SISTEMAS DE CONTROL EN TIEMPO CONTINUO

PEDRO LUIS SOLARTE CORREA
RICARDO ADOLFO ASTAIZA GUERRERO
ALBEIRO METUS UNI

CONTROL PID DE PLANTA DE TEMPERATURA

A:
ING. HERMES FABIAN VARGAS ROSERO

UNIVERSIDAD DEL CAUCA
FACULTAD DE INGENIERÍA ELECTRÓNICA Y TELECOMUNICACIONES
INGENIERÍA EN AUTOMÁTICA INDUSTRIAL
POPAYÁN, CAUCA
2016

INTRODUCCIÓN

Un controlador PID (Proporcional Integrativo Derivativo) es un mecanismo de control sobre una realimentación de lazo cerrado, que es usado ampliamente en la industria para realización de control de sistemas. El PID es un sistema al que le ingresa un error calculado a partir de la salida deseada, menos la salida obtenida y su salida es utilizada como entrada en el sistema que se quiere controlar. El controlador intenta minimizar el error ajustando la entrada del sistema.

OBJETIVOS DE LA PRACTICA

La finalidad de la realización de esta práctica es poder recrear un recinto en el cuál se pueda tener un control total de la temperatura. El control de la temperatura se hará sensando la misma, y enviando ese registro a nuestro sistema de control para que tome decisiones con respecto a un *set point* de temperatura deseada.

CONCEPTOS A TENER EN CUENTA

- **Control PID**

  El controlador PID viene determinado por tres parámetros, los cuales son: el proporcional, el integral y el derivativo. Dependiendo de la modalidad del controlador alguno de estos valores puede ser 0. Por ejemplo, un controlador proporcional tendrá el integral y el derivativo igualados a 0 y un controlador PI solo el derivativo será 0.

  Cada uno de estos parámetros influye en mayor medida sobre alguna característica de la salida (tiempo de establecimiento, sobreoscilación, entre otras) pero también influye sobre las demás, por lo que por mucho que ajustemos no encontraríamos un PID que redujera el tiempo de establecimiento a 0, la sobreoscilación a 0, o que el error sea 0. sino que se

trata más de ajustarlo a un término medio cumpliendo las especificaciones requeridas.

- **Matlab**

  La plataforma de MATLAB está optimizada para resolver problemas de ingeniería y científicos. El lenguaje de MATLAB, basado en matrices, es la forma más natural del mundo para expresar las matemáticas computacionales. Los gráficos integrados facilitan la visualización de los datos y la obtención de información a partir de ellos. Una vasta librería de toolboxes preinstaladas le permiten empezar a trabajar inmediatamente con algoritmos esenciales para su dominio. El entorno de escritorio invita a experimentar, explorar y descubrir. Todas estas herramientas y prestaciones de MATLAB están probadas y diseñadas rigurosamente para trabajar juntas.

- **Simulink**

  Es un paquete de software para modelar, simular y analizar sistemas dinámicos, soporta sistemas lineales y no lineales, modelados en tiempo continuo, muestreados o en hibrido entre los dos. Los sistemas pueden ser también multifrecuencia, es decir, tienen diferentes partes que se muestrean o se actualizan con diferentes velocidades.

- **Arduino**

  Arduino es una plataforma de prototipos electrónica de código abierto basada en hardware y software que son flexibles y fáciles de usar. Arduino puede sentir el entorno mediante la recepción de entradas desde una variedad de sensores y puede afectar a su alrededor mediante el control de luces, motores y otros dispositivos.

- **Arduino Mega 2560**

  Arduino es una plataforma física computacional basada en una sencilla placa con entradas y salidas, analógicas y digitales, y en un entorno de desarrollo que implementa el lenguaje Processing/Wiring. El Arduino Mega está basado en el microcontrolador ATMega2560. Tiene 54 pines de entradas/salidas digitales (14 de las cuales pueden ser utilizadas como salidas PWM), 16 entradas análogas, 4 UARTs (puertos serial por hardware), cristal oscilador de 16 Mhz, conexión USB, jack de alimentación, conector ICSP y botón de reset.

- **Sensor de temperatura LM35**

  El LM35 es un sensor de temperatura con una precisión calibrada de 1 ºC. Su rango de medición abarca desde -55 °C hasta 150 °C. La salida es lineal y cada grado Celsius equivale a 10 mV, por lo tanto:

  150 ºC = 1500 mV

  -55 ºC = -550 mV

  Opera de 4v a 30v.

- **Optoacoplador**

  Un Optoacoplador es un circuito integrado muy básico compuesto generalmente por un diodo LED y un fototransistor unidos de tal forma que cuando una señal eléctrica circula a través del LED haciendo que brille, la luz que este emite es recibida por la base del fototransistor, que empieza a actuar en modo saturación

  Se puede utilizar este dispositivo a modo de interfaz entre dos circuitos, de tal forma que quedarían unidos ópticamente, lo que a efectos de protección del circuito, se traduce en colocar una resistencia de un valor muy alto, lo que lo hace especialmente útil para proteger contra los picos de tensión.

- **Triac**

  El triac es un dispositivo semiconductor de tres terminales que se usa para controlar el flujo de corriente promedio a una carga, con la particularidad de que conduce en ambos sentidos y puede ser bloqueado por inversión de la tensión o al disminuir la corriente por debajo del valor de mantenimiento. El triac puede ser disparado independientemente de la polarización de puerta, es decir, mediante una corriente de puerta positiva o negativa.

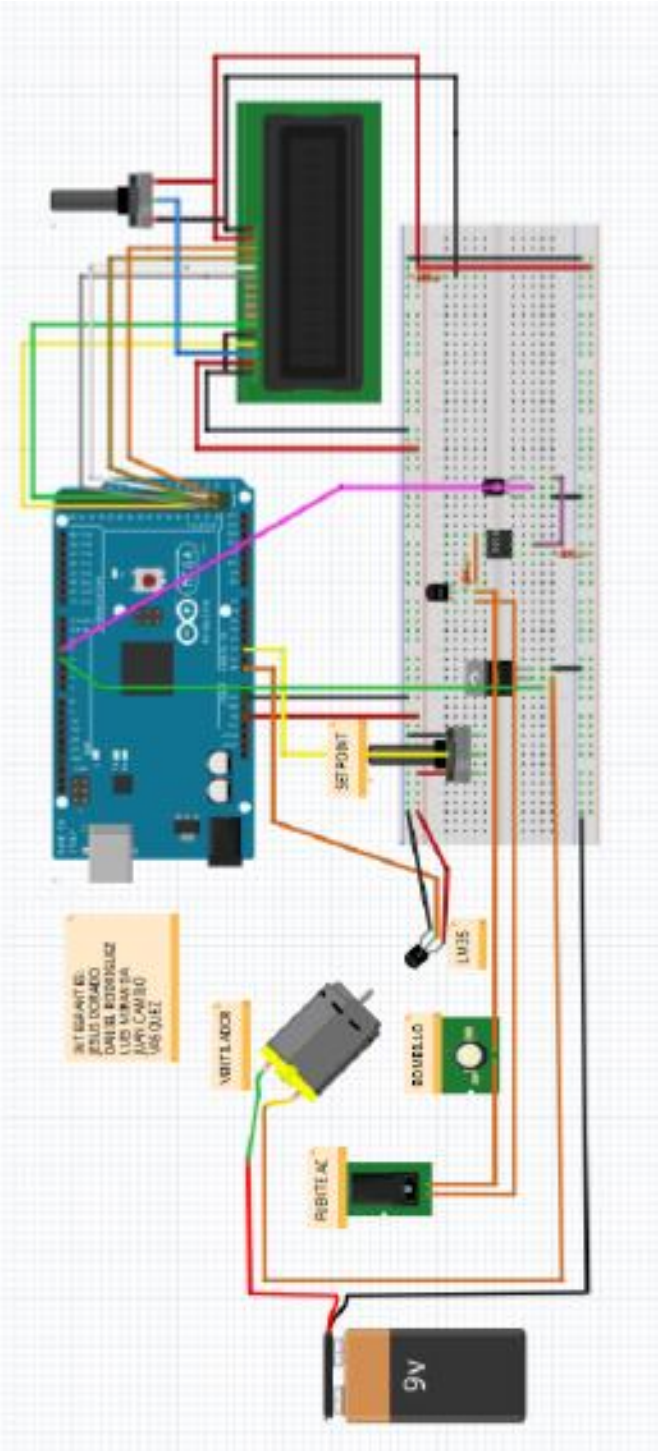
PRÁCTICA

Elementos utilizados

- Caja de cartón para simulación del recinto
- Arduino Mega 2560
- Sensor de temperatura LM35
- Fuente de 12V
- Jumpers
- Pantalla LCD
- Resistencias de 20Ω, 220Ω, 1KΩ, 5KΩ, 10KΩ
- Triac BT136
- Optoacoplador MOC3022
- Transistor 2N2222A, TIP31C
- Protoboard
- Bombillo incandescente de 70W
- Motor DC 12V
- Plafón
- Matlab 2014ª y librerías de Simulink para Arduino
- Software Arduino

PROCEDIMIENTO

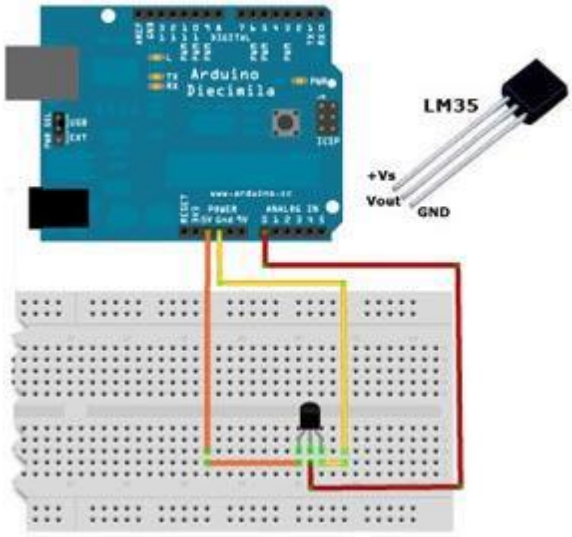
Realizamos el montaje de todos componentes que se mencionaron anteriormente en la protoboard, basados en el siguiente diagrama.



Posteriormente se adecuaron los componentes dentro de la caja, logrando recrear el recinto al cual se le quiere controlar la temperatura.

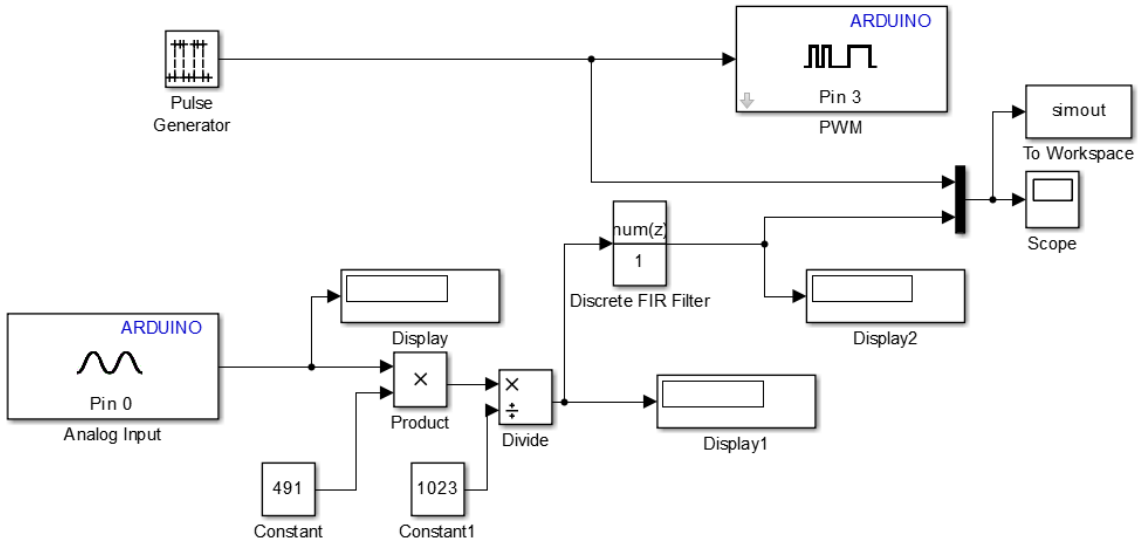
Ahora se muestra la manera de conectar los elementos como el sensor, el MOC y el Triac. Sensor de temperatura LM35: Para realizar la lectura de los datos de temperatura se utilizó el dispositivo LM35, el cual es un sensor de temperatura que opera con 10 mV / °C. Dicho sensor se implementó en la parte superior de la planta justo debajo de la bombilla incandescente.

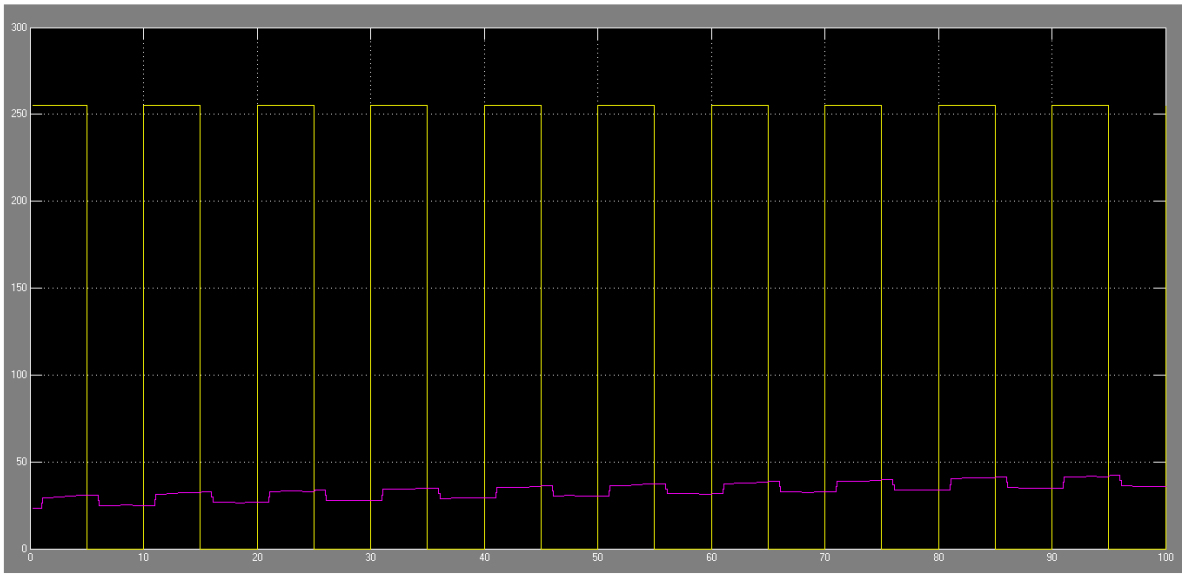
La conexión se muestra en la siguiente imagen



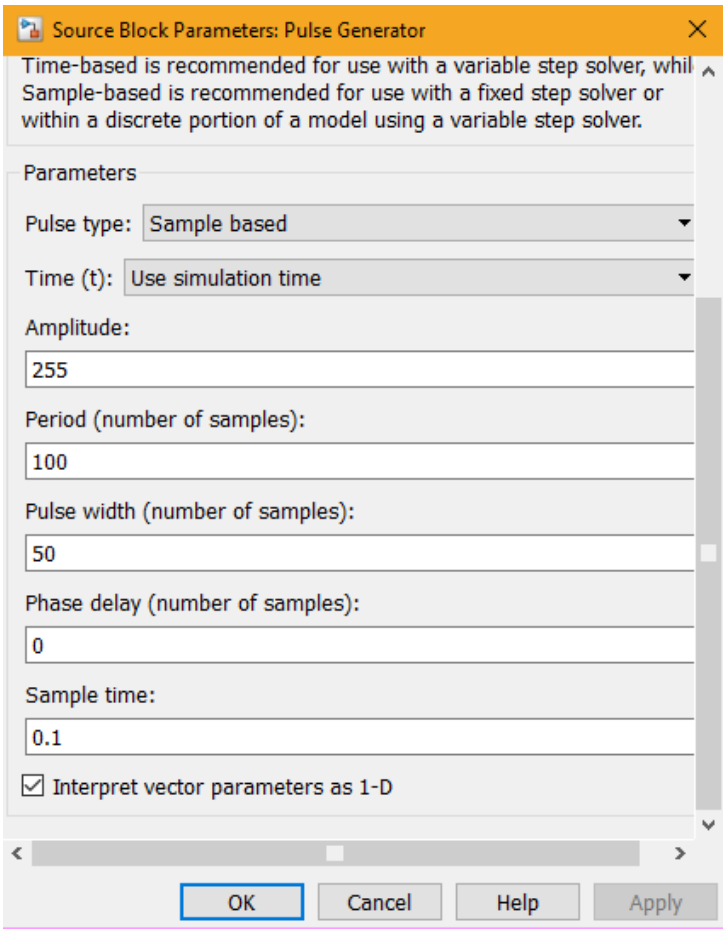
IDENTIFICACIÓN DE LA PLANTA

Primeramente, se generó un impulso al actuador, es decir el que va a generar el calor para elevar la temperatura que en este caso es la bombilla y como realimentación la lectura del sensor LM35 para saber el comportamiento de la planta ante una excitación.



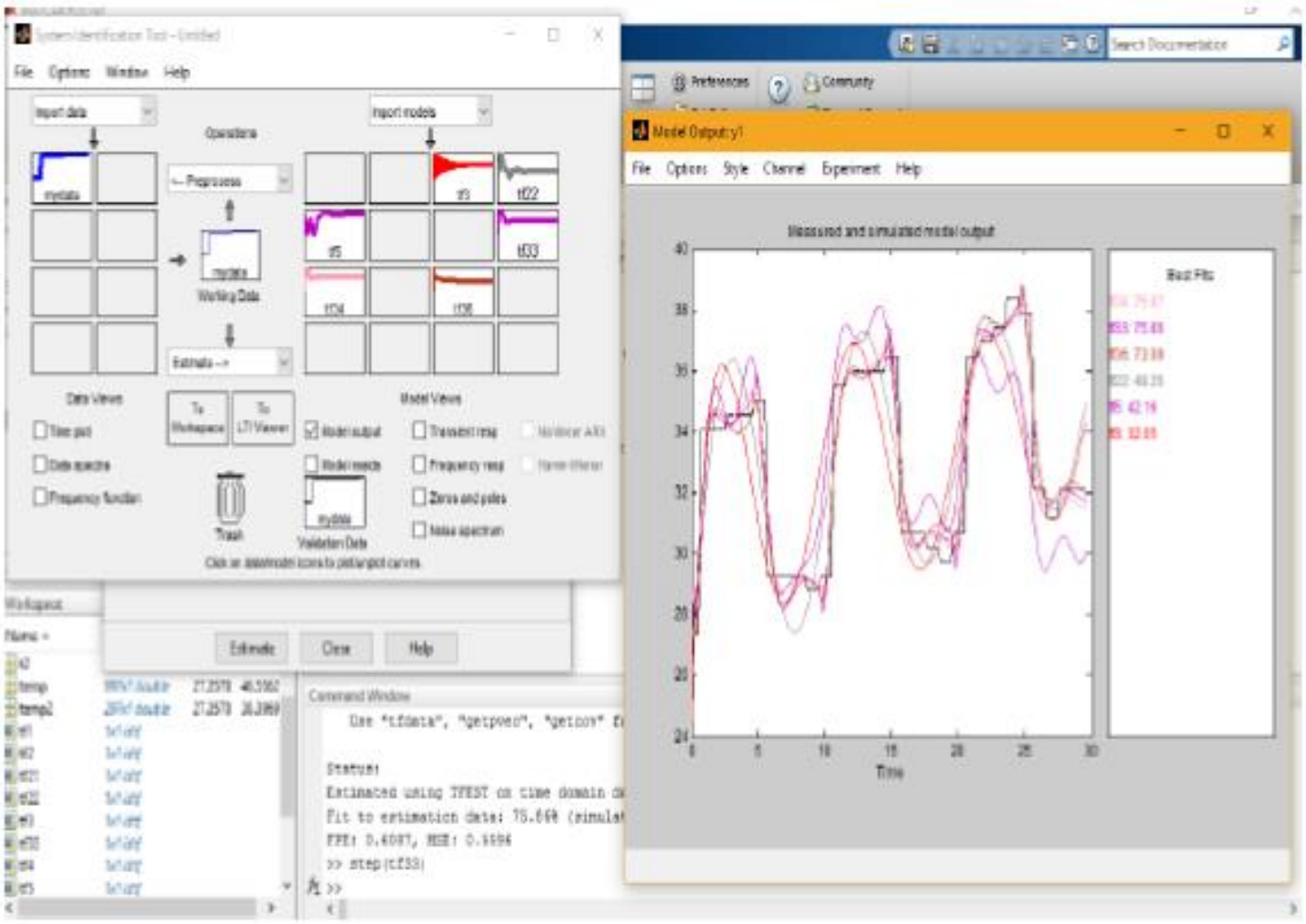


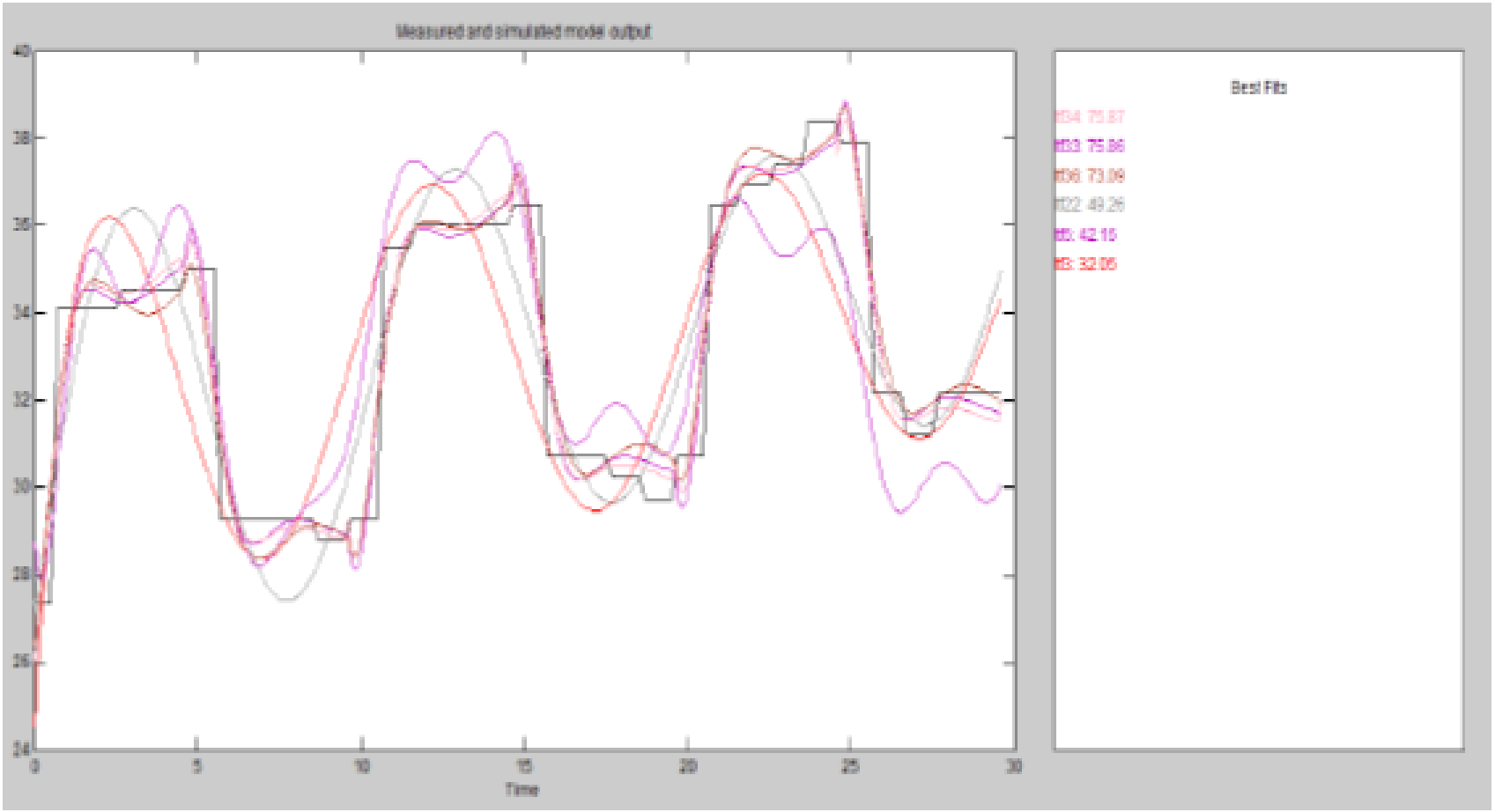
Se hicieron de 0 a 500 muestras en intervalos de 0.1, para obtener un modelo más exacto, con lo cual se escoge las mejores muestras para obtener los valores de I/O que se van a utilizar para hacer la identificación.



Luego se procede a crear dos vectores (para cada señal), con el rango de tiempo y los datos en ese rango para proporcionarlos a Matlab y crear una gráfica de I vs O. Este procedimiento se hace con la función **ident,** la cual estima una función de transferencia.

El paso siguiente es empezar a variar los polos y ceros de la función obtenida para obtener una exactitud como mínimo de 81%.


File Options Window Help
Import data
Operations
Preprocess
Working Data
Estimate -->
Data Views
Model Views
Model output
Transient resp
Frequency resp
Zeros and poles
Noise spectrum
Trash
Validation Data
Estimate
Close
Help
Model Output: y1
File Options Style Channel Experiment Help
Measured and simulated model output
Best Fits
Time
Command Window
Status:
Estimated using TFEST on time domain
Fit to estimation data: 75.869
FPE: 0.4087, MSE: 0.4094
>> step(tf33)


Measured and simulated model output
Best Fits
Time

Como se puede apreciar en la anterior imagen, en nuestro caso obtener la función de transferencia que cumpliera con el anterior requerimiento no fue posible, ya que el sistema presentaba muchas oscilaciones, pero después de hacer muchas pruebas se obtuvo una función de porcentaje cercano siendo de 75.87%. En la siguiente ilustración se puede observar la función de transferencia estimada.

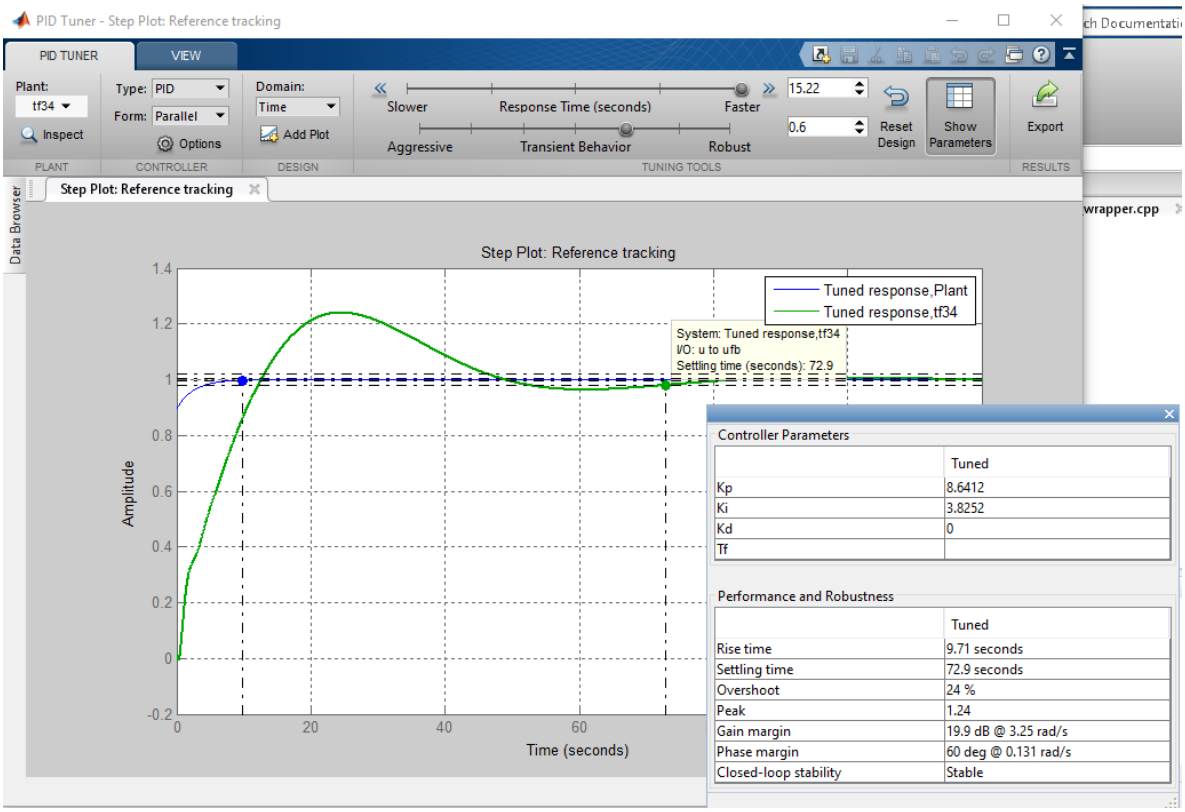
```
tf34 =

  From input "u1" to output "y1":
  0.002001 s^3 - 0.02435 s^2 + 0.1034 s + 0.01344
  -----------------------------------------------
        s^3 + 2.474 s^2 + 4.502 s + 0.03287
```

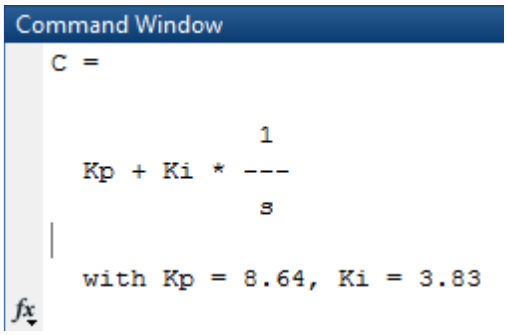
ESTIMACIÓN Y SINTONIZACIÓN DEL CONTROLADOR

Para este procedimiento se hace uso de la herramienta **pidtool,** para hacer la sintonización de la planta, tomando la función de transferencia estimada (en nuestro caso de 3 orden, que proporcionaba casi el 76% de exactitud) para modificar la respuesta del sistema siendo más rápido o más robusto, lo cual modifica las constantes del controlador que estima el **pidtool.**

Una vez obtenido un modelo que cumpla unas condiciones de rapidez, pero estable, se exporta la función de transferencia del controlador, con las variables del mismo. Como se puede observar en la siguiente imagen, la sintonización se hace para que el sistema cumpla unos requerimientos de rapidez y estabilidad, lo que se hace es hacer una compensación para que el sistema sea eficiente para estas dos características.

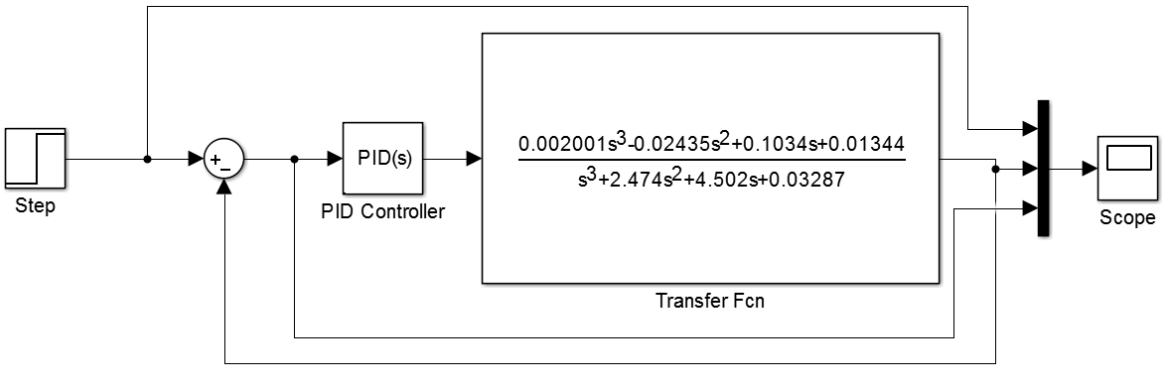


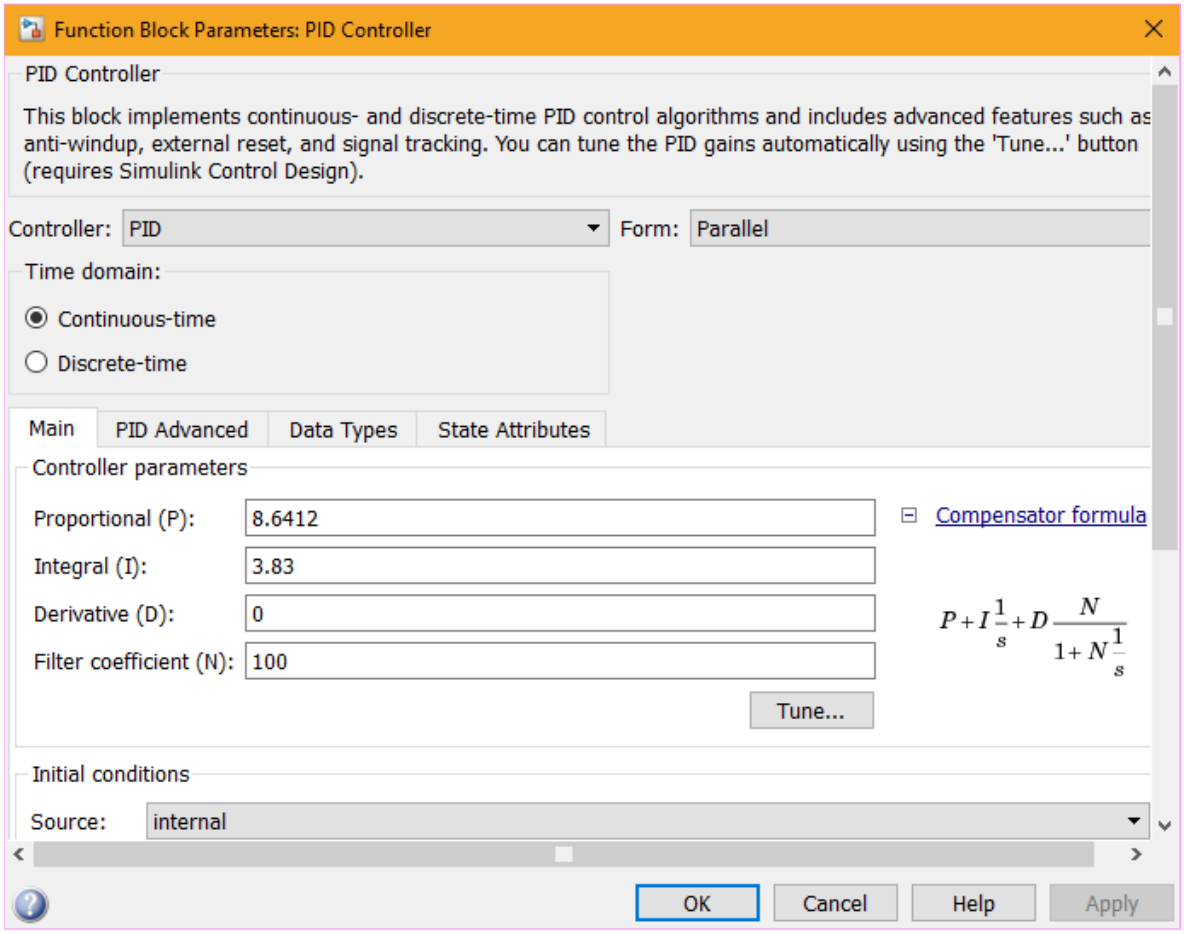
Una vez obtenidas las constantes del controlador y la función de transferencia de la planta, se procede a hacer las pruebas tanto en el código de Simulink como en la interfaz de Arduino.

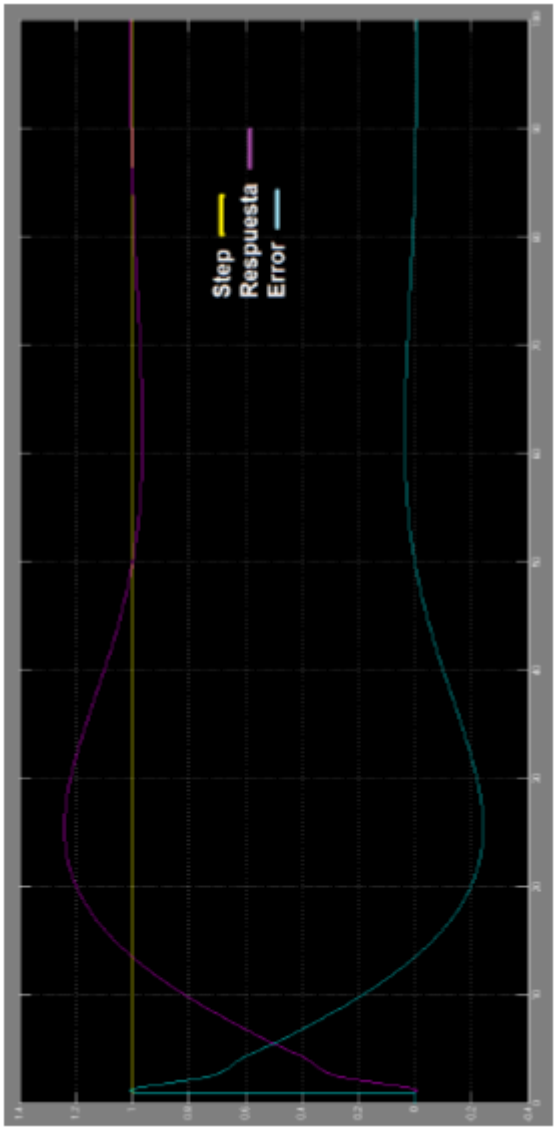


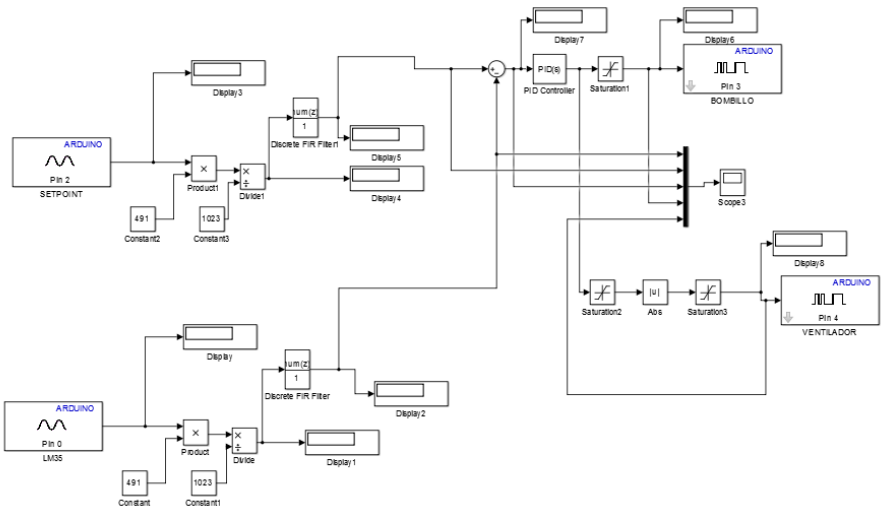
PRUEBAS DE FUNCIONAMIENTO EN SIMULINK Y ARDUINO

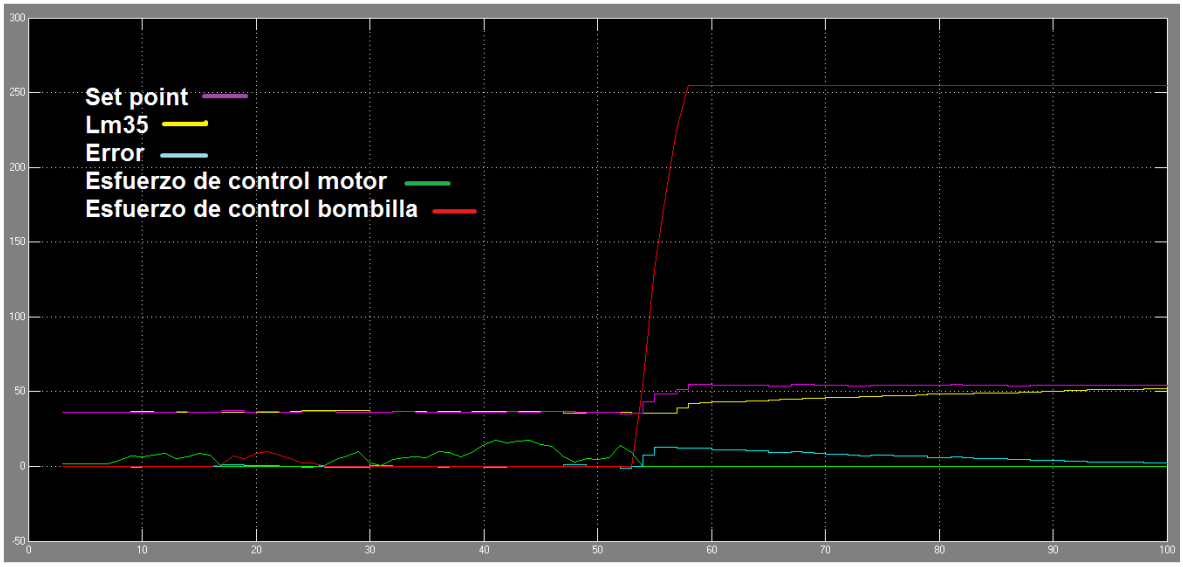
En el caso de Simulink se proporcionan las constantes obtenidas a la función/bloque PID y se procede a activar el sistema. En nuestro caso la planta se comporta de manera muy eficiente y para comprobar que sea el mismo comportamiento de manera teórica, se simula la planta con la función de transferencia estimada y las constantes del controlador para observar gráficamente la respuesta del sistema.



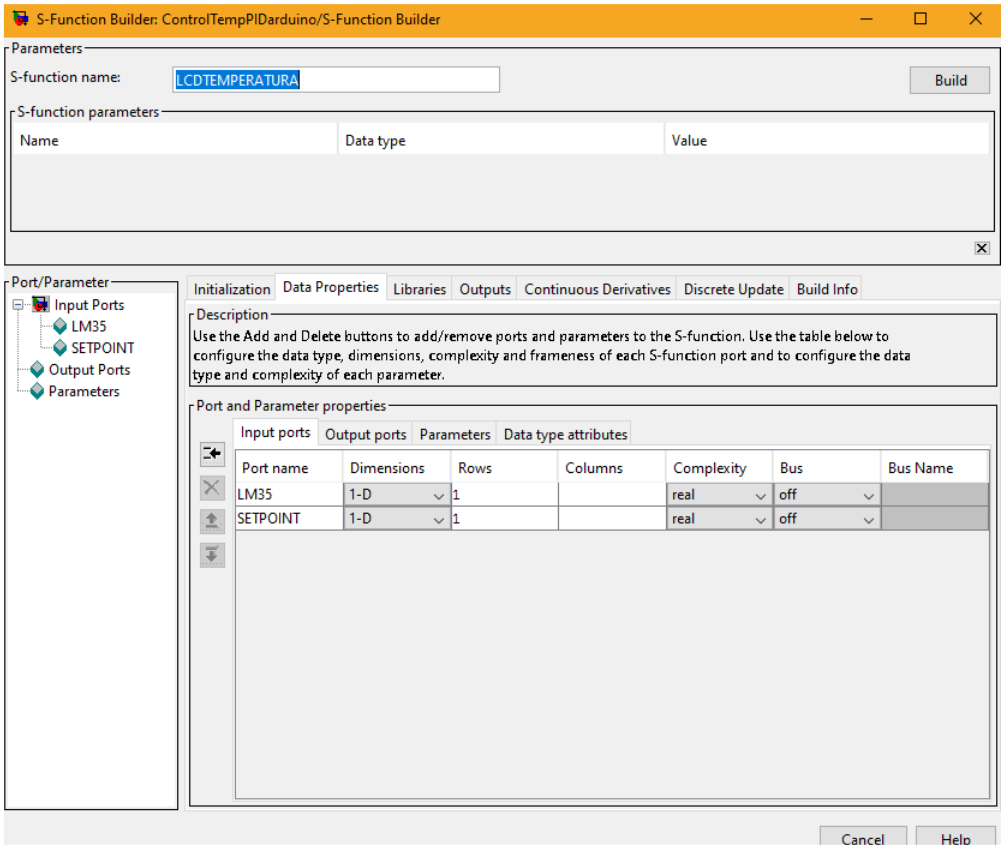


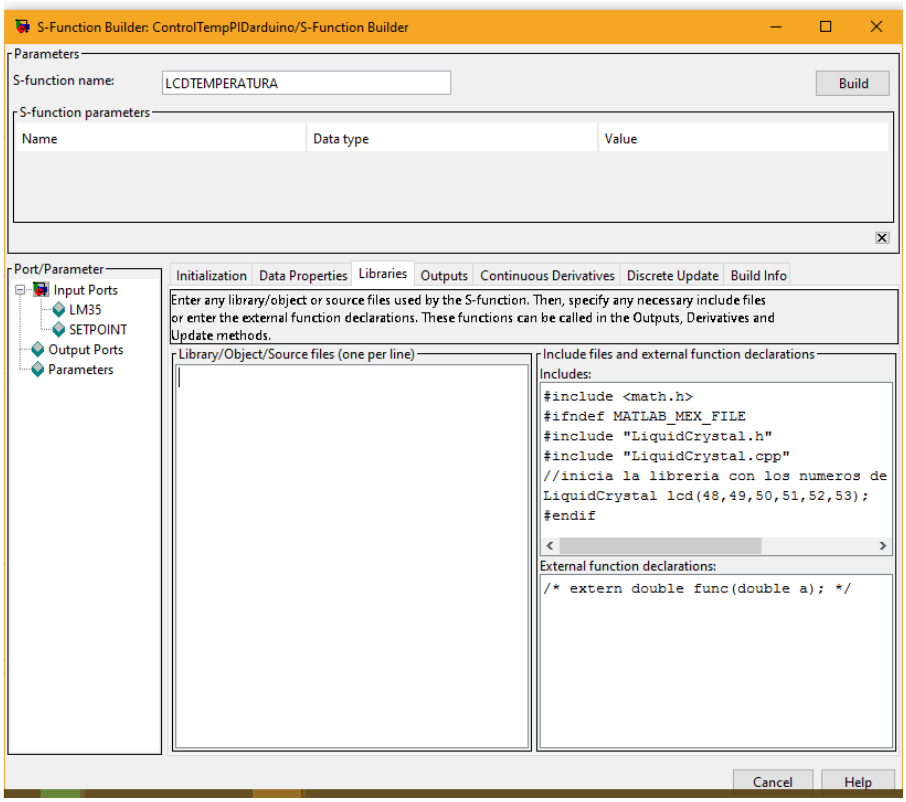

Step
Respuesta
Error

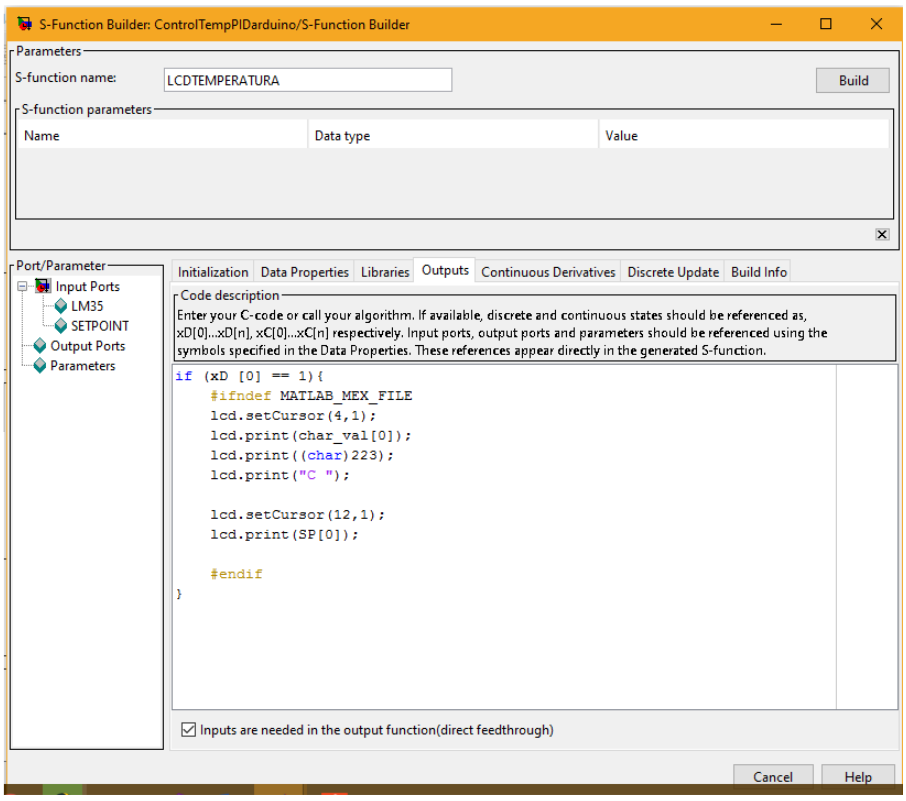

Display7
Display6
ARDUINO
Pin 3
BOMBILLO
PID(s)
PID Controller
Saturation1
Display3
num(z)
1
Discrete FIR Filter1
Display5
Display4
ARDUINO
Pin 2
SETPOINT
Product1
Divide1
491
Constant2
1023
Constant3
Scope3
Display8
ARDUINO
Pin 4
VENTILADOR
Saturation2
Abs
|u|
Saturation3
Display
num(z)
1
Discrete FIR Filter
Display2
ARDUINO
Pin 0
LM35
Product
Divide
Display1
491
Constant
1023
Constant1

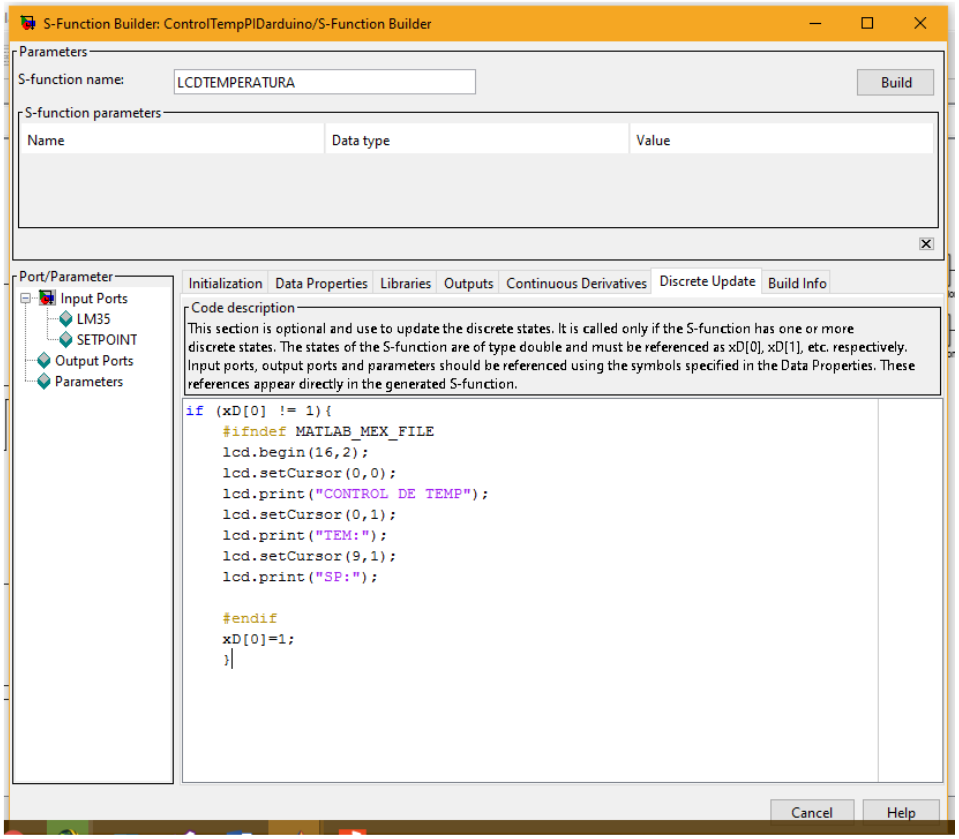


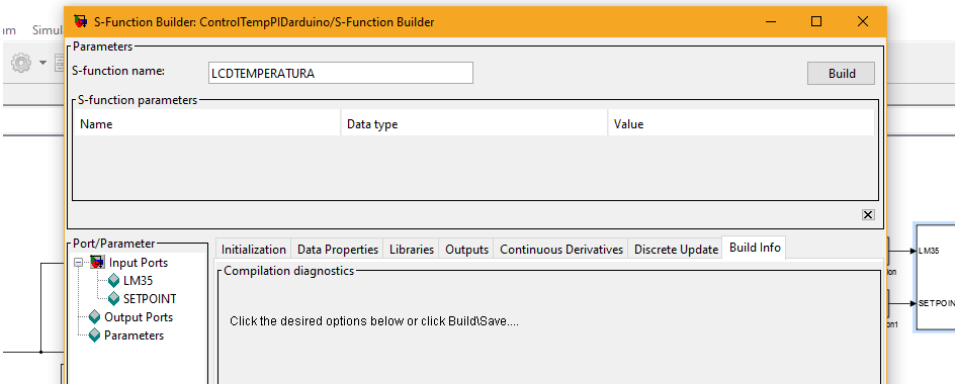
A continuación, se procede a la creación del bloque para la pantalla LCD 16X2 con **S-FUNTION BUILDER** de la librería de Simulink, se abre su configuración y se realizan los siguientes cambios.




S-Function Builder: ControlTempPIDarduino/S-Function Builder
Parameters
S-function name:
LCDTEMPERATURA
Build
S-function parameters
Name
Data type
Value
Port/Parameter
Input Ports
LM35
SETPOINT
Output Ports
Parameters
Initialization
Data Properties
Libraries
Outputs
Continuous Derivatives
Discrete Update
Build Info
Enter any library/object or source files used by the S-function. Then, specify any necessary include files or enter the external function declarations. These functions can be called in the Outputs, Derivatives and Update methods.
Library/Object/Source files (one per line)
Include files and external function declarations
Includes:
#include <math.h>
#ifndef MATLAB_MEX_FILE
#include "LiquidCrystal.h"
#include "LiquidCrystal.cpp"
//inicia la libreria con los numeros de
LiquidCrystal lcd(48,49,50,51,52,53);
#endif
External function declarations:
/* extern double func(double a); */
Cancel
Help

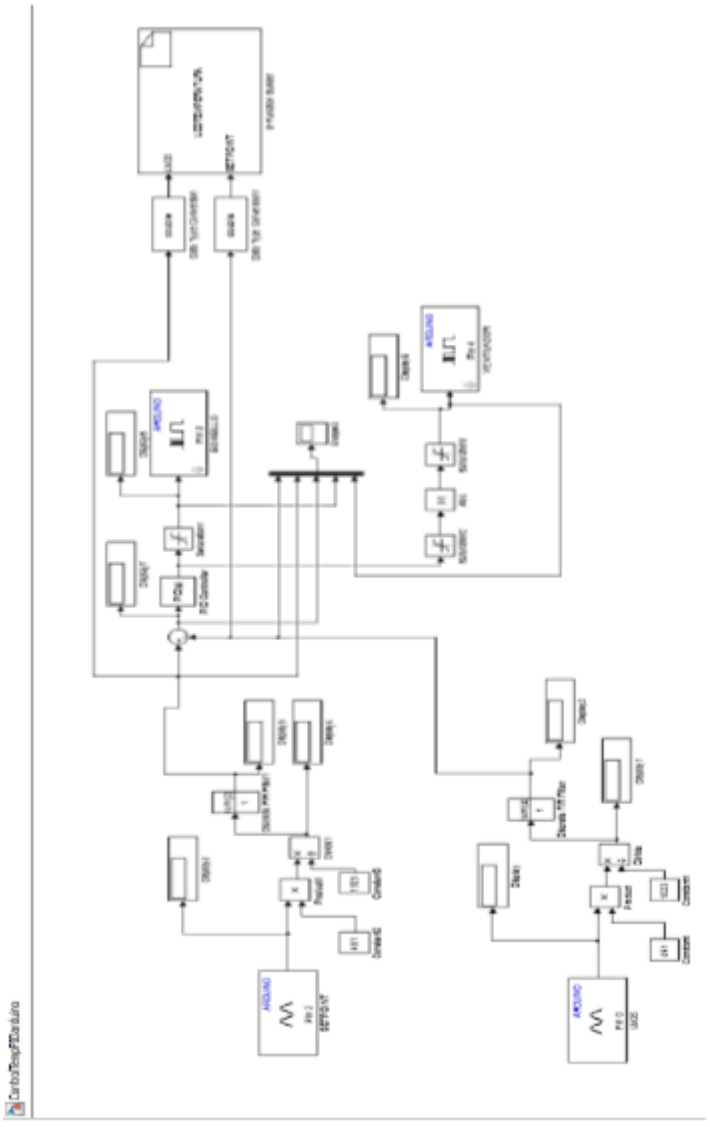

S-Function Builder: ControlTempPIDarduino/S-Function Builder
Parameters
S-function name:
LCDTEMPERATURA
Build
S-function parameters
Name
Data type
Value
Port/Parameter
Input Ports
LM35
SETPOINT
Output Ports
Parameters
Initialization
Data Properties
Libraries
Outputs
Continuous Derivatives
Discrete Update
Build Info
Code description
Enter your C-code or call your algorithm. If available, discrete and continuous states should be referenced as, xD[0]...xD[n], xC[0]...xC[n] respectively. Input ports, output ports and parameters should be referenced using the symbols specified in the Data Properties. These references appear directly in the generated S-function.
if (xD [0] == 1){
#ifndef MATLAB_MEX_FILE
lcd.setCursor(4,1);
lcd.print(char_val[0]);
lcd.print((char)223);
lcd.print("C ");
lcd.setCursor(12,1);
lcd.print(SP[0]);
#endif
}
Inputs are needed in the output function(direct feedthrough)
Cancel
Help





El archivo “LCDTEMPERATURA.C” se crea en Matlab al finalizar la compilación, lo siguiente que se debe hacer es agregar **extern “C”** renombrar el archivo con extensión **cpp**.

```
 */
extern "C" void LCDTEMPERATURA_Outputs_wrapper(const real_T *char_val,
            const real_T *SP,
            const real_T *xD)
{
/* %%%-SFUNWIZ wrapper Outputs Changes BEGIN --- EDIT HERE TO  END */
```



CODIGO EN ARDUINO

```
#include <PID_v1.h>
#include <LiquidCrystal.h>

LiquidCrystal lcd (48,49,50,51,52,53);

#define PIN_LM35 0
#define PIN_SETPOINT 2
#define PIN_BOMBILLO 3
#define PIN_VENTILADOR 4


double Setpoint=0, Sensor=0, Bombillo=0, Ventilador=0, Outbombillo=0, Outventilador=0;

double Kp=10, Ki=2.5, Kd=1;
PID PIDBombillo(&Sensor, &Bombillo, &Setpoint, Kp, Ki, Kd, DIRECT);
PID PIDVentilador(&Sensor, &Ventilador, &Setpoint, Kp, Ki, Kd, REVERSE);

void setup()
{
  //initialize the variables we're linked to
  Serial.begin(9600);
  lcd.begin(16,2);
  lcd.print("  SP  =        C");
  lcd.setCursor(0,1);
  lcd.print(" LM35 =        C");

  Sensor = analogRead(PIN_LM35);
  Setpoint = analogRead(PIN_SETPOINT);

  //turn the PID on
  PIDBombillo.SetMode(AUTOMATIC);
  PIDVentilador.SetMode(AUTOMATIC);

}

void loop()
{

  Sensor = analogRead(PIN_LM35);
  Sensor=(Sensor*491)/1024;
  //Sensor=map(Sensor,0,491,0,150);
  Setpoint = analogRead(PIN_SETPOINT);
  Setpoint=(Setpoint*491)/1024;
```

```
  Setpoint=map(Setpoint,0,491,0,120);


  PIDBombillo.Compute();
  PIDVentilador.Compute();

  Outbombillo=map(Bombillo,0,255,0,120);
  analogWrite(PIN_BOMBILLO, Bombillo);
  Outventilador=map(Ventilador,0,255,0,12);
  analogWrite(PIN_VENTILADOR, Ventilador);

  lcd.setCursor(10,0);
  lcd.print(Setpoint);
  lcd.setCursor(10,1);
  lcd.print(Sensor);

  if(Setpoint<10&&Setpoint>0){
    lcd.setCursor(11,0);
    lcd.print("  ");
    lcd.setCursor(11,1);
    lcd.print("  ");
  }
  if(Setpoint<100&&Setpoint>9){
    lcd.setCursor(12,0);
    lcd.print(" ");
    lcd.setCursor(12,1);
    lcd.print(" ");
  }

  Serial.print(" Setpoint: ");
  Serial.print(Setpoint);
  Serial.print(" C,");
  Serial.print(" Temperatura LM35: ");
  Serial.print(Sensor);
  Serial.print(" C,");
  Serial.print(" Voltaje en Motor: ");
  Serial.print(Outventilador);
  Serial.print(" DC,");
  Serial.print(" Voltaje en Bombillo: ");
  Serial.print(Outbombillo);
  Serial.println(" AC");

}
```

Conclusiones

- Como primera consideración, el montaje de la planta se hizo de manera conjunta al interior de la caja, como se pensó en un inicio, pero cuando se empezaron a realizar las pruebas del funcionamiento, se pudo observar que el circuito de potencia generaba interferencia en la lectura del LM35 y las señales de la pantalla LCD, por lo tanto, se modificó el montaje, dividiendo los circuitos de potencia (donde se suministra la corriente alterna AC) y la configuración que permite leer la temperatura con el circuito necesario para la LCD (donde se alimenta con corriente continua DC); de esta manera evitamos compartir la misma protoboard con AC y DC eliminando las posibles interferencias en la circuitería.

- Para la identificación del modelo de la planta se necesitó tomar valores con intervalos más cortos ya que un sistema de control de temperatura trabaja con tiempos prolongados y por esto es sistema no se estabiliza rápidamente, por lo tanto, al tomar valores en intervalos más cortos podemos generar una función más precisa para que la planta reaccione de manera más rápida.

- Las gráficas que se generan al momento de estimar el modelo de la planta, presentan muchas oscilaciones debido a que los instrumentos no son de precisión, como por ejemplo la variable set point que se modifica a través de un potenciómetro, ante cualquier movimiento genera una señal que se ve como un disturbio en la planta. Por otra parte, el sensor genera una buena lectura, es decir con un error bajo, pero ya que es un elemento muy básico también presenta interferencias ante alguna corriente parasita o algún movimiento.

- Durante la implementación del controlador, observamos que el sistema se podría controlar mediante dos tipos de lenguaje; el primero gracias a la librería PID de Arduino con la que se pudo generar un código (anexo al

correo) que donde se pueden variar las constantes del controlador PID Kp, Ki y Kd. Por otra parte gracias a la investigación que hicimos en internet pudimos observar que también se podía implementar un código en Simulink, el cual permite hacer la vinculación de la tarjeta Arduino con el ordenador y suministrar dicho código para hacer el control de la planta, al igual que el código en la interfaz de Arduino se modifican las constantes del PID para sintonizar la planta pero con la ventaja de que se puede observar en tiempo real el comportamiento de las variables, Set point, esfuerzo de control, erro, PWM del motor y la bombilla.

- A pesar de que la estimación del modelo no alcanzo el 80%, el sistema ya compensado funciona correctamente y con una eficiencia muy buena, respondiendo de manera rápida ante una variación del set point y manteniendo la temperatura, activando la bombilla como el motor para regular de manera eficiente la lectura del sensor.

BIBLIOGRAFIA

- https://control-pid.wikispaces.com/
- https://www.mathworks.com/products/matlab.html
- https://www.arduino.cc/
- http://www.educachip.com/optoacoplador-que-es-y-como-utilizarlo/