida en forma de tensión se mete a la planta a través del actuador, que en este caso es el motor de corriente continua. Dicha tensión es proporcional a la velocidad angular de las ruedas en una constante que desconocemos. El movimiento rotacional de las ruedas traslada la velocidad a la superficie, desplazándose en el espacio con diferentes pérdidas por deslizamientos entre las ruedas y la superficie. La salida de la planta será la distancia real del coche al obstáculo.

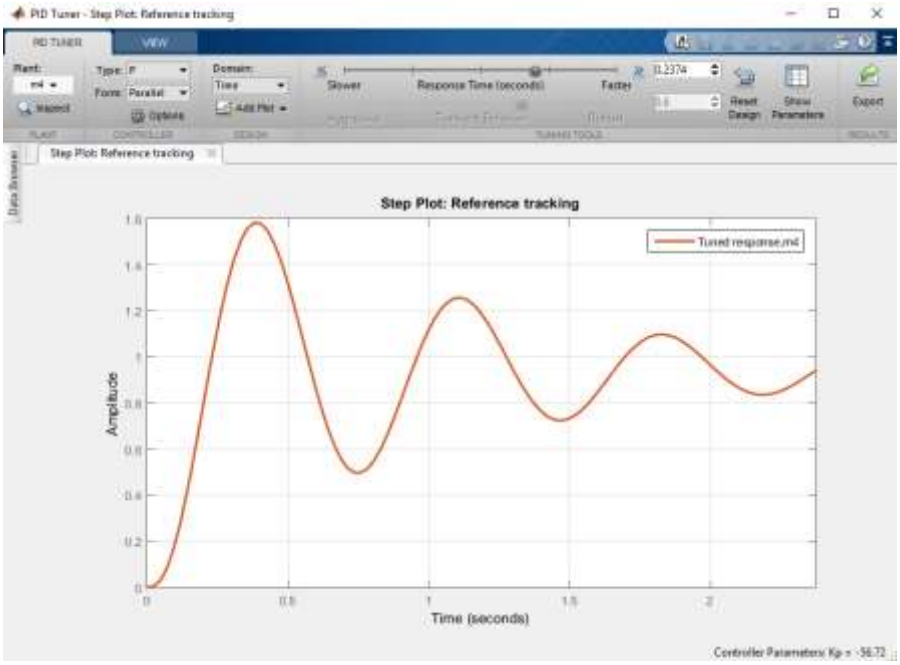
La distancia real es realimentada por el sensor ultrasonidos que la traduce a un tren de pulsos. Dicho tren de pulsos debe ser interpretado, ya en el microcontrolador Arduino, para sacar la distancia y asignarla a una variable justo antes de volver a ser comparada. La distancia de referencia será introducida por comunicación serial entre el microcontrolador y un ordenador, aunque haya una predefinida en la misma programación del micro.

# Diseño del controlador

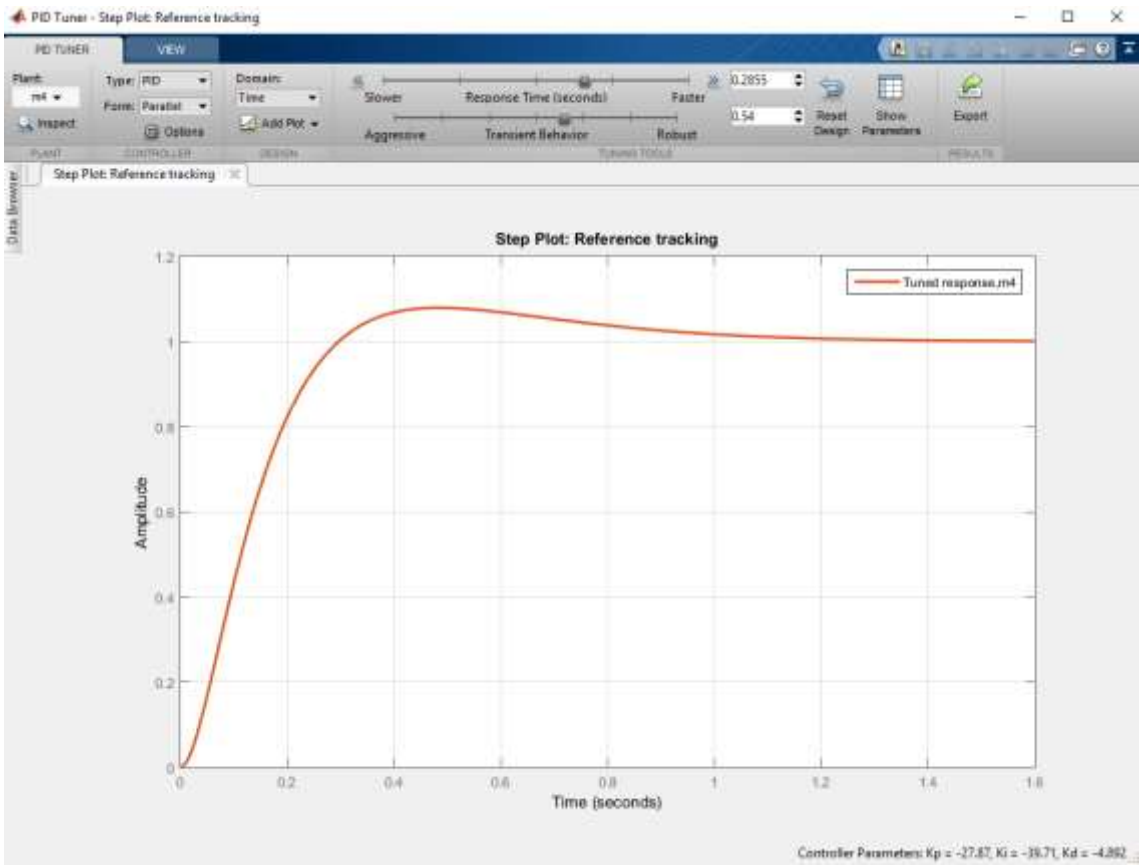
El siguiente paso a llevar a cabo es el cálculo de las constantes del controlador PID. Sacaremos las constantes más adecuadas a nuestra planta y a nuestros objetivos con la herramienta *pidTuner* de Matlab.




En primer lugar debemos importar el modelo de la planta anteriormente calculado. Para ello usaremos el desplegable *Plant*. Una vez cargado, se nos empezará a mostrar la estimación de respuesta al escalón de la planta con el controlador.



Ya tan sólo nos quedaría seleccionar el tipo de controlador gracias a los desplegables, PID en nuestro caso, y ajustar de manera interactiva la respuesta usando los dos sliders que regulan el tiempo de respuesta y el comportamiento en el estado transitorio.




En nuestro caso optaremos por una respuesta relativamente rápida, ya que los motores necesitan cierto nivel de tensión para vencer el par de arranque desde parado y esto proporcionará agilidad al prototipo y, por otro lado, permitiremos cierta sobreoscilación para tratar de amortiguar los deslizamientos de las ruedas con la superficie. Haciendo click en *Show Parameters* podremos observar las constantes del controlador y datos numéricos de la respuesta, tales como el tiempo de subida, el tiempo de establecimiento, la sobreoscilación o la estabilidad.

Controller Parameters

| | Tuned |
|---|---|
| Kp | -27.8749 |
| Ki | -39.7089 |
| Kd | -4.8919 |
| Tf | |

Performance and Robustness

| | Tuned |
|---|---|
| Rise time | 0.196 seconds |
| Settling time | 0.956 seconds |
| Overshoot | 7.86 % |
| Peak | 1.08 |
| Gain margin | Inf dB @ Inf rad/s |
| Phase margin | 68.1 deg @ 7.01 rad/s |
| Closed-loop stability | Stable |

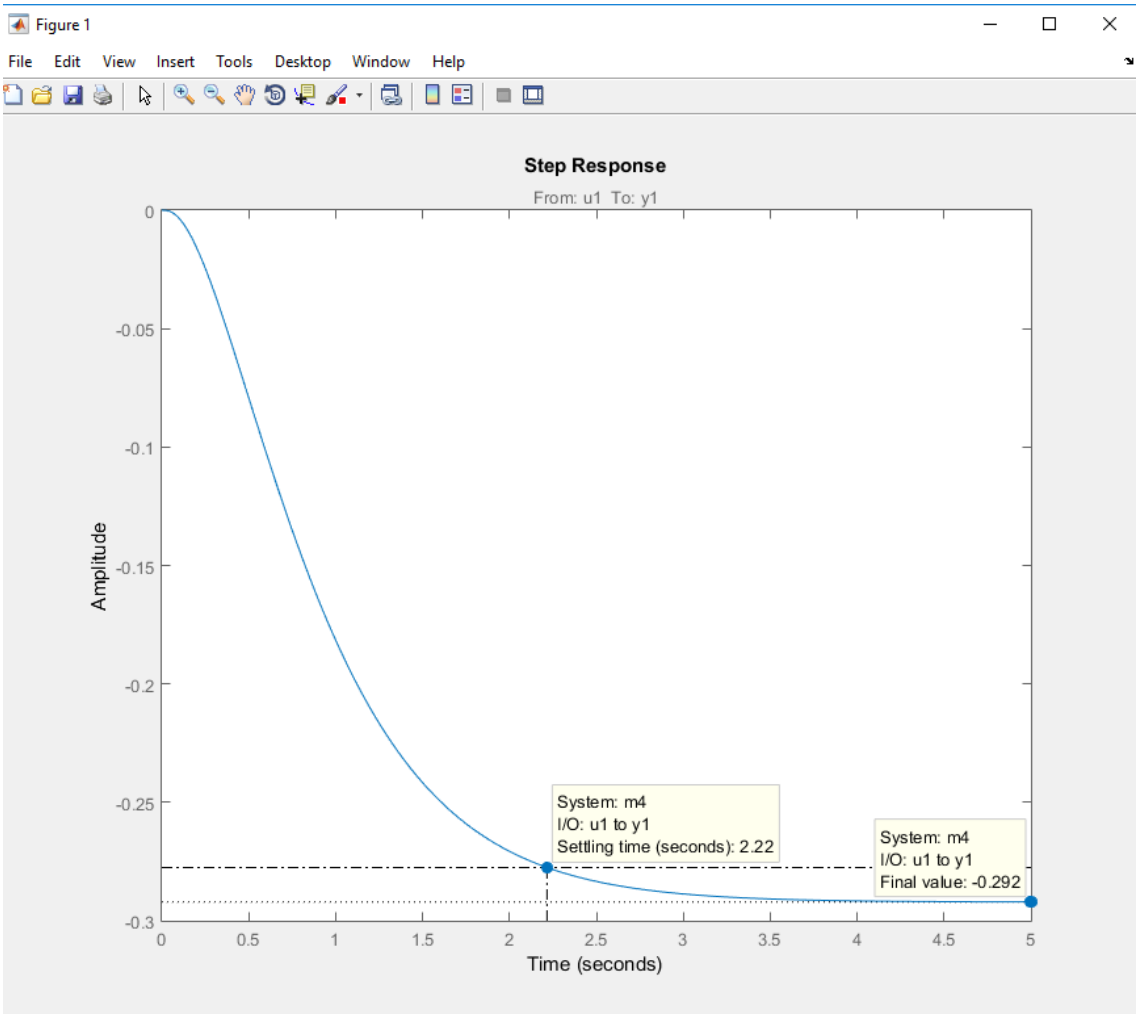
Una vez obtenemos la respuesta que deseamos para nuestro prototipo, pulsamos *Export*, añadimos un nombre y llevaríamos el controlador al *Workspace* donde podremos usarlo en otras operaciones.

# Simulación en Matlab

A continuación pasamos a la simulación del modelo, con y sin el controlador.

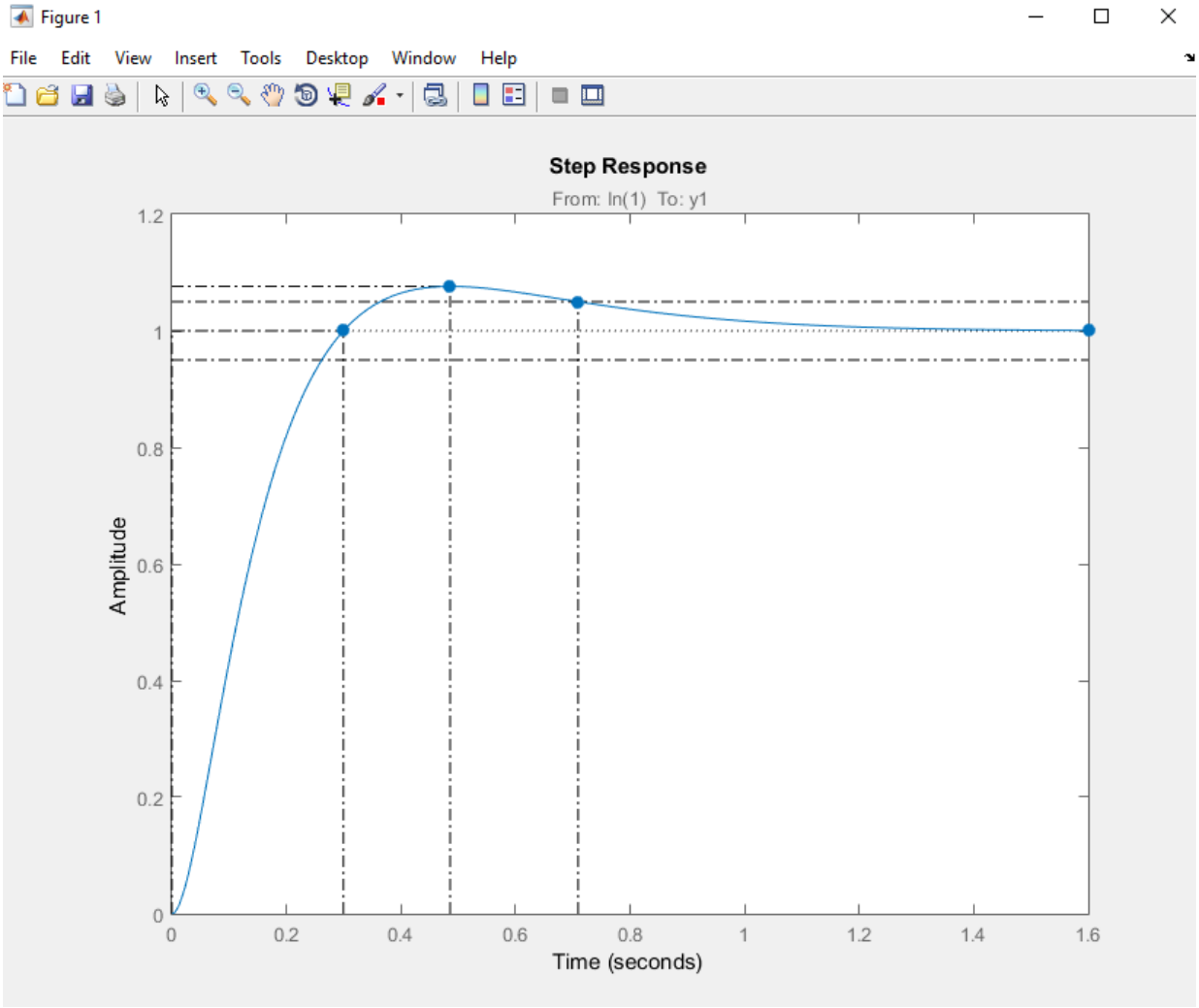
**Respuesta al escalón del modelo de la planta:**

>> step(m4)



**Respuesta al escalón del modelo con el controlador:**

```
modelo=tf(m4);
controlador=tf(C4);
modelo_c=feedback(modelo*controlador,1);
step(modelo_c)
```

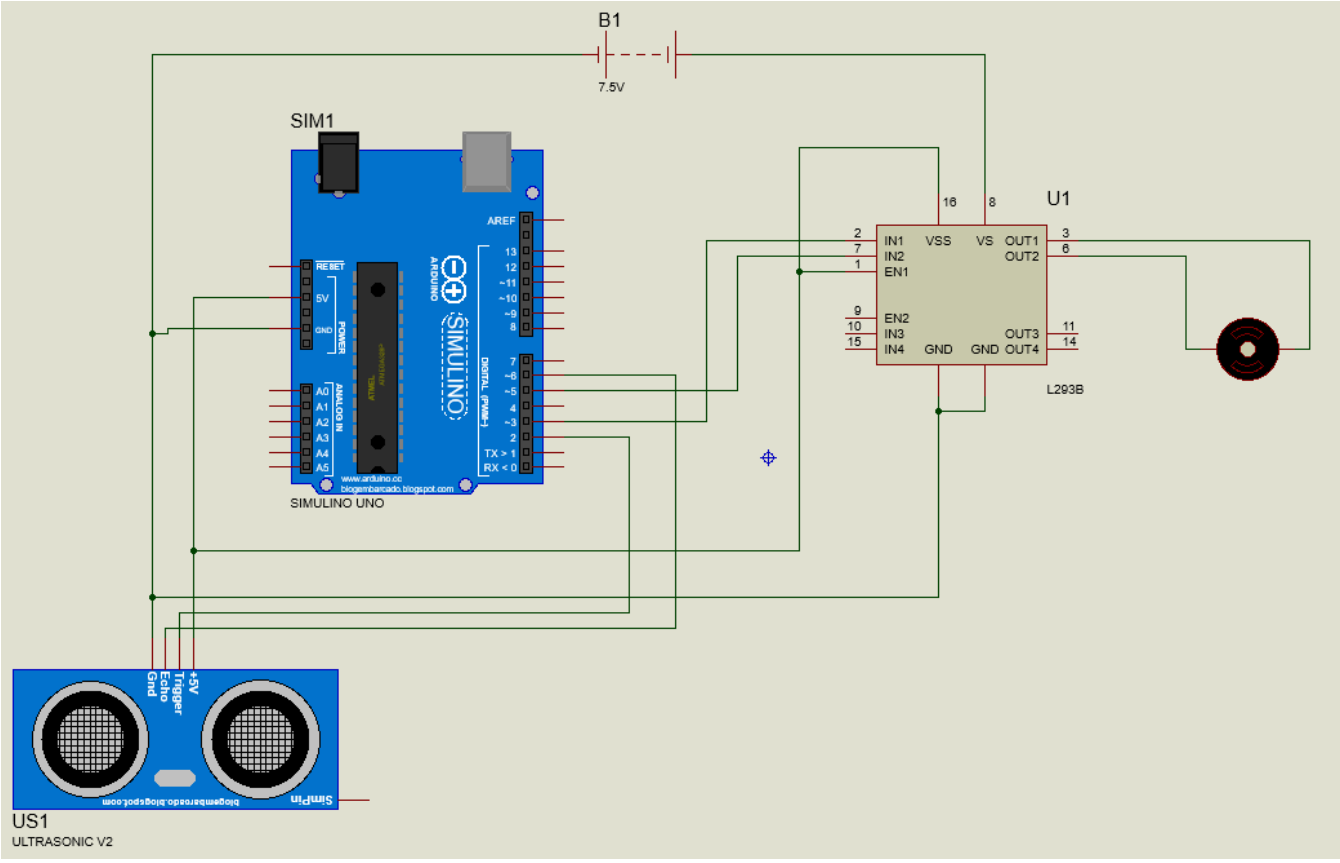


# Montaje

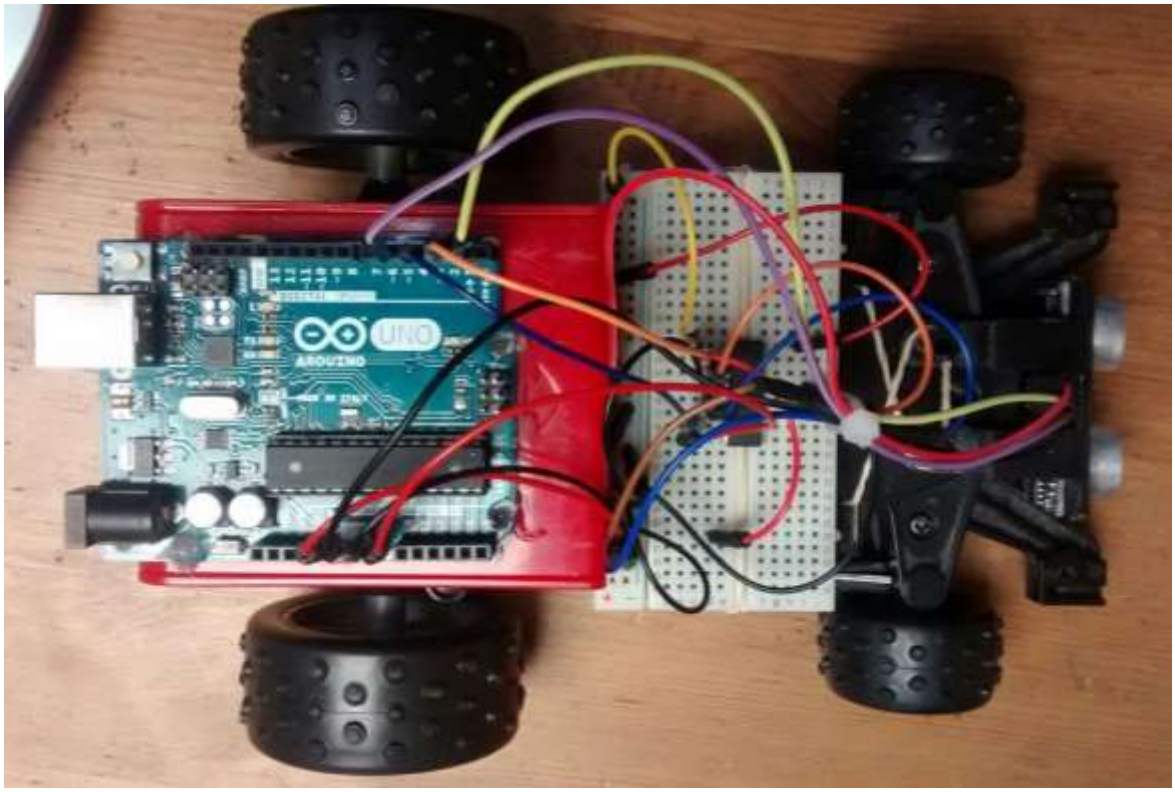
El prototipo es el chasis de un antiguo coche RC al que se le ha retirado toda la parte electrónica, dejando tan solo el motor y los elementos mecánicos. Además, se ha fijado la dirección ya que nos interesa que sólo se mueva en una dirección.

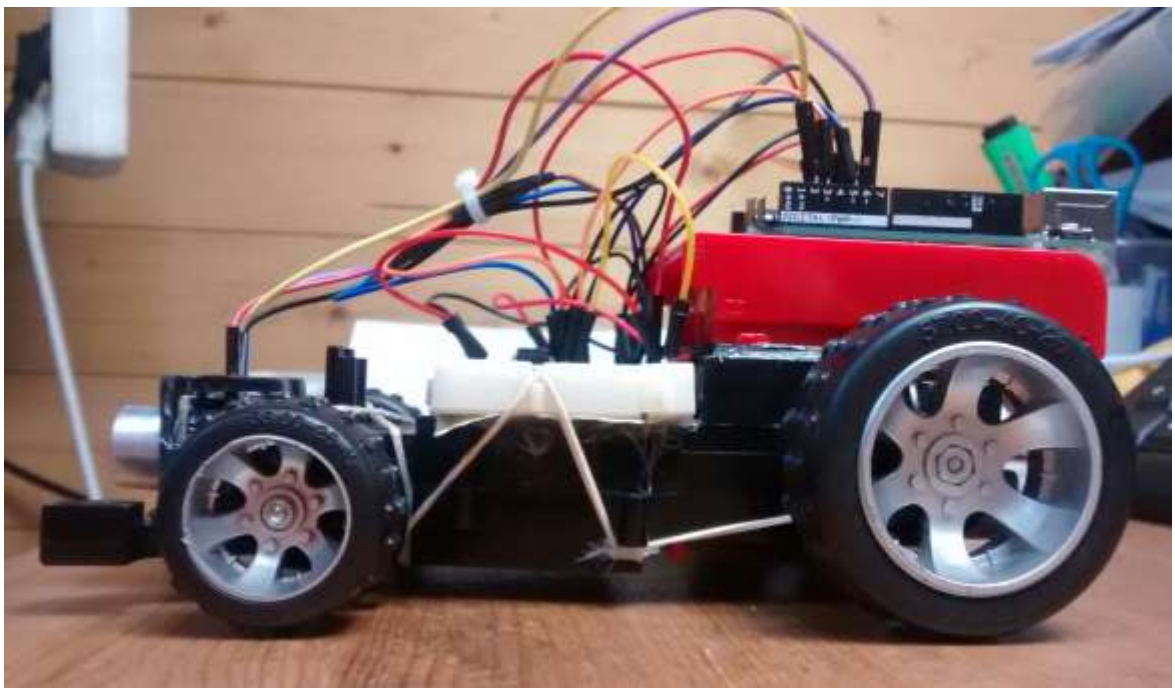
Se ha fabricado una pieza para sujetar el sensor de ultrasonidos en la posición correcta y la fuente externa de alimentación (5 pilas AA), el microcontrolador y una pequeña protoboard con las conexiones y el puente H se han fijado sobre el chasis del coche.

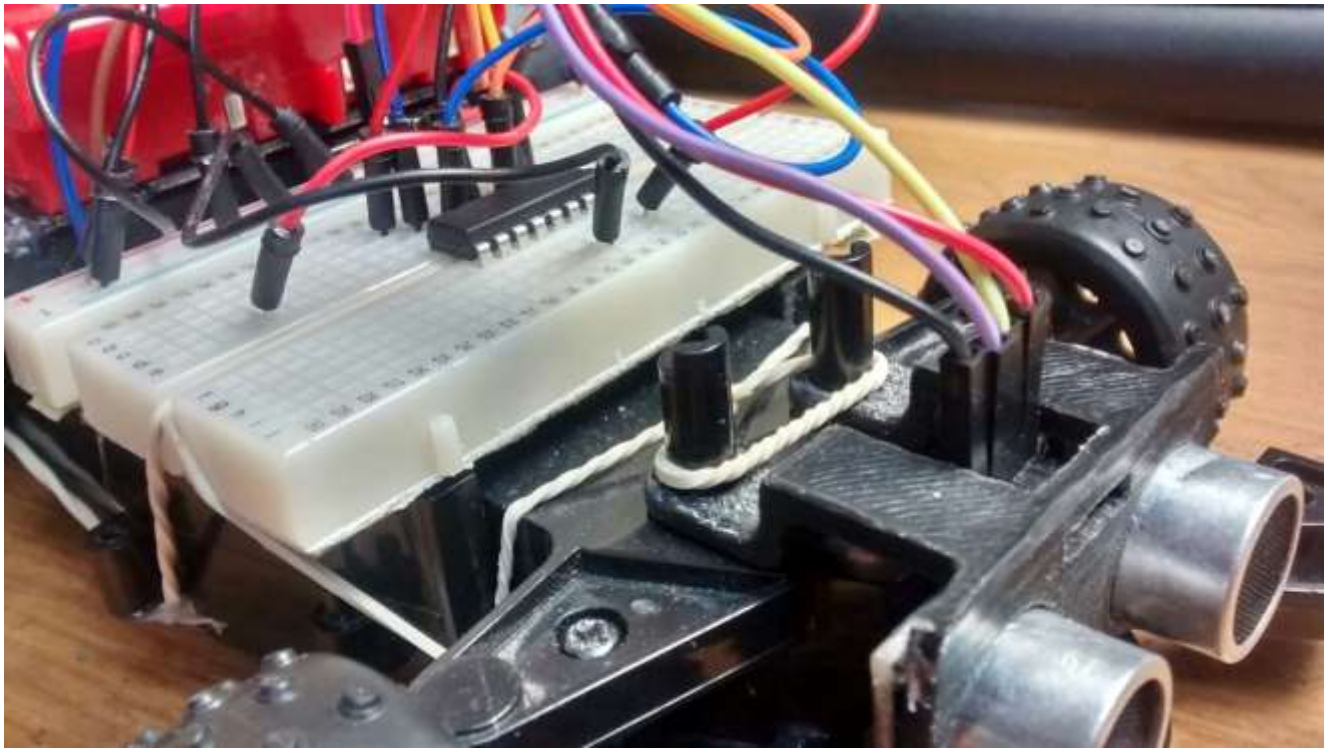
El diagrama de conexiones, realizado en Proteus, es el siguiente:

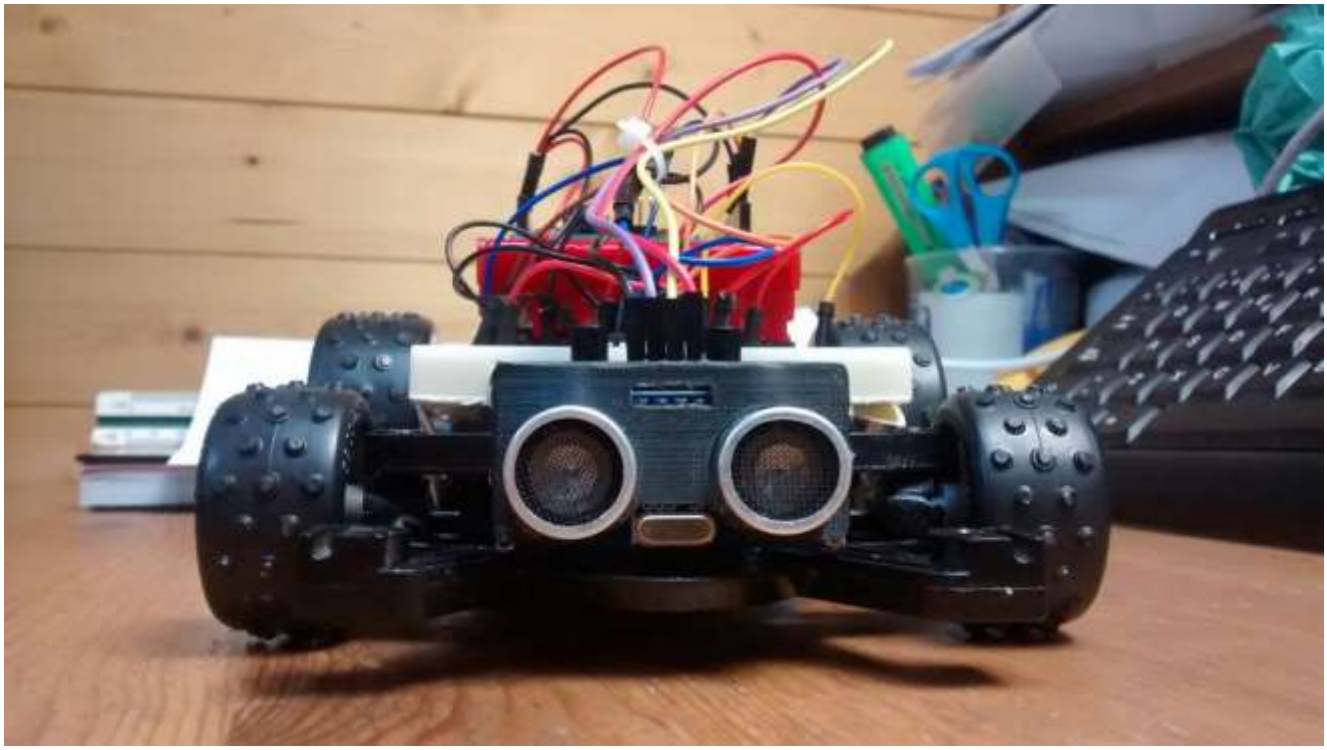


**Fotos tras el montaje:**









# Código Arduino

A continuación copio el código del programa metido al Arduino UNO. Usaremos la librería PID_v1 para simplificar y optimizar el controlador.

```
#include <PID_v1.h>

#define trigPin 2
#define echoPin 6
#define fwdPin 3
#define bwdPin 5

int output;

char cadena[4];    //Creamos un array que almacenará los caracteres que escribiremos en la consola del PC
byte posicion=0;  //Variable para cambiar la posición de los caracteres del array

double distancia, duracion, tension;    //Variables para el sensor de US y el controlador

double ref=30;                          //Distancia de referencia inicial

double Kp=27.65280482674217, Kd=4.887091689039301, Ki=39.11721241597037;//Constantes del PID

PID Control(&distancia, &tension, &ref, Kp, Ki, Kd,REVERSE);    //Declaramos el PID llamado Control

void setup() {
  Serial.begin(9600);                   //Comunicacion serial para enviar y recibir datos a traves del PC

  pinMode (trigPin,OUTPUT);             //Configuración de los pines
  pinMode (echoPin,INPUT);
  pinMode (fwdPin,OUTPUT);
  pinMode (bwdPin,OUTPUT);

  Control.SetMode(AUTOMATIC);           //Encendemos el PID
  Control.SetOutputLimits(-255, 255);         //Limitamos los valores de salida del PID
}

void loop() {

  if(Serial.available())              //Si hay datos dentro del buffer
  {
    memset(cadena, 0,sizeof(cadena));   // borra el contenido de "cadena" desde la posición 0 hasta el final

    while(Serial.available()>0)         //Mientras haya datos en el buffer
    {
      delay(5);                         //Delay para mejorar la recepción de datos
      cadena[posicion]=Serial.read();   //Lee los caracteres de "cadena" posicion a posicion
      posicion++;
    }

    ref=atoi(cadena);       //Convertimos la cadena de caracteres en enteros y se lo asignamos a la referencia
    posicion=0;             //Ponemos la posicion a 0
  }
```

```
 digitalWrite (trigPin,HIGH);          //Actualizacion de la distancia
 delayMicroseconds (1000);
 digitalWrite (trigPin,LOW);
 duracion = pulseIn(echoPin,HIGH);
 distancia=(duracion/2) / 29.1;

 Control.Compute();                    //Actualizacion de la salida del PID

 Serial.print(tension);                //Impresion serial de la referencia, la tension y la distancia
 Serial.print("\t");
 Serial.print(ref);
 Serial.print("\t");
 Serial.println(distancia);
                                 //SALIDAS

 if (tension < 0)                      //Si tension es negativa
   {
    digitalWrite(fwdPin,LOW);          //El coche se mueve hacia atras
    output=abs(tension);               //Se adecua tension al valor PWM
    analogWrite(bwdPin,output);        //Se escribe el valor analogico
   }
 else if(tension > 0)                  //Si tension es positiva
   {
    digitalWrite(bwdPin,LOW);          //El coche se mueve hacia delante;
    analogWrite(fwdPin,tension);       //Se escribe el valor analogico
   }
 else                                  //Si tension es cero
   {
    digitalWrite(fwdPin,LOW);          //El coche no se mueve
    digitalWrite(bwdPin,LOW);
   }
}
```

Por último, destacar del código la funcionalidad de poder cambiar la referencia de distancia desde el monitor serial de Arduino.

# Observaciones

Pienso que este proyecto ha sido uno de los más útiles realizados durante el grado. No sólo porque te ayuda a comprender el funcionamiento de los sistemas de control, en particular de los controladores PID, sino porque te obliga a buscar mucha información acerca de los métodos que puedes llevar a cabo para la obtención del modelo de la planta en el caso de que no conozcas cada variable física de la misma, como me ocurrió a mí. Quizás esto último fue lo más complicado del trabajo.

El proyecto ayuda a conocer muchas funcionalidades de una de las facetas de una herramienta tan versátil como Matlab; herramienta que no comenzamos a tocar hasta el quinto cuatrimestre. Da la sensación de que he exprimido bien las herramientas *Ident* y *pidTuner* pero sé de sobra que aún hay mucho por detrás y no me extrañaría que haya cogido el camino largo en alguna ocasión.

Más allá de lo relacionado con el control, el proyecto te fuerza a usar datasheets de componentes tales como el L293B o el HC-SR04, conocer Arduino (que, aunque en mi caso ya lo había usado con frecuencia, siempre viene bien) y en mi proyecto he incluido tres pequeñas piezas modeladas en Autodesk Inventor e impresas en 3D. En CREA (Club de Robótica, Electrónica y Automática) solemos imprimir piezas usando Cura por petición de los socios así que es una tecnología que conozco y domino. Las piezas son una simple lámina que protege conexiones bajo el cajetín de pilas, el soporte para el sensor de ultrasonidos y una pequeña pieza que ha servido para reparar las sujeciones de la rueda delantera izquierda, que estaban dañadas.

Por último, comentar que me gustaría seguir desarrollado este proyecto. Mi intención es imprimir una carcasa, diseñar una pequeña PCB para retirar la protoboard e incluir una pantalla LCD. El objetivo es que, a través de CREA, el proyecto sirva de ejemplo de los diferentes tipos de controladores y sus respuestas para alumnos de nuevo ingreso o nuevos socios del club.