Universidad de Costa Rica

Escuela de Ingeniería Eléctrica

IE0624 - Laboratorio de Microcontroladores

Informe del Laboratorio 3

PID, GPIO, ADC, comunicaciones USART y SPI

Profesor

Marco Villalta Fallas

Jose Daniel Blanco Solis, B71137

III 2024

6 de Febrero de 2025

Índice

Resumen

Este informe muestra el desarrollo e implementación de un sistema de control de temperatura basado en un controlador PID utilizando Arduino UNO, se presentan los aspectos teóricos del PID y componentes electrónicos empleados, se describe el diseño del hardware, la programación en Arduino para el control del sistema y la comunicación serial con un programa en Python que permite la adquisición y análisis de los datos generados, finalmente, se presentan los resultados obtenidos para el laboratorio y se analiza respecto a la funcionalidad del sistema implementado.

1. Introducción

El objetivo principal de este laboratorio es desarrollar un sistema de control de temperatura que logre mantener la temperatura de una incubadora dentro de un rango definido por el usuario, se utiliza un controlador PID que ajusta la señal de salida en función de la diferencia entre la temperatura medida y el setpoint establecido, el sistema también es capaz de comunicar sus resultados en tiempo real a través de un puerto serial para su posterior análisis.

Para este caso se implementa un sistema de regulación de temperatura basado en un controlador PID utilizando un Arduino UNO, este sistema emplea sensores para medir la temperatura, una pantalla LCD PCD8544 para visualizar los valores críticos del sistema y un canal de comunicación serial que permite la transferencia de datos a una computadora, donde son almacenados en un archivo CSV y posteriormente analizados mediante gráficos generados con Python.

2. Nota teórica

2.1. Controlador PID

Su funcionamiento se basa en la corrección del error entre la variable de proceso y el setpoint deseado, mediante la acción de tres términos, proporcional (P) responde de manera instantánea al error presente en el sistema generando una corrección que es proporcional a la magnitud del error, el control proporcional por sí solo no es suficiente para eliminar completamente el error en estado estacionario.

Se utiliza el término integral (I) el cual acumula el error en el tiempo y ajusta la señal de control para eliminar el error permanente que se mantiene después de aplicar la acción proporcional.

El término derivativo (D) actúa como mecanismo de anticipación al predecir la tendencia del error y aplicar una corrección antes de que el error se vuelva significativo, para esta implementación se utiliza el diagrama de bloques de figura 1.

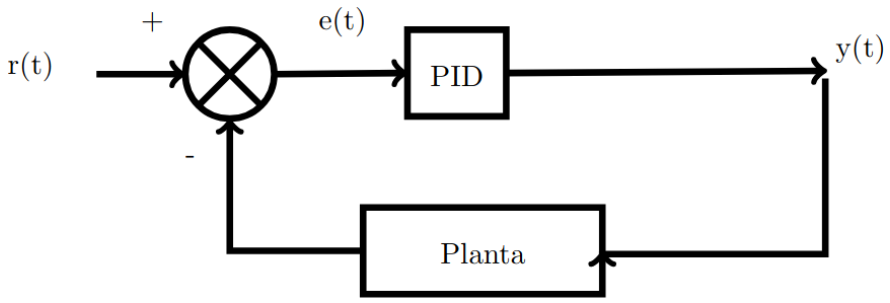

Figura 1: Diagrama de bloques del proceso térmico simulado.

2.2. Arduino UNO

El sistema de control PID se implementa en un Arduino UNO, el cual esta basado en el chip ATMega328P con arquitectura AVR de 8 bits, este dispositivo cuenta con múltiples pines de entrada y salida que permiten la lectura de sensores y el control de actuadores.

La lectura de temperatura se realiza mediante un sensor analógico que convierte la señal térmica en un voltaje proporcional, este valor es posteriormente procesado para generar la acción de control correspondiente y mostrar la información en la pantalla LCD, además, se emplea una interfaz serial para transferir los datos a una computadora, donde se registran y analizan, el esquema de puertos y conexiones de la placa se muestra en la figura 2.

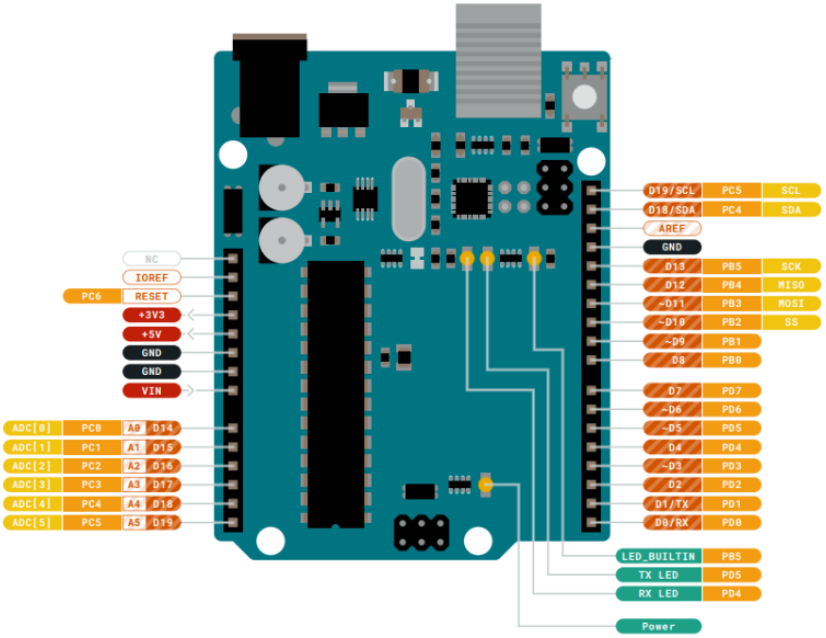

Figura 2: Puertos y conexiones del Arduino UNO.

2.3. LCD PCD8544

Para la visualización de los datos en tiempo real se utiliza la pantalla LCD PCD8544 que corresponde a un display gráfico monocromático, su integración con Arduino se facilita mediante el uso de la librería que permite inicializar y manejar el contenido mostrado en pantalla, se implementa un mecanismo de indicadores luminosos mediante LEDs para alertar sobre el estado del sistema: LED azul indica que la temperatura está por debajo del umbral inferior, LED verde confirma que el sistema opera dentro del rango permitido y LED rojo advierte que la temperatura ha excedido el límite superior establecido, el diagrama de bloques del display se muestra en la figura 3.

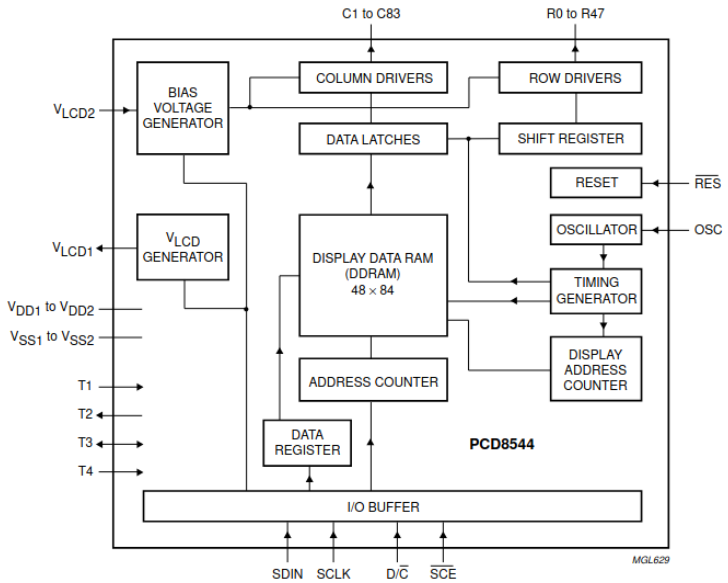

Figura 3: Diagrama de bloques del LCD PCD8544.

2.4. Diseño del circuito

Uno de los elementos fundamentales es el sensor de temperatura, el cual está conectado a una entrada analógica del Arduino para proporcionar mediciones, estas lecturas son procesadas por el controlador PID cuya salida se encarga de regular el comportamiento del sistema tratando de mantener la temperatura dentro de los parámetros establecidos, para la visualización de la información relevante, como se aprecia en la figura 4 se incorporó una pantalla LCD PCD8544, la cual se comunica con el Arduino mediante el protocolo SPI en esta se muestran constantemente los valores de temperatura medidos, el setpoint definido por el usuario y la señal de control generada por el PID, facilitando el monitoreo del sistema en tiempo real.

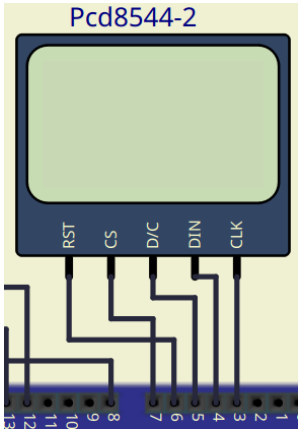

Figura 4: Conexión del LCD PCD8544 .

Se añadieron tres LEDs indicadores conectados a pines digitales, como se aprecia en la figura 5, los cuales reflejan distintos estados operativos según la temperatura detectada, estos actúan como señales visuales rápidas que permiten identificar si el sistema está dentro del rango adecuado, en proceso de ajuste o fuera de los límites establecidos, el cual se puede apreciar en la figura 5

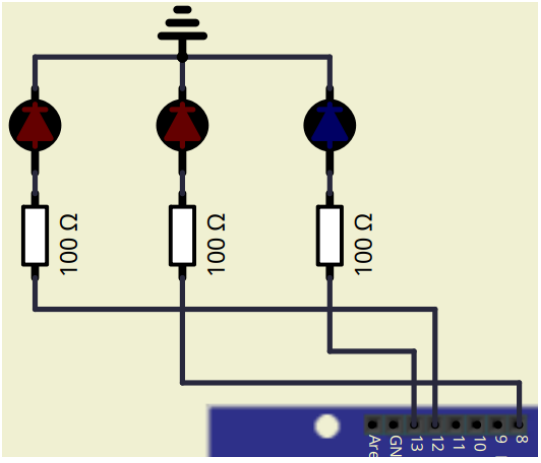

Figura 5: Circuito diseñado indicadores LED.

El circuito también cuenta con un sistema de comunicación serial basado en la interfaz USART, configurada a un baud rate de 9600 bps, lo que permite el envío continuo de datos desde el Arduino

hacia una computadora, asegurando un registro detallado del comportamiento del sistema, los datos incluyen la temperatura medida, setpoint y salida del PID, estos son capturados por un programa en Python utilizando la biblioteca pyserial y una vez recibidos se almacenan en un archivo CSV para su posterior análisis y procesamiento, mediante la biblioteca matplotlib se generan gráficas que permiten visualizar la evolución de la temperatura y el desempeño del controlador, permitiendo evaluar el comportamiento del sistema.

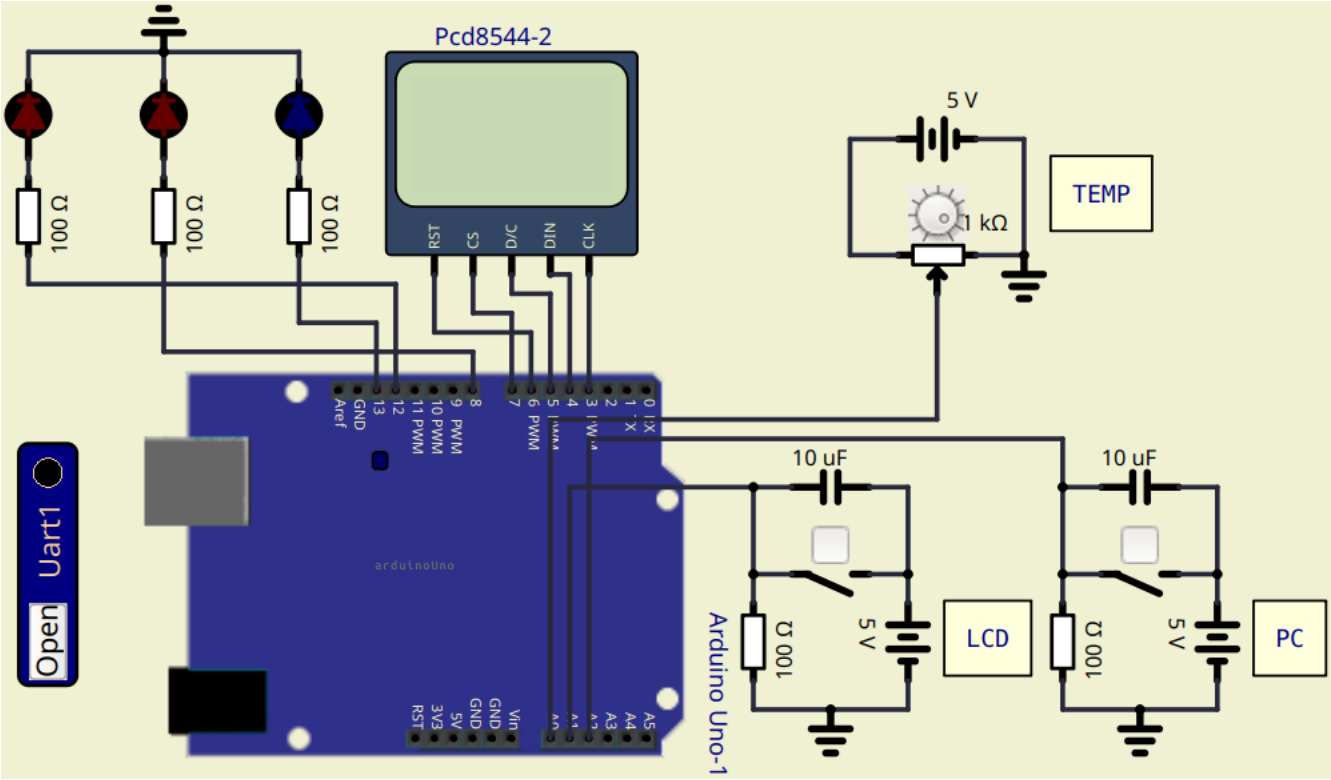

Figura 6: Circuito completo diseñado.

3. Desarrollo/Análisis

El código en Arduino se encarga de gestionar la lectura de sensores, el procesamiento de la señal de temperatura mediante el controlador PID y la actualización de la pantalla LCD con los valores relevantes, los parámetros iniciales del controlador PID fueron seleccionados y ajustados a través de pruebas sucesivas para lograr una respuesta estable, a continuación se presenta el código diseñado.

```
#include <PCD8544.h>
#include <PID_v1_bc.h>

// Pantalla LCD
PCD8544 lcd;

// Variables para el PID
double Input, Output, Setpoint;
PID myPID(&Input, &Output, &Setpoint, 15, 0.5, 1, DIRECT);

// Pines y configuraci n inicial
const int LED_LOW = 8, LED_MED = 12, LED_HIGH = 13;
const int SENSOR_TEMP = A0, SENSOR_LCD = A1, SENSOR_SERIAL = A2;

void setup() {
  Serial.begin(9600);
  pinMode(LED_LOW, OUTPUT);
  pinMode(LED_MED, OUTPUT);
  pinMode(LED_HIGH, OUTPUT);
  lcd.begin();
  myPID.SetOutputLimits(-125, 125);
  myPID.SetMode(AUTOMATIC);
}

void loop() {
  leerSensores();
  myPID.Compute();
  actualizarLCD();
  actualizarLEDs();
  enviarDatosSerial();
  delay(500);
}

void leerSensores() {
  Setpoint = map(analogRead(SENSOR_TEMP), 0, 1023, 20, 80);
  float tempWatts = (Output * 25.0) / 255.0;
  Input = simPlant(tempWatts);
}

```

```
40 void actualizarLCD() {
41   lcd.setPower(analogRead(SENSOR_LCD) > 700);
42   lcd.setCursor(0, 1); lcd.print("Opt:␣"); lcd.print(Setpoint);
43   lcd.setCursor(0, 2); lcd.print("Ctrl:␣"); lcd.print(Output);
44   lcd.setCursor(0, 3); lcd.print("Temp:␣"); lcd.print(Input);
45 }
46
47 void actualizarLEDs() {
48   digitalWrite(LED_LOW, Input < 30);
49   digitalWrite(LED_MED, Input >= 30 && Input <= 42);
50   digitalWrite(LED_HIGH, Input > 42);
51 }
52
53 void enviarDatosSerial() {
54   if (analogRead(SENSOR_SERIAL) > 700) {
55     Serial.print(Setpoint); Serial.print(",");
56     Serial.print(Output); Serial.print(",");
57     Serial.println(Input);
58   }
59 }
60
61 // Respuesta t rmica de la planta
62 float simPlant(float Q) { // heat input in W (or J/s)
63   // simulate a 1x1x2cm aluminum block with a heater and passive
       ambient cooling
64  // float C = 237; // W/mK thermal conduction coefficient for Al
65   float h = 5; // W/m2K thermal convection coefficient for Al passive
66   float Cps = 0.89; // J/g  C
67   float area = 1e-4; // m2 area for convection
68   float mass = 10; // g
69   float Tamb = 25; //  C
70   static float T = Tamb; //  C
71   static uint32_t last = 0;
72   uint32_t interval = 100; // ms
73
74   if (millis() - last >= interval) {
75     last += interval;
76     // 0-dimensional heat transfer
77     T = T + Q * interval / 1000 / mass / Cps - (T - Tamb) * area * h;
78   }
79   return T;
80 }
```

El programa en Python se encarga de recibir los datos enviados a través del puerto serial, almacenarlos en un archivo CSV y generar gráficos que permiten visualizar la evolución del sistema en el tiempo, a continuación se presenta el código diseñado.

```
import serial
import time
import csv
import pandas as pd
import matplotlib.pyplot as plt

ser = serial.Serial('/dev/pts/3', 9600)
counter = 0

data_file = "dataCollecter.csv"

# Archivo CSV para escritura
with open(data_file, "w", newline="") as csvfile:
    csv_writer = csv.writer(csvfile)
    csv_writer.writerow(["Tiempo (s)", "Setpoint", "Input", "Output"
       ])  # Encabezado

    try:
        while True:
            counter += 0.5

            # Leer desde puerto serial
            line = ser.readline().decode().strip()
            setpoint, input_val, output = map(float, line.split(","))

            # Escribir en el archivo CSV
            csv_writer.writerow([counter, setpoint, input_val, output
               ])

            # Datos en terminal
            print(f"[{counter}] {setpoint}, {input_val}, {output}")

    except KeyboardInterrupt:
        ser.close()
        print("Conexi n cerrada.")

# Leer datos desde el CSV
data = pd.read_csv(data_file)

# Extraer las columnas
tiempo = data.iloc[:, 0]  # Primera columna
setpoint = data.iloc[:, 1]  # Segunda columna
```

```
input_val = data.iloc[:, 2]  # Tercera columna
output = data.iloc[:, 3]  # Cuarta columna

# Graficar Setpoint y guardarlo como PNG
plt.figure(figsize=(8, 6))
plt.plot(tiempo, setpoint, label="Setpoint", color='r')
plt.xlabel("Tiempo (s)")
plt.ylabel("Setpoint")
plt.title("Temperatura de operacion del sistema")
plt.legend()
plt.grid(True)
plt.savefig("grafica_opt.png")  # Guardar gr fica de Setpoint como PNG
plt.close()  # Cerrar la gr fica para evitar que se superpongan

# Graficar Input y guardarlo como PNG
plt.figure(figsize=(8, 6))
plt.plot(tiempo, input_val, label="Input", color='g')
plt.xlabel("Tiempo (s)")
plt.ylabel("Input")
plt.title("Temperatura sensada del sistema")
plt.legend()
plt.grid(True)
plt.savefig("grafica_temp.png")  # Guardar gr fica de Input como PNG
plt.close()  # Cerrar la gr fica para evitar que se superpongan

# Graficar Output y guardarlo como PNG
plt.figure(figsize=(8, 6))
plt.plot(tiempo, output, label="Output", color='b')
plt.xlabel("Tiempo (s)")
plt.ylabel("Output")
plt.title("Se al de control del sistema")
plt.legend()
plt.grid(True)
plt.savefig("grafica_ctrl.png")  # Guardar gr fica de Output como PNG
plt.close()  # Cerrar la gr fica para evitar que se superpongan
```

3.1. Pruebas de funcionamiento

Las pruebas realizadas al sistema incluyeron tanto la verificación de las conexiones físicas del circuito como del código implementado, se revisaron todas las conexiones para garantizar que no hubiera errores en el cableado y posteriormente se cargó el programa en el Arduino UNO, los LEDs respondieron correctamente a las señales del microcontrolador, encendiéndose y apagándose según el rango, temperatura menor a 30^oC (figura 7), temperatura superior a 42^oC (figura 8) y temperatura entre 25^oC y 42^oC (figura 9).

Uno de los aspectos destacados fue la correcta visualización de los parametrso del sistema en el display LCD, durante las pruebas se observó que el display actualizaba los números en sincronía con el cambio de producido por el PID.

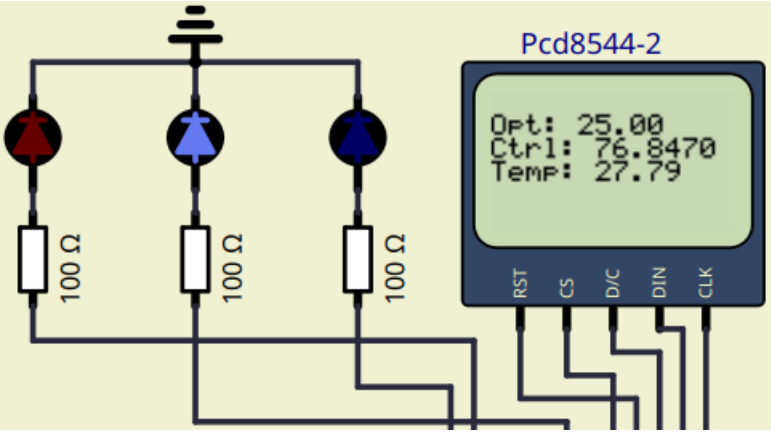

Figura 7: Ejecución en temperatura menor a 30^oC.

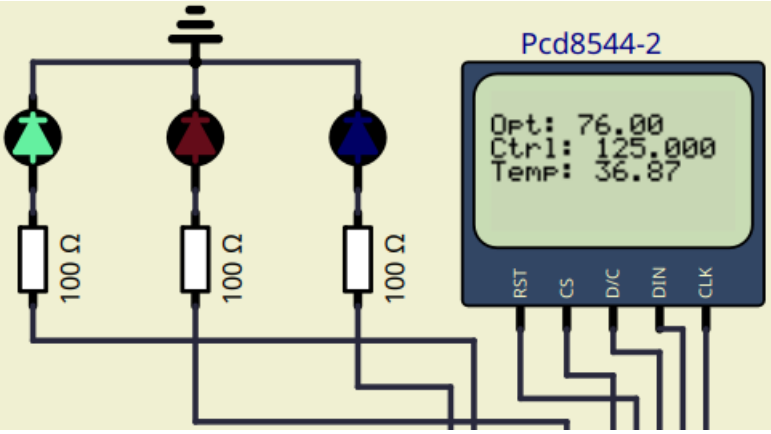

Figura 8: Ejecución en temperatura superior a 42^oC.

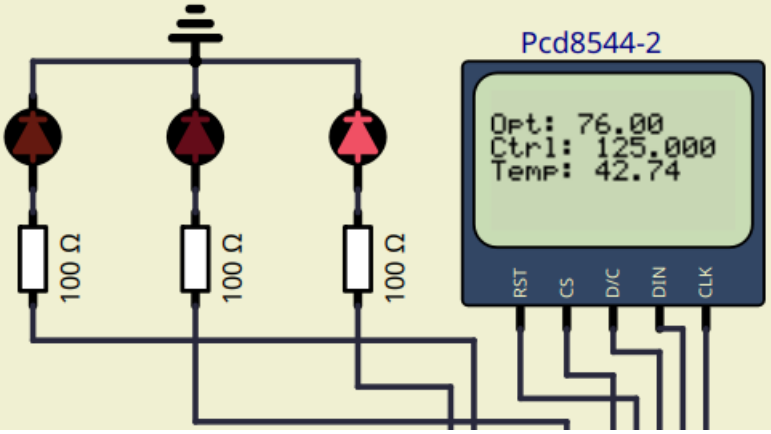

Figura 9: Ejecución en temperatura entre 30^oC y 42^oC.

La figura 10 muestra la evolución de la señal de control, inicia con un valor elevado para llevar la temperatura al setpoint deseado y a medida que la temperatura se estabiliza la señal de control se ajusta gradualmente, disminuyendo hasta acercarse a un valor más bajo, en el momento en que el setpoint cambia nuevamente la señal de control reacciona con un ajuste, para alcanzar el nuevo valor.

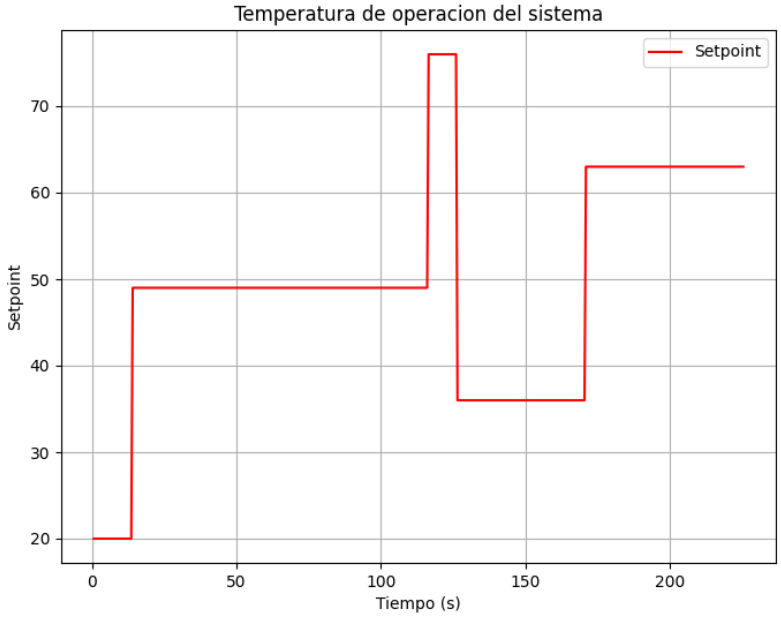

Figura 10: Gráfica de señal control.

La figura 11 corresponde a la temperatura de operación del sistema, se observa cómo el setpoint varía en distintos momentos, con incrementos y decrementos que indican ajustes manuales permitendo evaluar la capacidad del sistema para alcanzar y mantener la temperatura.

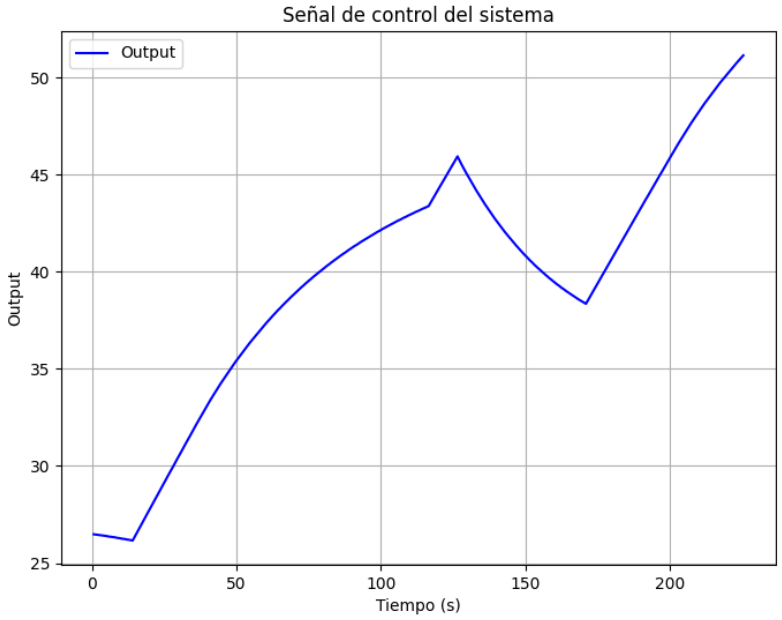

Figura 11: Gráfica de señal salida.

La figura 12 muestra la temperatura sensada del sistema, se observan cambios significativos en respuesta a los cambios del setpoint, inicialmente la temperatura sigue una tendencia creciente hasta alcanzar el primer valor fijado, estabilizándose momentáneamente y ante una reducción del setpoint, se observa un descenso indicando el proceso de enfriamiento.

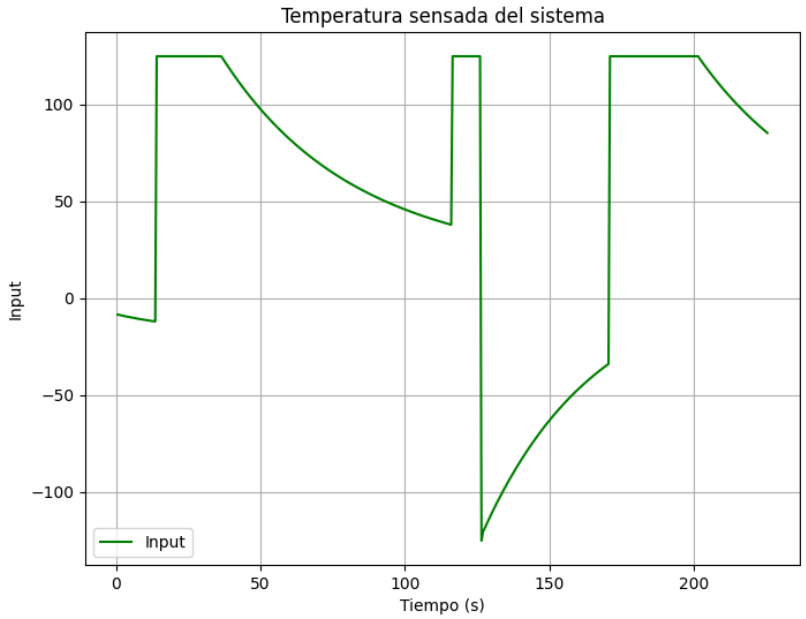

Figura 12: Gráfica de señal entrada.

4. Conclusiones y Recomendaciones

El laboratorio permitió implementar un sistema de control PID en Arduino logrando una regulación estable y eficiente de la temperatura, la integración de sensores, pantalla LCD y comunicación serial facilitó el monitoreo, se comprobó que la técnica PID es efectiva para procesos térmicos, destacando la importancia de una correcta sintonización de parámetros.

Referencias

[1] Arduino. (2024). *Arduino UNO R3: Product Reference Manual*. Recuperado de https://docs.arduino.cc/resources/datasheets/A000066-datasheet.pdf

[2] NXP Semiconductors. (1999). *PCD8544* 48×84 *pixels matrix LCD controller/driver*. Recuperado de: https://www.alldatasheet.com/datasheet-pdf/download/18170/PHILIPS/PCD8544.html.