Control Continuo-Discreto y Discreto - Discreto

Juan José Galindo
{ est.juan.galindo6 }@unimilitar.edu.co
Profesor: Andres Castro

***Resumen*—Para este laboratorio se presentarán la investigación, análisis, construcción y conclusiones tras la realización al control continuo-discreto y discreto-discreto con el uso de microcontroladores para un sistema térmico por recamaras, las características de este control se hallaron con base al modelado del sistema. Estableciendo las respectivas características y constantes para el sistema ya establecido.**

***Palabras clave*—Control, Discreto, Microcontrolador, PID, Matlab, Serial, Regulador,LabView, Continuo, Microcontrolador, Python**

.

I. INTRODUCCIÓN

En esta práctica se busca realizar la implementación de un control PID discreto y un controlador discreto para un sistema térmico con control de tipo continuo-discreto y discreto-discreto se presentan diferentes ventajas y desventajas en respuesta transitoria, error de estado estable y velocidad de estabilización, se realiza el respectivo procedimiento para su caracterización, para lo cual primero se establecen ecuaciones que relacionan el cambio de la temperatura con la intensidad del bombillo, para después realizar su respectiva linealización para determinar su espacio de estados, y respectiva función de transferencia. por tal razón se desea mostrar la forma en qué dicho control funciona al unir la tecnología del Arduino que es el microcontrolador utilizado junto con la tecnología computacional para vincular el dispositivo con sus controles y visualizar la temperatura a través de una interfaz gráfica de usuario y la planta bastante común dentro de la vida real, como lo es una planta térmica. Un modelo de sistema de control digital o discreto se puede analizar desde diferentes perspectivas, incluyendo algoritmos de control en tiempo discreto, la conversión entre los dominios analógicos y digital, teniendo en cuenta el rendimiento del sistema ya que uno de los aspectos más importantes en un sistema de control discreto es el tiempo de muestreo, porque a diferencia del control de tipo continuo analógico donde la señal siempre está presente, el objetivo aquí es muestrear cada cierto período, presentando así la señal de salida en pasos.

II. MARCO TEÓRICO

Software Matlab

Software Matlab es un sistema de cómputo numérico que ofrece un entorno de desarrollo integrado con un lenguaje de programación propio

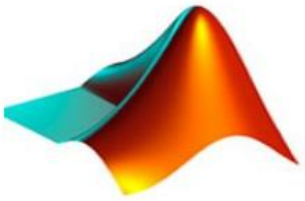


Figura 1: Logotipo MATLAB.

Microcontrolador

Un microcontrolador es un circuito integrado programable, capaz de ejecutar las órdenes grabadas en su memoria. Está compuesto de varios bloques funcionales que cumplen una tarea específica.

Figura 2: Ejemplo Microcontrolador.

Control discreto

Un sistema de control discreto es aquel que incluye un computador digital en el bucle de control para realizar un procesamiento de señal.

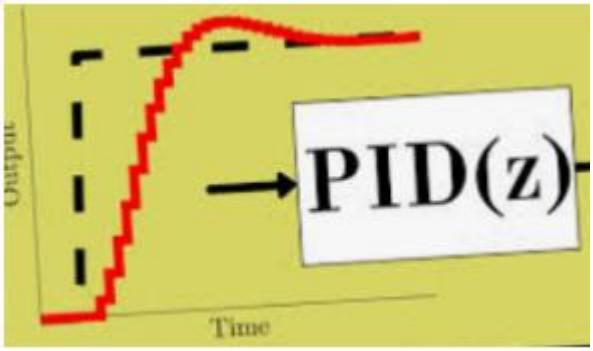


Figura 3: Dibujo alusivo a un Control Discreto.

III. COMPETENCIAS A DESARROLLAR

● Identifica las variables que intervienen en un problema de ingeniería.

● Propone y/o fórmula modelos que representan las relaciones de las variables de un problema.

● Establece los requerimientos de ingeniería que permiten la adecuada operación de un sistema, a fin de cumplir normativas y necesidades del usuario final.

IV. TRABAJO PREVIO

¿Cuántos métodos de discretización se encuentran en la literatura? Dar ejemplo de cada uno de ellos (por lo menos 4), aplicado a sistemas reales.

Step Invariance Method:

$$G_c(z) = (1 - z^{-1})\, z\, \{l^{-1}(\frac{G_c(s)}{s})\, |t = kT\}$$

Ejemplo: T=0.1 segundos.

$$G_c(s) = \frac{5(s+2)}{(s+1)(s+10)}$$

$$G_c(z) = (1 - z^{-1})\, z\, \{l^{-1}(\frac{5(s+2)}{s(s+1)(s+10)})\, |t = kT\}$$

$$G_c(s) = \frac{0.33381(z - 0.8198)}{(z - 0.9048)(z - 0.3679)}$$

- Euler hacia adelante:

$$\frac{d}{dt}\, x(t) = \frac{c(t+T) - x(t)}{T}$$

Aplicando Laplace se obtiene: $s = \frac{z-1}{T}$

Este método no preserva la estabilidad. En general, si T es suficientemente grande, el controlador discreto Gc(z) puede ser inestable incluso si Gc(s) es estable.

Ejemplo: Ts=0.1s

$$G(s) = \frac{3}{s^2 + 3s + 2}$$

$$G(z) = \frac{3}{(\frac{z-1}{T})^2 + 3(\frac{z-1}{T}) + 2}$$

$$G(z) = \frac{0.03}{z^2 - 1.7z + 0.72}$$

$$G(z) = \frac{0.03}{z^2 - 1.7z + 0.72} * \frac{z^{-2}}{z^{-2}}$$

$$G(z) = \frac{0.03z^{-2}}{1 - 1.7z^{-1} + 0.72z^{-2}}$$

- Euler hacia atrás:

$$\frac{d}{dt}\, x(t) = \frac{x(t) - x(t-T)}{T}$$

Aplicando Laplace se obtiene: $s = \frac{z-1}{Tz}$ Este método si preserva la estabilidad de Gc(s).

Ejemplo:

$$G(z) = \frac{3}{(\frac{z-1}{T})^2 + 3(\frac{z-1}{T}) + 2}$$

$$G(z) = \frac{0.02272z^2}{z^2 - 1.74224z + 0.7575}$$

- Tustin o transformación bilineal:

$$s = \frac{2(z-1)}{T(z+1)}$$

Ejemplo:

$$G(z) = \frac{3}{(\frac{2(z-1)}{T(z+1)})^2 + 3(\frac{2(z-1)}{T(z+1)}) + 2}$$

$$G(z) = \frac{0.0064935z^2 + 0.01298z + 0.0064935}{z^2 - 1.7229z + 0.740259}$$

$$G(z) = \frac{0.0064935 + 0.001298z^{-1} + 0.0064935z^{-2}}{1 - 1.7229z^{-1} + 0.740259z^{-2}}$$

¿Cómo se selecciona el tiempo de muestreo? Presentar 2 métodos diferentes y sus ejemplos respectivos (sistemas reales).

La mínima frecuencia de muestreo fs a la cual una función continua en el tiempo se discretiza sin que se pierda información debe ser el doble de la frecuencia de la componente de frecuencia más alta presente en la señal. (teorema de muestreo de Nyquist-Shannon)

$$\frac{1}{Ts} = fs \geq 2\, fmax$$

Ejemplo:

$$H(s) = \frac{1}{1 + sT_0}$$

$$f_B = f_0 = \frac{1}{2\pi T_0}$$

Frecuencia del ancho de banda Donde se atenúa 3dB

$$\omega_0 = \frac{1}{T_0} = 2\pi f_0$$

¿Qué es una ecuación en diferencias?

Se le llama ecuación en diferencias a cualquier expresión del tipo

$$F(Y_{t+n}, Y_{t+n-1}, Y_{t+n-2}, \ldots, Y_{t+1}, Y_t, t) = 0$$

Teniendo como solución de la misma, una sucesión y que la cumpla.

¿Qué es un retenedor de orden cero?

Su funcionamiento se basa en mantener constante la señal entre muestra y muestra. Es el más usado por su sencillez, y porque además suele venir incluido en los convertidores analógico-digital.

$$G(z) = (1 - z^{-1})Z(\frac{G(s)}{s})$$

¿Cómo se hace y se implementa la comunicación serial entre un microcontrolador y PC?

Protocolo de comunicación serie o serial

Este tipo de comunicación envía de forma secuencial un bit de dato por un canal o bus. aunque es más lento que la comunicación en paralelo, permite enviar datos a una distancia mayor y usando menos líneas. Es el protocolo más usado hoy en día, seguramente tu computadora o laptop tiene varios puertos USB. Este es un tipo de comunicación serial. Existen dos tipos de comunicación serial:

- Síncrona o sincrónica.
- Asíncrona o asincrónica

Comunicación Serial Síncrona

Este tipo de comunicación usa dos líneas, una por la cual se transmiten los datos y otra de reloj que sirve para sincronizar la transferencia de los datos. Esta forma de comunicación trabaja de la siguiente manera, el transmisor pone un bit en la línea de datos y el receptor espera a que se produzca un cambio del pulso de la señal de reloj, esto es que se produzca lo que se llama un flanco de bajada, en ese momento toma la lectura de la línea de datos. Esto lo hace para todos los bits que sean enviados.

Comunicación Serial Asíncrona

Este tipo de comunicación serial no tiene señal de reloj, por lo cual la sincronización la establece el emisor enviando unos bits especial al receptor

para que este conozca el inicio del mensaje y el fin del mensaje. Un parámetro importante en este tipo de comunicación es la velocidad de transmisión, ya que tanto el emisor como el receptor deben tener la misma velocidad. Esta velocidad se conoce como baud rate o baudios por segundo.
Ya conociendo los tipos de comunicación serial voy a explicar los más usados en proyectos con microcontroladores. Si haz realizado algún proyecto de electrónica con microcontroladores es posible que ya hayas utilizado alguno de estos protocolos de comunicación.

UART (Universal Asynchronous Receiver - Transmiter)

Este protocolo de comunicación es del tipo asíncrono, por lo cual no se requiere una señal de reloj. Usa 3 líneas, una para transmitir datos, una para recibir datos, y otra de tierra. Los pines usados para este tipo de comunicación están rotulados en las tarjetas de desarrollo con las letras RX y TX. Cuando se configura este tipo de comunicación se deben colocar ciertos parametros iguales en ambos dispositivos que se van a comunicar, de lo contrario la comunicación fallara. Los parametros mencionados son los siguientes:

● Velocidad de la transmisión: Esta se expresa en baudios por segundo y es la cantidad de bits que se transmitirán en un segundo.

● Bits de parada: La cantidad de bits que le indican al receptor que la transmisión del paquete ha terminado.

● Bit de paridad: Este bit se usa para determinar si hubo error en la transmisión. Es una forma sencilla de verificación de errores. Este valor puede ser: impar, par, o sin bit de paridad.

I2C (Inter - Integrated Circuit)

Este protocolo de comunicación serial es síncrono, es decir que si usa señal de reloj. Este usa dos líneas, una de datos y otra de reloj. A diferencia del UART donde un dispositivo es emisor y receptor al mismo tiempo, aquí uno de los dispositivos es maestro y los demás son esclavos. El maestro es el que inicia la comunicación. Este protocolo puede trabajar en forma de bus donde los dispositivos se conecten a las mismas líneas para intercambiar datos, cada dispositivo tiene una dirección hexadecimal que lo diferencia dentro del bus.

Una consideración importante del protocolo I2C es que trabaja del modo half duplex, es decir, que la comunicación va en una sola dirección en cada momento.
La mayoría de los microcontroladores soportan este protocolo de comunicación. En las tarjetas de desarrollo se identifica con las letras SCL para la señal de reloj y SDA para los datos.

SPI (Serial Peripheral Internet)

Este protocolo de comunicación es síncrono, por lo cual usa una señal de reloj. Usa dos líneas para la transferencia de datos, lo cual hace posible que trabaje en full duplex, es decir, puede enviar y recibir datos al mismo tiempo. Los periféricos pueden conectarse a bus de datos al igual que en el protocolo I2C, pero SPI no trabaja dirección del periférico, sino que tiene una línea llamada SS (Slave Select) que permite informarle al esclavo cuando debe enviar información. En resumen el protocolo SPI tiene 4 líneas. Estas son:

● MISO (Master In Slave Out): Esta es la línea usada por el esclavo par enviar datos al master.

● MOSI (Master Out Slave IN): Esta es la línea usada por el master para enviar datos al esclavo.

● SCK (Serial Clock): Esta es la señal de reloj para la sincronización de la comunicación.

● SS (Slave Select): Es la la línea que indica cuando el master quiere hablar con el esclavo. Puede existir más de una línea de este tipo para conectar varios esclavos.

I2S (Integrated Interchip Sound)

Este es un protocolo serial del tipo síncrono usado para conectar dispositivos de audio digital. La conexión es similar a I2C, ya que también pueden conectarse varios dispositivos a un bus de datos. Este protocolo usa 3 líneas, estas son:

● SCK (Serial Clock): Esta es la línea de la señal de reloj.

● FS (Frame Select): Esta línea se usa para seleccionar el canal izquierdo o derecho. También se puede encontrar con las letras WS (Word Select).

● SD (Serial Data): Esta es la línea por donde se envían los datos.

Al igual que en I2C debe haber un dispositivo master. Este protocolo es mayormente usado para trabajar con audio en estereo y se puede conseguir en componentes de audio que manejan convertidores ADC y DAC.
Para este caso en específico que se decidió trabajar con el microcontrolador Arduino se explicará cómo se hace la conexión. La comunicación serial entre dos dispositivos únicamente utiliza 3 líneas que son:
Rx (recepción de datos) Tx (transmisión de datos) GND (común)

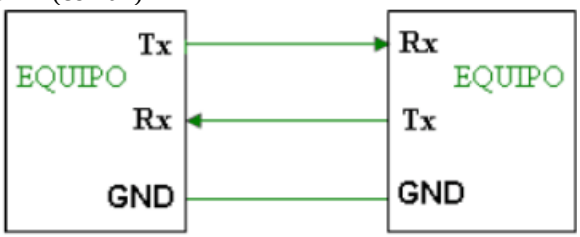


Figura 4: Protocolo comunicación serial Arduino.

¿Cómo se diseña una interfaz gráfica de usuario en C + + o C, tal que, permita mostrar datos gráficos y en textbox que se reciben por el puerto serial, modificar una variable por teclado-local y enviarla por el puerto serial? Ilustre con un ejemplo y anexe el código.

Funciones

MiniWin es super-simple: sólo es un miniconjunto de funciones. Para usar MiniWin solamente hay que seguir 2 pasos importantes:

● Poner arriba del programa principal:
● #include "miniwin.h"
● using namespace miniwin;

Fíjate en que hay comillas dobles y no ángulos alrededor de miniwin.h en el #include. El using namespace miniwin te será familiar por su equivalente con std.

● Hacer la función main así:
● int main() {
● return 0;}

Es decir, sin parámetros y devolviendo int. El return 0 es obligatorio. Aparte de eso se trata de utilizar las funciones que se comentan a continuación.

Control de la ventana

En MiniWin, al ejecutar el programa principal siempre se creará una sola ventana, a la que nos referiremos como “la ventana”. Esta ventana no se puede redimensionar con el ratón (solamente con la acción vredimensiona()) y mantiene el dibujo que pintas aunque la minimices. Algunas funciones utilizan coordenadas en esta ventana. Las coordenadas son un par ordenado de valores, donde el primer valor es x y el segundo es y:

● La esquina superior-izquierda es el orígen, con coordenadas (0,0).
● A medida que nos desplazamos a la derecha la coordenada x crece.
● A medida que nos desplazamos hacia abajo la coordenada y crece.
El diagrama sería el siguiente:

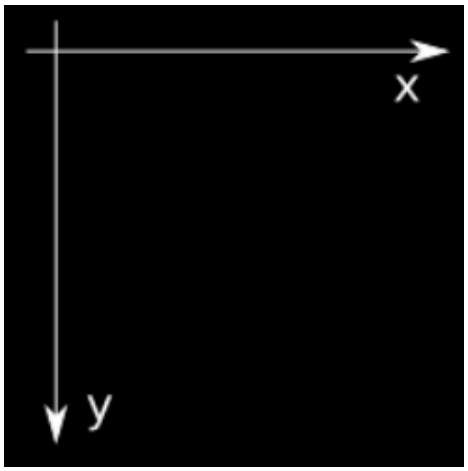


Figura 5: plano de la ventana creada.

Hay 3 funciones relacionadas con las dimensiones de la ventana.

void vredimensiona(int ancho, int alto)

Acción que cambia las dimensiones (en píxeles) de la ventana. El primer parámetro indica la anchura y el segundo la altura, ambos son enteros. Por ejemplo:

vredimensiona(800, 600);

cambiará las dimensiones de la ventana a 800 por 600 píxeles. El hecho de redimensionar la ventana implica que ésta se borrará, como si hiciésemos borra() justo después del vredimensiona().

int vancho()

Función que averigua el ancho de la ventana en píxeles, devolviendo un entero. Por ejemplo, el siguiente código utiliza la función vancho():

```
int a = vancho();
if (a > 500) {
cout << "El ancho de la ventana es mayor que 500" << endl; }
```

int valto()

Función que averigua la altura de la ventana en píxeles, devolviendo un entero. Es similar a vancho().

void vcierra()

Acción que cierra la ventana y termina el programa. Si no se llama esta acción, cuando acaba la función main la ventana se queda abierta mostrando el dibujo que hayamos hecho, y hay que cerrarla manualmente. Esto nos puede interesar para observar el dibujo que hayamos hecho.

Pintar en la ventana

Para pintar en la ventana hay que utilizar alguna de las acciones linea(), rectangulo(), circulo(), etc. y luego hay que invocar la acción refresca(). Esencialmente, todo lo que se pinta se acumula en un "buffer" y luego la acción refresca() hace visible en la ventana lo que se haya pintado previamente. Esto tiene una ventaja y un inconveniente. La ventaja es que permite pintar muchas cosas y luego refrescar solo una vez, que es importante cuando se hacen juegos. El inconveniente es que si se olvida la llamada a refresca(), entonces no aparece por pantalla nada de lo que se ha pintado y puede parecer que no funciona nuestro programa. En definitiva, es importante recordar llamar a :cpp:func:`refresca` al acabar de pintar.
Para cambiar el color con el que se pinta, hay que llamar a la función color() antes de pintar, es decir, todo lo que se pinta después de la instrucción:

```
color(ROJO);
```

saldrá en color rojo.
Por ejemplo, el siguiente programa

```
#include "miniwin.h"
using namespace miniwin;
int main() { vredimensiona(200, 200);
linea(0, 0, 100, 100);
color(AZUL);
rectangulo(10, 10, 50, 50); color(ROJO);
circulo(100, 50, 20); color(AZUL);
circulo_lleno(50, 100, 20);
color(VERDE);
rectangulo_lleno(50, 50, 100, 100);
refresca(); }
```

muestra el siguiente dibujo:

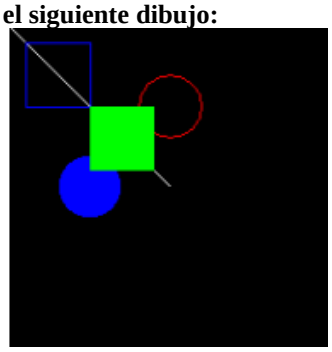


Figura 6: ejemplos de figuras creadas.

En MiniWin disponemos de las siguiente acciones para pintar objetos:.

void punto(float x, float y)

Pinta un solo punto de la pantalla, en la posición (x, y). En función de la resolución de la pantalla esto puede costar un poco de ver.
Por ejemplo:

```
color(AMARILLO);
punto(10, 10);
punto(9, 10);
punto(10, 9);
```

Para realizar la interfaz gráfica de usuario,

GUI por sus siglas en inglés, primero es necesario abrir el puerto serial por donde se van a transmitir y recibir los datos. Tal como se observa a continuación:

```
namespace Arduino_GUI{
  public partial class Form1 : Form {
    public Form1(){
      InitializeComponent();
      serialPort1.Open();
      }
    }
  }
```

Figura 7: Apertura del puerto serial

Luego, para enviar datos a través de un botón, forma similar ´ a como se realiza con un cuadro de texto, se realiza de la siguiente forma

```
private void OnButton_Click(object sender, Even
{
    // Send command to the arduino to turn pin
    serialPort1.Write("A");
}
```

Figura 8: Código empleado para abrir enviar datos a través del puerto serial

Para modificar una variable y enviarla por teclado-local es necesario crear el textbox, y mediante la acción de un botón adquirir el dato String que contenga el textbox, para finalmente enviarlo como se indicó en la figura 3. Si bien el envío de datos se realiza con el comando" serialPort.write()", para recibir datos del microcontrolador es necesaria la sentencia"serialPort.Read()".

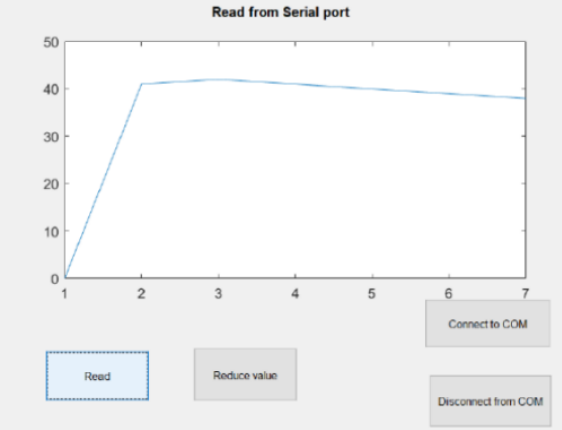


Figura 9: Ejemplo de una GUI que realiza comunicación serial con un µC

¿Cómo se garantiza el almacenamiento de los datos recibidos y enviados por el puerto, tal que, permitan posteriormente reconstruir el estado de un sistema?

archivos CSV

programas de hojas de cálculo pueden hacer el trabajo por debajo de los cálculos financieros complejos como los presupuestos de los hogares, los impuestos y las finanzas corporativas. A menudo utilizan sus propios formatos de archivo propietarios, sin embargo, que no transfieren fácilmente a otras aplicaciones, como procesadores de texto.

Una solución es escribir la información en un formato de valores separados por comas (CSV), que puede ser leído por muchos programas.

Instrucciones

1 Ejecutar un programa de hoja de cálculo que puede escribir archivos CSV. Los ejemplos incluyen el comercial de Microsoft Excel o WordPerfect Quattro Pro, o la libre OpenOffice Calc.
2 Ingrese la información de la hoja de cálculo que desea que aparezca en el archivo CSV. Formatos no se conservan en este tipo de archivos, por lo que dejan de lado cualquier definiciones de fuente, gráficos o fronteras. Si el archivo se compone de varias hojas de trabajo, es necesario guardar cada hoja como un archivo separado.
3 Seleccione el menú "Archivo" y haga clic en "Guardar como" para mostrar el cuadro de diálogo. Seleccione "CSV" en el menú desplegable "Guardar como tipo" y escriba un nuevo nombre de archivo según sea necesario.
4 Haga clic en "Guardar" para escribir la hoja de cálculo a un archivo CSV.

Lectura en matlab

de valores separados por comas (CSV) archivos son archivos de texto que contienen datos separados por comas y saltos de línea. Una matriz, por ejemplo, podría ser almacenado en un archivo CSV que contiene tantas líneas como hay filas, cada línea que contiene los elementos de la fila, separados por comas. Los archivos CSV son útiles porque son de fácil comprensión, tanto para los seres humanos utilizando un editor de texto plano y los programas de ordenador. Utilice el comando "csvwrite" en MATLAB para almacenar una matriz en un archivo CSV.

Instrucciones

1 Definir una sencilla matriz de tres por tres escribiendo el siguiente comando en la ventana de comandos de MATLAB:
x = [[1 2 3]; [4 5 6]; [7 8 9]];
2 Escribir la matriz X a un archivo CSV denominado csvtest.txt con el siguiente comando:
csvwrite ('csvtest.txt', x)
La extensión .txt es una opción razonable para el archivo CSV, ya que los archivos CSV pueden ser leídas por un editor de texto plano. Otra de las posibilidades es común .dat. El contenido del archivo no depende de la extensión de archivo que

elija. El archivo se guarda en el directorio por defecto de MATLAB, y aparece en el panel de "directorio actual" en el escritorio de MATLAB.
3 Añadir dos argumentos opcionales t "csvwrite para compensar la matriz con comas dentro del archivo:
csvwrite ('csvtest.txt', x, 2,0)
Los dos argumentos corresponden a la fila y la columna de desplazamiento. El ejemplo anterior se añaden dos filas vacías de comas a la parte superior del archivo. El siguiente comando coloca el inicio de la matriz en la quinta columna de la segunda fila:
csvwrite ('csvtest.txt', x, 1,4)

¿Cómo programar un microcontrolador de manera que realice procesos paralelos y no secuenciales? Ilustre con un ejemplo, anexando el código y datasheet del micro usado.

Para el control paralelo y secuencial se hace por medio de Interrupciones El manejador de interrupciones es el responsable de manejar eventos asincrónicos. La mayoría de estos eventos provienen de periféricos de hardware tales como interruptores, switchs, sensores, etc. Por ejemplo, un usuario que presiona un interruptor que provoca que se genere una interrupción.
Es un evento asíncrono que detiene la ejecución del código actual de acuerdo con su prioridad (cuanto más importante es la interrupción, mayor es su prioridad y se suspenderá la interrupción de menor prioridad). Las rutinas que se ejecutan durante la interrupción se denominan "rutinas de servicio de interrupción" o "rutinas de servicio de interrupción".

NVIC (Administrador de vector de interrupciones anidadas): Es una unidad de hardware responsable de la gestión de interrupciones en un microcontrolador basado en Cortex-M, lo que hace que Distinga entre las excepciones del sistema que se originan en el núcleo de la tarjeta y las excepciones de hardware de los dispositivos periféricos externos (también conocidas como "solicitudes de interrupción"). El programador maneja las interrupciones usando ISR específicos, que están codificados a un nivel superior (generalmente usando el lenguaje C). Con la ayuda de una tabla indirecta que contiene direcciones en la memoria de la rutina del servicio de interrupción, el procesador conoce la ubicación de estas rutinas. Esta tabla generalmente se denomina tabla de vectores de interrupciones, y cada microcontrolador STM32 define su propia tabla de vectores de interrupciones.

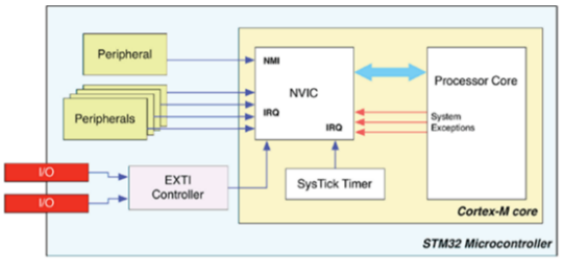


Figura 10: diagrama de funcionamiento de interrupciones para un sistema STM32.

Ejemplo de un programa realizando interrupciones

```
#include "stm32f4xx.h"
char BCD[10] = {126,48,109,121,51,91,95,112,127,123}; //vector con los numeros para el 7 segemnetos
int tiempoLED;  //variable que maneja el tiempo del led
int tiempoBCD=0; //variable que maneja el tiempo del conteo del 7 seg
```

Al principio del programa habilitamos la librería de la Stm32f4 que será la utilizada para la simulación en proteus. Se crea una variable BCD con 10 posiciones que indican los pines a encender para un contador de 0 a 9 siendo 126 el número binario para los segmentos del 0 y así sucesivamente.

Cifra	-gfedcba	Display
0000	00111111	0
0001	00000110	1
0010	01011011	2
0011	01001111	3
0100	01100110	4
0101	01101101	5
0110	01111101	6
0111	00000111	7
1000	01111111	8
1001	01100111	9
1010	01110111	A
1011	01111100	b
1100	00111001	C
1101	01011110	d
1110	01111001	E
1111	01110001	F

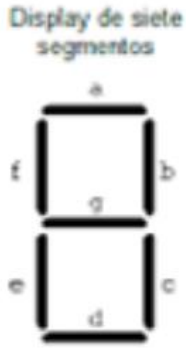


Figura 11: esquema de funcionamiento display 7 segmentos.

Las otras 2 variables serán utilizadas para poder manejar el tiempo de activación de los leds y el display.

```
extern "C"
{
void SysTick_Handler(void)  //instrucciones del systick
  {
  tiempoLED++;
  if (tiempoLED==2000){     //cuando el tiempo es de 2 seg se reinicia el tiempo del
  tiempoLED=0; }

  tiempoBCD++;
  if (tiempoBCD==10000){    //cuando el tiempo es de 10 seg se reinicia el tiempo de
  tiempoBCD=0; }

 }
```

Entramos así a definir el extern C, un tipo de main que se activara con respeto a un reloj Systick. El Hanlder es el que se encargara de dictar las operaciones que se encuentren dentro de su void. En este caso no indica hacer un contador para el tiempo del LED y del BCD y que sus valores se reinicien cuando lleguen a 2000 y 10000 respectivamente.

```
void EXTI0_IRQHandler(void)   //interrupcion en la linea 0
{
  EXTI->PR = 0X1;             //bandera que indica la interrupcion en pin
  GPIOB->ODR=BCD[0];          //7seg muestra el arreglo 0
  }

  void EXTI1_IRQHandler(void) //interrupcion en la linea 1
{
  EXTI->PR = 0X2;             //bandera que indica la interrupcion en pin
  GPIOB->ODR=BCD[1];          //7seg muestra el arreglo 1
  }

  void EXTI2_IRQHandler(void) //interrupcion en la linea 2
{
  EXTI->PR = 0X4;             //bandera que indica la interrupcion en pin
  GPIOB->ODR=BCD[2];          //7seg muestra el arreglo 2
  }

}
```

Dentro del extern C también tendremos los void para los 3 EXTI, correspondientes a los pulsadores que como su nombre indican, funcionaran como interrupciones de salida. Las 3 interrupciones dictan la misma instrucción: Cuando la línea este activa en el pin estipulado se le enviara una señal al display 7 segmentos. En la interrupción de la línea 0 se manda un 0, línea 1 un 1 y línea 2 un 2.

```
int main (void){
RCC -> APB2ENR |=(1UL << 14); //Habilitar el SYSCFG
RCC->AHB1ENR=0xFF; //Habilita el reloj de todos los puertos
//  GPIOB -> MODER |=0xAAAAAAAA;
  GPIOD -> MODER=0x55555555;  //puertos como salida para puertos D,C,B,A
  GPIOC -> MODER=0x55555555; //Pulsadores
  GPIOB -> MODER=0x55555555;
  GPIOA -> MODER=0x55555555;


SYSCFG -> EXTICR[0]=0X0222;   //Habilita el puerto C pines 0,1 y 2
  EXTI -> IMR|=7;       //Primeros 3 pines activados
  EXTI -> RTSR|=7;

  NVIC_EnableIRQ(EXTI0_IRQn); //Habilitar las interrupciones
  NVIC_EnableIRQ(EXTI1_IRQn);
  NVIC_EnableIRQ(EXTI2_IRQn);

  SystemCoreClockUpdate();
  SysTick_Config(SystemCoreClock/1000); //Systick en mili segundos
```

En el main tenemos varias activaciones:
· El APB2ENR es el que indica al pin correspondiente de habilitar el reloj de SYSCFG.
· AHB1ENR con el valor de FF que habilita los relojes de todos los puertos
· Puertos GPIO A, B, C y D habilita todo sus pines como salidas
· Ya en el SYSFG habilitamos los pines 0,1 y 2 del puerto C basados en la siguiente tabla

Y				
	0	L3	L2	L1	L0
	1	L7	L6	L5	L4
	2	L11	L10	L9	L8
	3	L15	L14	L13	L12

PUERTO (POR LINEA)			
PUERTO A	0	PUERTO E	4
PUERTO B	1	PUERTO F	5
PUERTO C	2	PUERTO G	6
PUERTO D	3	PUERTO H	7

· Los NVIC le dan autorización a las interrupciones de actuar en el programa.
· SysTick_Config es la variable de tiempo que opera en el programa, y al dividir el clock en 1000 se transforma su valor de 1 segundo a 1m segundo.

```
while(1){

  //Led titilar cada 2 seg
  if (tiempoLED >0 && tiempoLED <1000 ){
  GPIOA ->ODR=0x1;}
  else{
  GPIOA ->ODR=0x0;}

  //Switch para cada numero
  switch (tiempoBCD){
  }
```

Ya para finalizar el programa tenemos el while(true) que opera activamente sobre el código. Como primer instrucción es una sentencia if que hace que el led se prenda entre el intervalo de tiempo de 0 a 1000mSeg y que se apaga en cualquier otro intervalo, pero como anterior mente en el Handler del Sytick el valor de tiempoLED se reinicia cuando llega a 2000mSeg. Es decir que se prende cada 2 segundos y permanece 1 titilando. El ultimo switch es el del tiempo BCD

```
case 0:
{GPIOD->ODR=BCD[0];
    }
break;
```

Este opera con 10 casos: 0, 1000, 2000, 3000, 4000, 5000…9000. Quiere decir que cada segundo actuara en un caso y se reinicia al 0 de nuevo. En estos casos hay in conteo recorriendo los arreglos de BCD del 0-9. De este forma se hace el conteo ascendente con cambio cada segundo.

En la simulación conectamos LED al pin 0 del puerto A, 7 segmentos de conteo en los primeros 7 pines del puerto D y 7 segmentos de interrupción en los primeros 7 pines del puerto B y por último los pulsadores a los pines 0, 1 y 2 del puerto C.

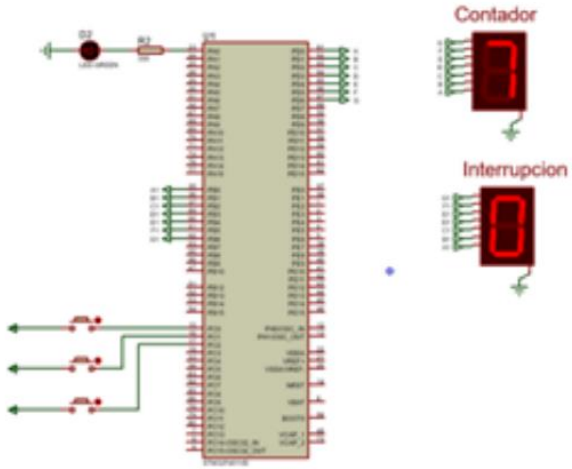


Figura 12: Simulación del programa.

V. DESARROLLO DE LA PRÁCTICA

En primer lugar se debe tener en cuenta el montaje en físico del siguiente sistema, incluyendo los sensores lm35, el bombillo alógeno, driver de potencia l298N, un Arduino, y el pc para la interfaz grafica

Figura 13: driver de potencia l298N

```
(9,map(u, 0, 10, 20, 255));
```

Figura 14: comando para el correcto manejo del driver de potencia l298N

Figura 15: arduino utilizado

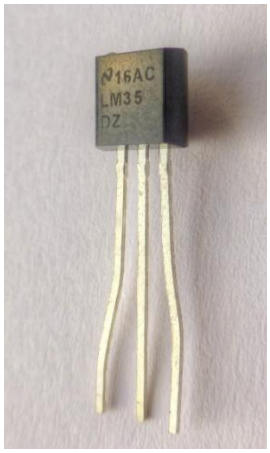


Figura 16: sensor lm35

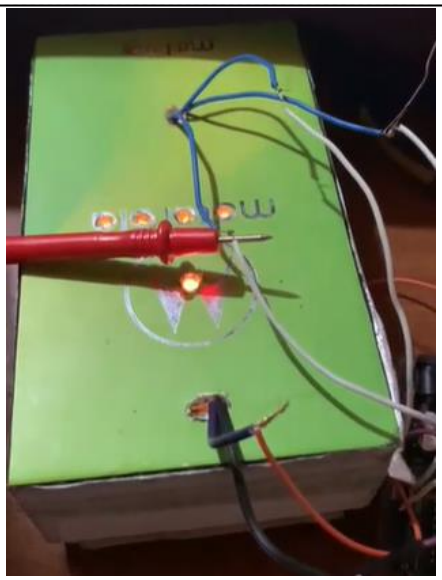


Figura 17: montaje completo de la planta termica

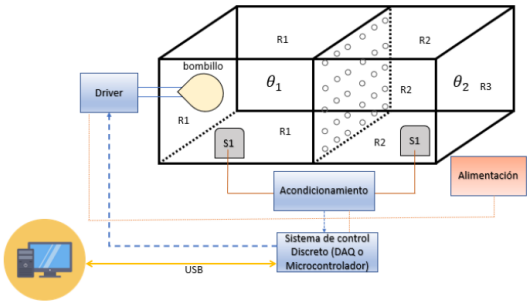
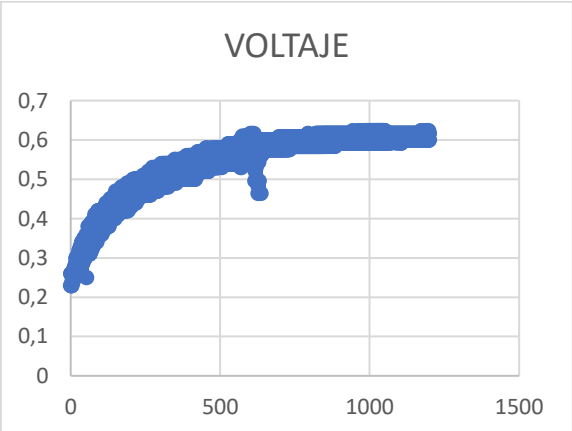


Figura 18: esquema de la planta termica

Para la caracterización de la planta se hizo una toma de datos para tener el valor experimental de en cuanto tiempo se estabilizaba y en que valor de ganancia aumentaba el sistema para poder obtener una función de transeferncia de 1 orden



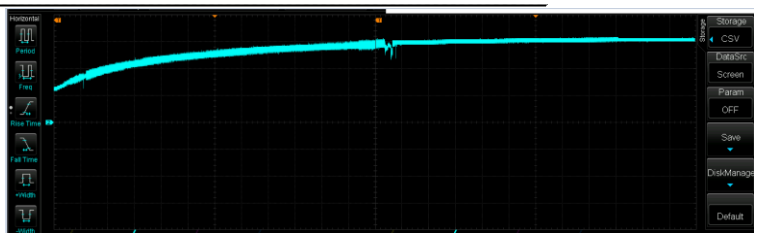


Figura 19: respuesta del sistema.

Teniendo los siguientes datos

ganancia	0,38
tiempo de establecimiento	1052
tao	210,4

Y obteniendo la siguiente función de transferencia

$$\frac{0.001806}{s + 0.004752}$$

Se calcula un polinomio deseado con los siguientes valores de control

$$ts = 300 \;\; Mp = 25\% \;\; \xi = 0.40$$
$$wn = 0.033 \; ts = 0.52677$$

$$PD = s^2 + 0.0264s + 1.89e^{-3}$$

Y se obtiene un polinomio caracteristico del sistema teneiendo en cuenta un control PID

$$s^2 + 0.004752s + 0.01806(kd\, s^2 + kp\, s + ki)$$

Igualando los polinomios tenemos

$$kd = 0$$
$$kp = 1.198$$
$$ki = 0.0602$$

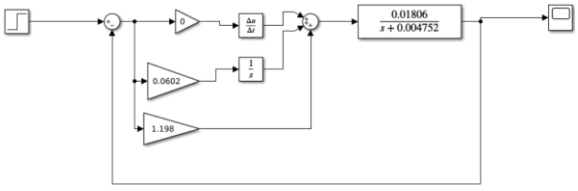


Figura 20: control PID simulado

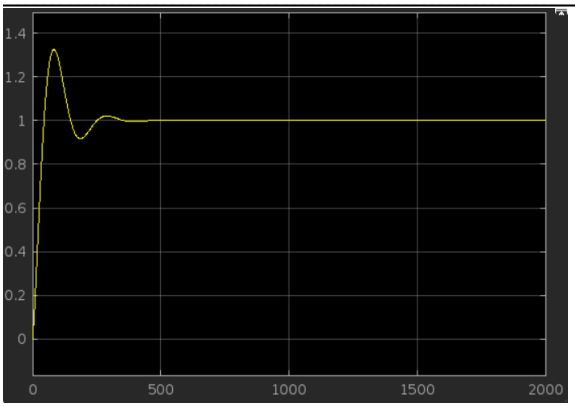


Figura 21: respuesta del control PID simulado

A ldiscretizar nuestra función de control tenemos un función tal que

$$\frac{s_o + s_1 z^{-1}}{1 - z^{-1}}$$

Y una ecuación en diferencias tal que

$$U(k) = s_0 e(k) + s_1 e(k-1) + u(k-1)$$

Donde:

$$s_o = kp + \frac{ki * ts}{2} = 1.2138$$

$$s_1 = -kp + \frac{ki * ts}{2} = -1.1821$$

$$\frac{1.2138 - 1.1821 z^{-1}}{1 - z^{-1}}$$

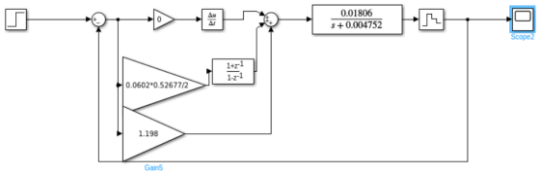


Figura 22: control PID simulado en discreto por ganancias

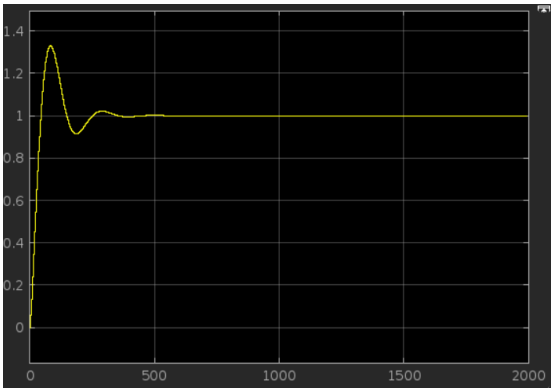


Figura 23: respuesta control PID simulado en discreto por ganancias

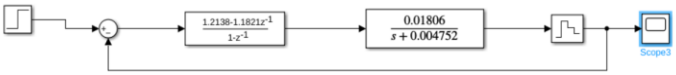


Figura 24: control PID en discreto simulado en discreto

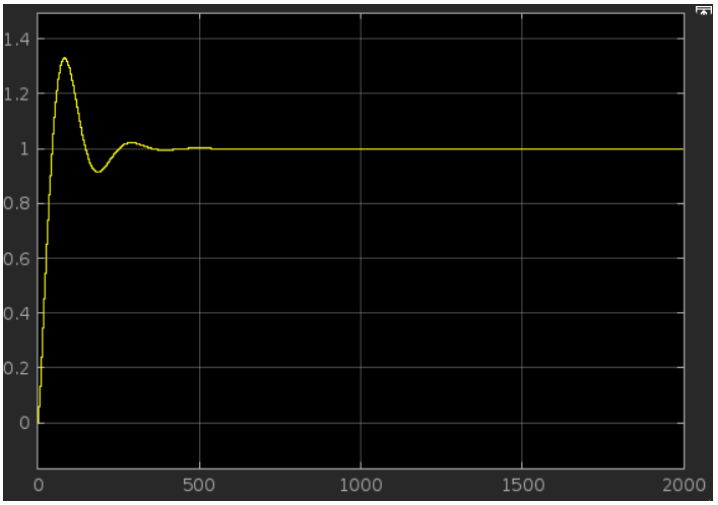


Figura 25: respuesta control PID en discreto simulado en discreto

Para la implementación se realizo la interfaz y la programación de la ecuación en diferencias en Python por lo cual

```
class App():
    def __init__(self):
        self.x_vals = []
        self.y_vals = []
        self.uk = []
        self.e_vals = []
        self.ref = 0.28
        self.valor_sensor = 0
        self.hab = True
        self.valor_tiempo = 0
        self.cont = 0
        self.ukm1 = 0
        self.ykm1 = 0
        self.ekm1 = 0
        self.q0 = 1.2138
        self.q1 = -1.1821
        self.ventana = Tk()
        self.ventana.title('Control Discreto')
        self.ventana.geometry('750x600')
        #self.ventana.background()
        self.dev = serial.Serial('COM4', baudrate = 115200, timeout=0.1)
```

Primero se definieron las variables y definimos el puerto serial a trabajar y las constantes so y s1 de nuestro sistema y posteriormente se programa la entrada, se realiza el error con una referencia y se incluye la ecuación en diferencias

```
try:
    self.ref = float(self.entrada_ref.get())
except:
    pass

self.e_vals[self.cont] = self.ref - self.valor_sensor #Error

if (self.ukm1+self.q0*self.e_vals[self.cont]+self.q1*self.ekm1) >= 10:
    self.uk[self.cont] = 10

else:
    self.uk[self.cont] = self.ukm1+self.q0*self.e_vals[self.cont]+self.q1*self.ekm1 #CONTROL PI  = u(k-1)+q0*e(k)+q1*e(k-1)
```

En arduinu el código es mas sencillo y lo que se propone es solo recibir los datos del sensor y darle un valor de tiempo enviando una cadena de caracteres para después ser graficadas mediante la interfaz de Python

```
void loop() {
  tiempo = millis();
  tiempo2 = (float)tiempo / 1000.0;
  voltaje = analogRead(A0);

  valor = voltaje * (5.0 / 1023.0);
  cadena = String(valor)+"-"+String(tiempo2);
  Serial.println(cadena);
```

Al graficar vemos el error que siempre debe de tender a 0 y el valor del sensor alcanzando una referencia colocada mediante una entrada en la ventana por el cual comprobamos el correcto funcionamiento de nuestro control.

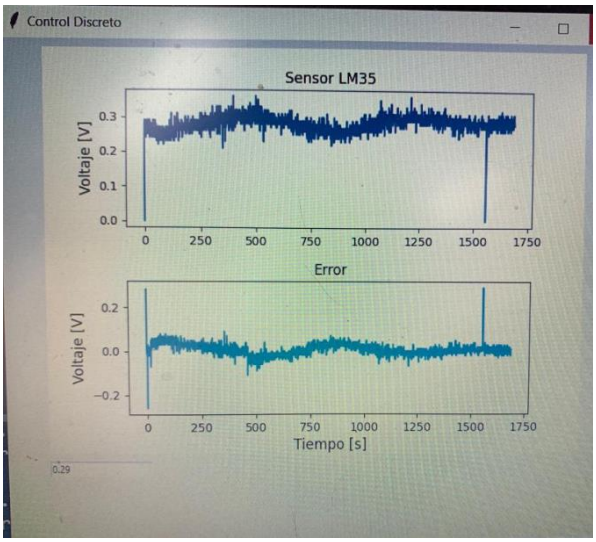


Figura 26: interfaz grafica donde evidenciamos lectura del sensor, error y la referencia.

Para el control discreto-discreto el primer paso es discretizar nuestra planta por el cual se utilizaron métodos de discretización como el retenedor de orden 0 (zoh) lo podemos ver a continuación

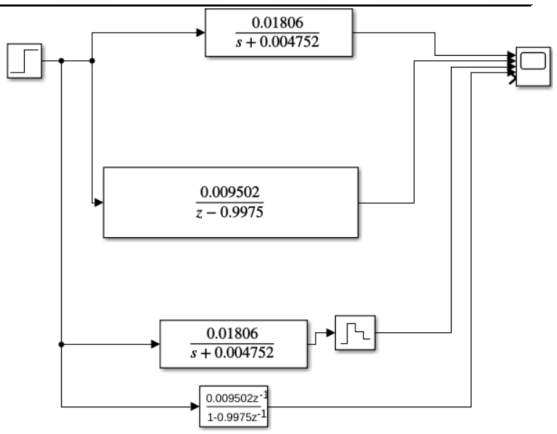


Figura 27: simulación de la planta discretizada.

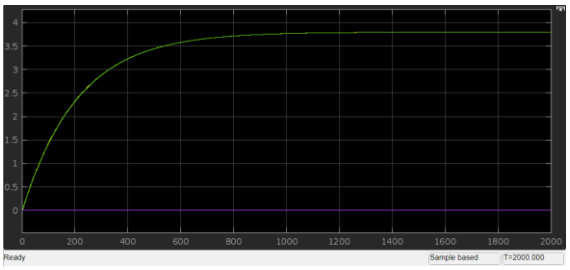


Figura 28: respuesta de cada una de las plantas discretizada, evidenciando la respuesta similar de cada sistema y verificando la correcta discretización.

Obtenemos un polinomio característico teniendo en cuenta:

$$pc = R(z^{-1}) * A(z^{-1}) + S(z^{-1}) * B(z^{-1})$$

Donde

$$\frac{s_o + s_1 z^{-1}}{1 - z^{-1}} = \frac{S(z^{-1})}{R(z^{-1})}$$

Y

$$\frac{0.009508 * z^{-1}}{1 - 0.9975 * z^{-1}} = \frac{B(z^{-1})}{A(z^{-1})}$$

Teniendo que

$$pc = 1 + (-1.9975 + s_0 0.009502)(z^{-1}) + (0.9975 + s_1 0.009502)(z^{-2})$$

Y obtenemos un polinomio deseado tal que

$$pd = 1 + p1(z^{-1}) + p2(z^{-2})$$
$$p1 = -2e^{\xi * wn * t_s} \cos\left(wn * T_s * \sqrt{1 - \xi^2}\right) = -0.1388$$
$$p2 = e^{-2\xi * wn * t_s} = 0.005092$$

E igualando polinomios tenemos que

$s_0 = 195.61$
$s_1 = -104.44$

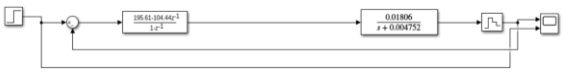


Figura 29: simulación de el control discretito discreto.

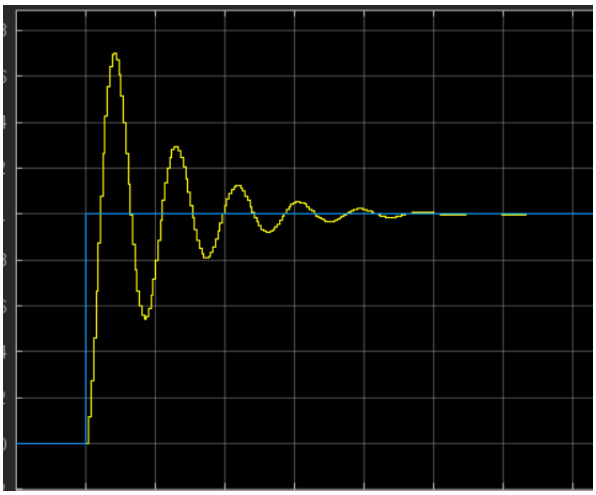


Figura 30: respuesta del control discretito discreto.

Para la implementación física realizamos la programación de la ecuación en diferencias en el microcontroloador tanto como la lectura y envio de datos y configuración de el serial

```
//Acción de Control
float u=0.0;
float u_1=0.0;
float e=0.0;
float e_1=0.0;
float ref=0.0;
float s0=1.2138;
float s1=-1.1844;
```

```
void setup() {

  Serial.begin(9600);
  pinMode(A0,INPUT);
  pinMode(9,OUTPUT);

  reflab=Serial.read();
  refst+=reflab;
  if(reflab=='\n')
  {
  refint=refst.toInt();
  ref=refint-10000;
  ref=ref/100;
  ref=ref/10;


tiempo = millis();
tiempo2 = (float)tiempo / 1000.0;
voltaje = analogRead(A0);

valor = voltaje * (5.0 / 1023.0);

  //Serial.println(cadena);

e=(ref-valor);
// Controle PID
u = s0*100*e + s1*100*e_1 + u_1 ; //Ley del controlador PID discreto
//Retorno a los valores reales
e_1=e;
u_1=u;
cadena = String(ref*100)+"|"+String(valor*100)+"|"+String(e*1000)+"|"+String(u*100);
Serial.println(cadena);
```

Para nuestra interfaz grafica utilizamos el software de labview en onde hacemos una progamacion intuitiva por bloques donde definimos los puertos, la lectura y el envio de datos, tomando en cuenta que los datos que se envían y se reciben son de formato string, donde debemos comvertirlos en tipo numerico

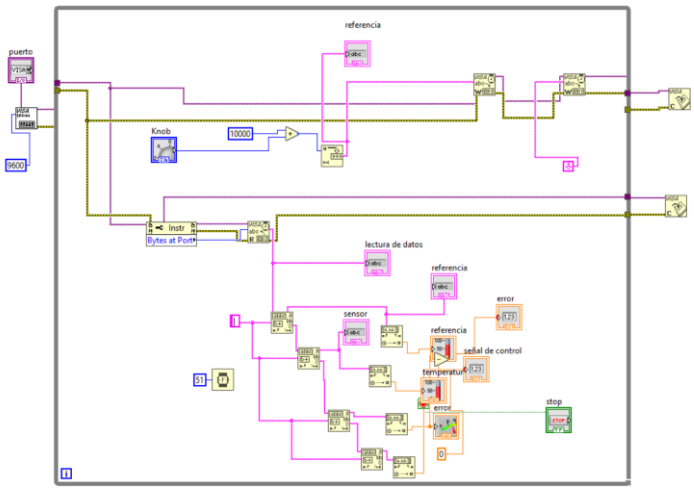


Figura 31: configuración de labview mediante programación por bloques para la lectura y envio de datos, haciendo uso de la interfaz grafica

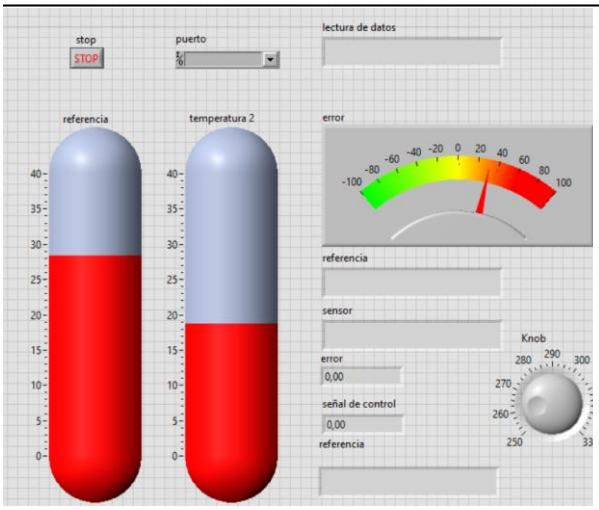


Figura 32: Interfaz grafica de labview

VI. CONCLUSIONES

● Se discretizó correctamente la planta por medio del método retenedor de orden cero, además se diseñó un controlador PID discreto el cual fue implementado en un microcontrolador en el cual no se obtuvieron los resultados esperados, sin embargo, se identificaron características que confirman el funcionamiento del mismo, además el controlador se simuló en Matlab comprobando su desempeño.

• Si no se calcula un tiempo de muestreo adecuado se van a generar medidas erróneas, es recomendable tener en cuenta el teorema de muestreo de NyquistShannon para evitar caer en dicho problema

• El control de forma discreta-discreta es mucho mejor que el método continuo-discreto, debido a que se pueden usar frecuencias más bajas, permitiendo aplicar los periodos de muestreo y frecuencia en la vida real, con las especificaciones de sistemas embebidos comercialmente conocidos y fáciles de encontrar.

• De acuerdo a los datos analizados y resultados favorables para ambos métodos de control es posible concluir que con el método de control Discreto-Discreto es más preciso que el del tipo continuo-discreto, ya que no se generan pérdidas de datos al encontrarse planta y regulador en el mismo mundo y bajo un tiempo de muestreo similar.

REFERENCIAS

[1] J. K. Author, “Title of chapter in the book,” in Title of His Published Book, xth ed. City of Publisher, Country, year.

[2] Youtube, canal Sergio Chaparro. https://youtu.be/IhvF6iY7n5k. Recuperado el 30 de Enero de 2017.

[3] Effen, M. C. (2004). MODELADO Y SIMULACIÓN DE SISTEMAS MECATRÓNICOS. Revista Investigación & Desarrollo, 1(3)

[4] B. C. KUO, Automatic Control Systems, Prentice-Hall, NJ, 5a. edición, 1987

[5] K. OGATA, Ingeniería de Control Moderna, Prentice-Hall Hispanoamericana, 4a. edición, 2003.

[6] G. H. HOSTETTER, C. J. SAVANT y R. T. STEFANI, Sistemas de Control, Interamericana, México, 1984.

[7] G. J. THALER y M. L. WILCOX, Máquinas Eléctricas, Editorial Limusa, México, 1974.

[8] AGUADO, A. & MARTÍNEZ, M., (2003); Identificación y control adaptativo, Prentice-Hall, Madrid, España

[9] Informática, U., Estudio, P., Transversales, E., Propietario, S. and 6.1, M., 2021. Matlab 6.1 - FCE. [online] Fce.unal.edu.co. Available at: o/ejes-tematicos-transversales/software-libre-y-propietario/1161-matlab -6-1.html> [Accessed 11 November 2021].

[10] HETPRO/TUTORIALES. 2021. Microcontrolador - qué es y para que sirve - HETPRO/TUTORIALES. [online] Available at: [Accessed 11 November 2021].

[11] Aleph.org.mx. 2021. ¿qué es un sistema de control discreto?. [online]Available at:<https://aleph.org.mx/que-es-un-sistema-de-control-discreto.[Accessed11 November 2021].

[12] Microcontroladores PIC 16F84. Desarrollo de proyectos. FernandoRamiro, Lucas J. López, Alfa omega, 2004 IX.

[13] MICROCONTROLADORES: FUNDAMENTOS Y APLICACIONES CON PIC

Pallas, Ramón; Valdés, Fernando (MARCOMBO, EDICIONES TÉCNICAS) 1ª edición, 2007 X.

[14] MICROCONTROLADORES PIC 2ª PARTE. PIC 16F87X. DISEÑO PRÁCTICO DE APLICACIONES Angulo Usategui, José María; Romero Yesa, Susana & Angulo Martínez, Ignacio (Editorial McGraw-Hill), 2ª edición, 2006 XI.

[15] Microelectronics, St. STM32F3xxx and STM32F4xxx Cortex-M4 programming Manual, 2013. URL: http: //www.st.com/web/en/resource/technical/document/programming_manual/DM00046982.pdf