Universidad Politécnica
de Madrid

**Escuela Técnica Superior de
Ingenieros Informáticos**

Diseño con Microcontroladores
Control de temperatura. Termostato

Manuel Calviño Lorente
Iván Clemente Álvarez del Rio

Madrid, diciembre 2024

Resumen

Este documento describe el diseño e implementación de un termostato que hemos desarrollado gracias a los conocimientos de la asignatura “Diseño con Microcontroladores” impartida por el departamento DATSI de la Escuela Técnica Superior de Ingenieros Informáticos, de la Universidad Politécnica de Madrid.

El sistema se basa en el microcontrolador Arduino Leonardo (ATMega32u4), que utiliza un controlador PID (Proporcional, Integral y Derivativo) para regular la temperatura con alta precisión. El sistema permite la medición y control de la temperatura mediante un sensor analógico (AD22100) y simula la inercia térmica utilizando un sensor de luminosidad (LDR).

El proyecto cuenta con una interfaz intuitiva, compuesta por un teclado analógico y un módulo LCD que permite la configuración y visualización de los modos de funcionamiento. Además, el sistema integra un Optotriac sincronizado con un detector de paso por ceso (ZCD) para controlar con precisión la energía suministrada al agente calefactor.

Tabla de contenidos

Resumen **2**
1 Introducción **1**
2 Desarrollo **2**
2.1 Descripción del Sistema y Componentes 2
2.2 Estructura del Software 4
2.3 Implementaciones de los Módulos Programados 5
3 Interfaz de usuario **13**
4 Conclusiones **15**

1 Introducción

El control de temperatura es un proceso fundamental en diversos entornos domésticos, industriales y educativos. Desde aplicaciones para sistemas de calefacciones hasta ventilación, aire acondicionado e incluso control de temperatura en infraestructuras críticas como CPDs (Centros de Procesamiento de Datos). La implementación de controladores automáticos como el control PID (Proporcional, Integral y Derivativo) juega un papel crucial en la eficiencia y estabilidad de estos sistemas.

En este contexto, el presente proyecto consiste en el diseño e implementación de un termostato basado en un microcontrolador Arduino Leonardo (ATMega32u4). Este sistema utiliza diversas entradas y salidas, como sensores, un módulo de visualización LCD, un teclado analógico, y un agente calefactor controlado mediante un Optotriac sincronizado con un detector de paso por cero (ZCD). Además, se ha implementado una interfaz de usuario sencilla para permitir el control manual y automático del sistema.

Propósito del Proyecto

El objetivo principal del proyecto se basa en el desarrollo de un sistema de control de temperatura de bajo coste y alta eficiencia. Además, este sistema debe permitir que se pueda:

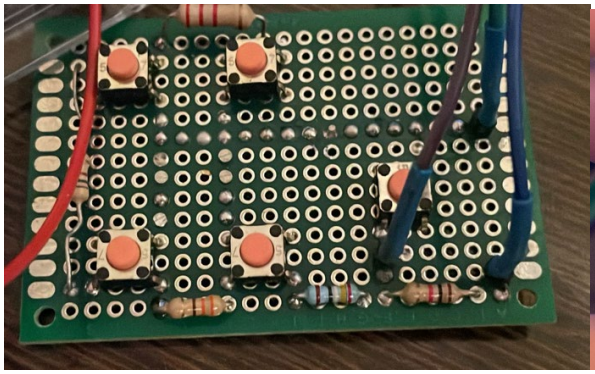
1. Medir y regular la temperatura de un sistema con alta resolución (1/4 de grado).
2. Controlar la energía suministrada a un agente calefactor en un rango del 0% al 100%.
3. Implementar un control PID para garantizar estabilidad en la temperatura objetivo con una tolerancia de ±1°C.
4. Simular un entorno dinámico mediante un sensor de luminosidad (LDR), dada la inercia térmica inherente de los sistemas reales.
5. Proporcionar una interfaz de usuario intuitiva, con un teclado analógico para configuraciones y un módulo LCD para la visualización de datos en tiempo real.
6. Permitir modos de funcionamiento manual y automático (Termostato) con opciones de control por potenciómetro.

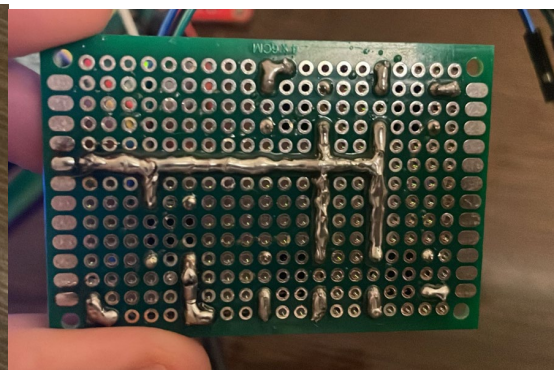
2 Desarrollo

2.1 Descripción del Sistema y Componentes

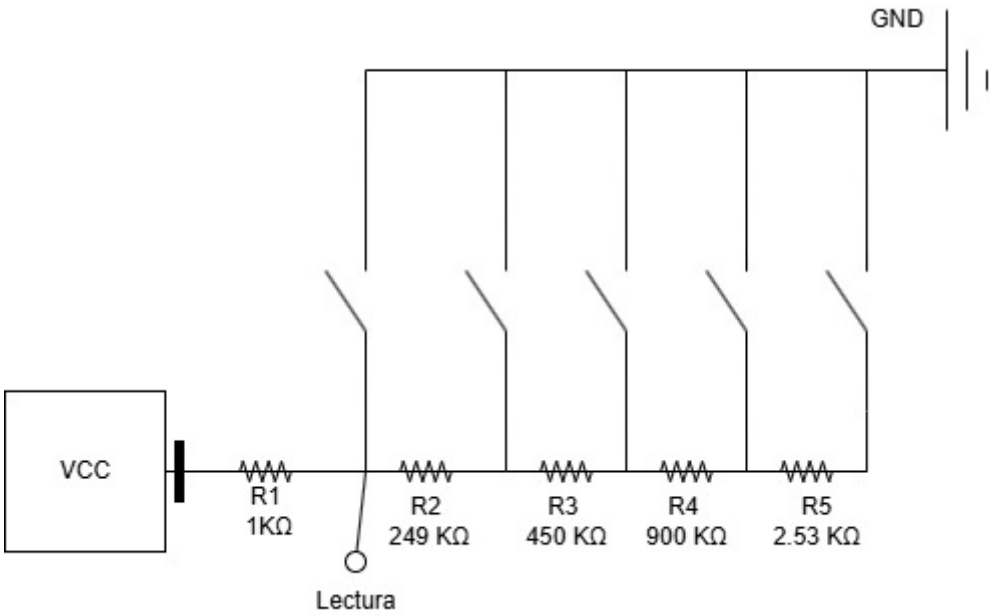
Componente	Pin Arduino	Descripción
Teclado analógico	**A1**	Entrada analógica (divisor de tensión).
Sensor de temperatura	**A2**	Entrada analógica del sensor AD22100.
Sensor de luminosidad	**A3**	Simula el comportamiento térmico.
Optotriac	**D9 (PB5)**	Controla el agente calefactor.
Detector de paso cero	**D7 (PORTE6)**	Entrada para sincronización ZCD.
LCD (HD44780U)	**D2-D5, D10-D12**	Control directo del módulo LCD:
	D12	RS (Definido en PORTD6).
	D11	RW (Definido en PORTB7).
	D10	Enable (Definido en PORTB6).
	D5	D4 (Definido en PORTC6).
	D4	D5 (Definido en PORTD4).
	D3	D6 (Definido en PORTD0).
	D2	D7 (Definido en PORTD1).

Además, parte del diseño del sistema consistió en el desarrollo de un teclado analógico usando pulsadores sueltos, una placa de desarrollo, resistencias, pines, cables y bandas retráctiles para realizar un divisor de tensión para poder realizar una lectura analógica de los pulsadores de manera precisa, casi sin rebotes.



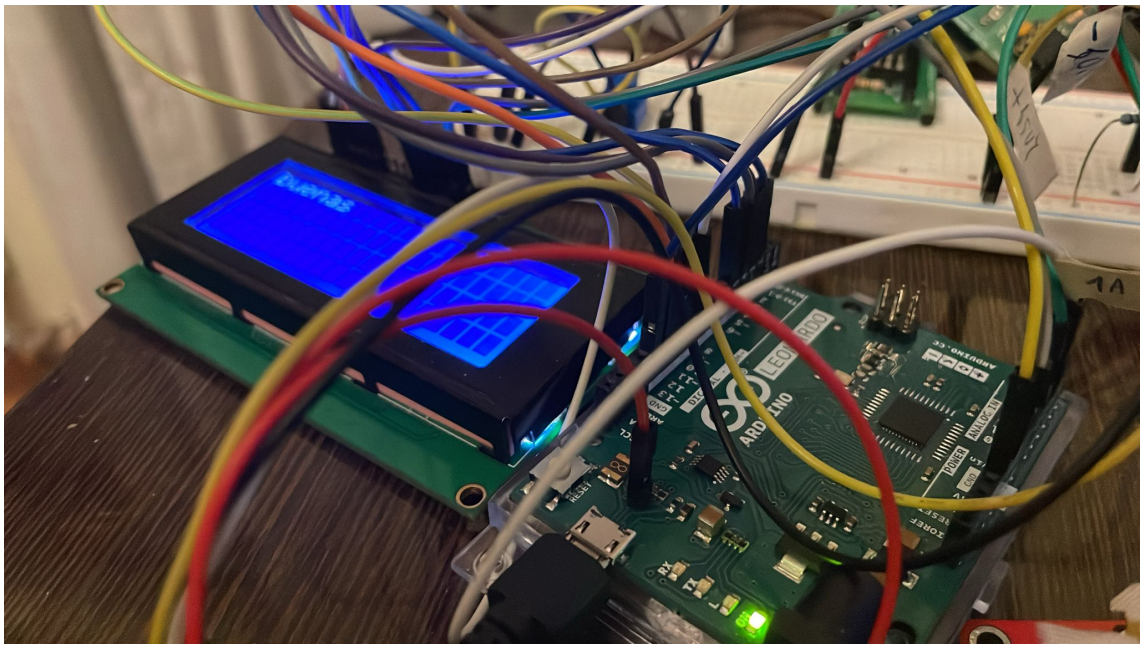


Como se puede observar se optó por una soldadura usando puentes de estaño con plomo 60/40 en el reverso de la placa.



(Esquema del divisor de tensión)

A continuación, se adjunta una foto del sistema montado con todos sus componentes.



2.2 Estructura del Software

La implementación del sistema está dividida en módulos independientes, cada uno responsable de una funcionalidad específica. A continuación, se presenta la estructura de archivos utilizada en el proyecto:

Archivo	**Descripción**
Lcd.ino	Control del módulo LCD, incluyendo visualización de datos y configuración.
UI.ino	Control del módulo LCD, incluyendo visualización de datos y configuración.
analog.ino	Conversión y manejo de señales analógicas mediante el ADC.
lightbulb.ino	Interfaz de usuario para gestionar modos y comandos del sistema.
sensors.ino	Lectura de sensores de temperatura y luminosidad.
teclado.ino	Gestión del teclado analógico para la entrada de datos.
termostato.ino	Lectura de sensores de temperatura y luminosidad.

El sistema funciona mediante la interacción coordinada de sus componentes y módulos programados. El microcontrolador Arduino Leonardo se encarga de leer los valores de los sensores de temperatura y luminosidad a través de sus entradas analógicas. Estos datos son procesados por el controlador PID para calcular la salida correspondiente, que se traduce en un ajuste de energía del calefactor controlado por el Optotriac.

El detector de paso por cero (ZCD) asegura la sincronización del control de fase del Optotriac con la red eléctrica, garantizando un ajuste preciso del flujo de energía al agente calefactor. A su vez, el teclado analógico permite al usuario introducir comandos y configurar los modos de funcionamiento, los cuales se visualizan en tiempo real a través del módulo LCD.

A continuación, procedemos a explicar la implementación a nivel de software del sistema.

2.3 Implementaciones de los Módulos Programados

En este capítulo se presentan las implementaciones de los módulos programados que conforman el sistema, destacando su integración y funcionalidad dentro del proyecto. Los módulos desarrollados incluyen Analog, Lightbulb, LCD, Teclado, Sensores, Termostato y UI, cada uno desempeñando un papel esencial en el funcionamiento del sistema. A continuación, se describen en mayor detalle Analog, Lightbulb y Termostato al ser el núcleo principal del sistema. El resto de los módulos se comentan por encima por simplicidad.

Módulo Analog

El módulo Analog es fundamental para la adquisición de datos analógicos provenientes de los sensores de temperatura y luminosidad. Este módulo configura el ADC (Convertidor Analógico-Digital) para convertir las señales analógicas en valores digitales procesables por el microcontrolador.

La configuración inicial del ADC se realiza mediante la función configureADC:

```
// Configuración inicial del ADC
void configureADC(uint8_t adcPin) {
    sbi(ADCSRA, ADEN); // Habilitar el ADC
    sbi(ADCSRA, ADPS2); // Prescaler de /128 al activar ADPS2..ADPS0
    sbi(ADCSRA, ADPS1);
    sbi(ADCSRA, ADPS0);
    sbi(ADCSRA, ADIE); // Habilitar interrupciones del ADC
    // Configurar el pin ADC
    ADMUX = (ADMUX & 0xE0) | (adcPin & 0b011111); // Configurar MUX4..MUX0
    ADCSRB = (ADCSRB & 0xDF) | ((adcPin & 0b100000)); // Configura MUX5 | Primero aplicamos mascara (bit 5 a '0') y luego aplicamos el valor de adcPin
    // Seleccionar referencia de voltaje como Vcc
    cbi(ADMUX, REFS1);
    sbi(ADMUX, REFS0);
}
```

Esta función habilita el ADC, ajusta el prescaler a 128 para garantizar una frecuencia adecuada de operación (125 kHz) y configura el pin analógico deseado.

El módulo Analog utiliza un temporizador configurado en modo CTC para disparar conversiones periódicas:

```
void configureTimer() {
    cli(); // Deshabilitar interrupciones globales
    sbi(TCCR1B, CS12); // Prescaler de /1024
    cbi(TCCR1B, CS11);
    sbi(TCCR1B, CS10);
    cbi(TCCR1B, WGM13); // Modo CTC
    sbi(TCCR1B, WGM12);
    OCR1A = 156; // Valor TOP para 10 milisegundos
```

```
    sbi(TIMSK1, OCIE1A); // Habilitar interrupciones en el canal A
    sei(); // Habilitar interrupciones globales
}
```

Cuando el temporizador alcanza el valor de comparación definido, inicia una conversión automáticamente a través de la interrupción del Timer1:

```
ISR(TIMER1_COMPA_vect) {
    if (!bit_is_set(ADCSRA, ADSC)) {
        sbi(ADCSRA, ADSC); // Iniciar conversión
    }
}
```

Los resultados de las conversiones se manejan mediante la interrupción del ADC:

```
ISR(ADC_vect) {
    conversion = ADC;       // Guardar el resultado
    is_converted = true;    // Señalar que la conversión está lista
}
```

El módulo incluye funciones auxiliares para verificar el estado de las conversiones y obtener resultados:

```
bool isConversionComplete() {
    cli();
    bool value = is_converted;
    sei();
    return value;
}
int16_t getConversionResult() {
    cli();
    int16_t value = conversion;
    is_converted = false; // Reiniciar la bandera
    sei();
    return value;
}
```

Además, la función customAnalogRead gestiona la lectura de datos de un pin específico, combinando configuraciones del ADC, verificación de estado y almacenamiento de resultados:

```
int16_t customAnalogRead(uint8_t adcPin) {
    if (canConvert && !isConvertionReady(adcPin)) {
        configureADC(adcPin); // Configurar el ADC para el pin deseado
        canConvert = false;
        putConverting(adcPin);
    }
    if (isConversionComplete() && isConvertionReady(adcPin)) {
        putConverting(adcPin);
        canConvert = true;
```

```
        if (isConvertionGood(adcPin)) // Desechar la primera conversión
            setConvertion(adcPin, getConversionResult());
        else
            return customAnalogRead(adcPin);
    }
    return getConvertion(adcPin);
}
```

Módulo Lightbulb

El módulo Lightbulb es responsable de controlar la potencia entregada al calefactor mediante el uso de un Optotriac sincronizado con un detector de cruce por cero (ZCD). Este método permite ajustar con precisión el porcentaje de energía entregada, asegurando eficiencia y reduciendo interferencias.

La configuración inicial del sistema incluye la preparación de pines y la inicialización del temporizador:

```
void lightbulbSetup() {
  CLEAR_BIT(DDRE, DDE6);  // Configurar INT6 como entrada
  CLEAR_BIT(PORTE, PORTE6); // Sin pull-up
  // Configurar interrupción INT6 para detección de cruce por cero
  EICRB &= ~((1 << ISC61) | (1 << ISC60)); // Flanco ascendente
  EICRB |= (1 << ISC60);
  EIFR |= (1 << INTF6);   // Limpiar bandera de interrupción
  EIMSK |= (1 << INT6);   // Habilitar interrupción INT6
  SET_BIT(DDRB, DDB5);    // Configurar PB5 como salida
  SET_BIT(PORTB, PB5);    // Inicialmente en HIGH
  setup_timer3();
}
```

Cuando se detecta un cruce por cero, la interrupción INT6 reinicia el temporizador:

```
ISR(INT6_vect) {
    TCNT3 = 0; // Reiniciar contador
    TCCR3B |= (1 << CS31) | (1 << CS30); // Iniciar temporizador con prescaler 64
}
```

El temporizador genera una interrupción cuando alcanza el valor definido, activando el Optotriac para permitir el paso de corriente:

```
ISR(TIMER3_COMPA_vect) {
    CLEAR_BIT(PORTB, PB5);   // Activar optotriac
    _delay_us(30);           // Pulso corto al LED del optotriac
    SET_BIT(PORTB, PB5);     // Desactivar optotriac

    TCCR3B &= ~((1 << CS32) | (1 << CS31) | (1 << CS30)); // Detener temporizador
}
```

La configuración del temporizador se realiza para operar en modo CTC (Clear Timer on Compare Match):

```
void setup_timer3() {
    TCCR3A = 0;                          // Temporizador en modo normal
    TCCR3B = (1 << WGM32);               // Modo CTC
    TIMSK3 = (1 << OCIE3A);              // Habilitar interrupción por comparación
}
```

El brillo deseado se ajusta mediante la función setLightPercent, que calcula el retardo necesario basado en el porcentaje de energía requerido:

```
void setLightPercent(double f) {
    if (f < 0.0) f = 0.0;
    if (f > 0.9) f = 0.9;

    double adjustedF = f; // Relación directa para brillo proporcional

    double delayMs = (acos(1.0 - 2.0 * (1.0 - adjustedF)) / pi) * 10.0;
    delayMs += 0.3; // Ajustar desfase de corriente

    OCR3A = (uint16_t)(delayMs * 250); // Convertir milisegundos a ticks
}
```

Esta función utiliza una relación trigonométrica para determinar el tiempo de activación del Optotriac, ajustando el ángulo de fase y garantizando un control eficiente de la potencia.

El diseño del módulo Lightbulb asegura una sincronización precisa con la red eléctrica y proporciona flexibilidad en el ajuste de la potencia del calefactor.

Módulo LCD

El módulo LCD implementa todas las funcionalidades necesarias para controlar un módulo de visualización LCD en modo de 4 bits, siguiendo las especificaciones del controlador HD44780. Además, este módulo incluye una función específica para gestionar de manera dinámica la información que se muestra en la pantalla, en función de un modo de operación definido por una variable global. Esta variable permite adaptar el contenido mostrado al contexto actual del sistema, ya sea la visualización de la temperatura, el control de intensidad lumínica, o configuraciones específicas.

La función de inicialización, lcd_init, es esencial para preparar el LCD en modo de 4 bits. Durante su ejecución, se configuran los pines necesarios como salidas, se define el modo de operación y se realizan comandos iniciales específicos, como habilitar dos líneas de texto y ajustar el formato de los caracteres a una matriz de 5x8 píxeles. Esta configuración asegura que el LCD esté listo para recibir comandos y datos.

Cuando se enciende el módulo LCD, su estado interno es indeterminado. Esto ocurre porque el controlador puede estar en un estado de encendido inicial (reset por hardware) o en un estado previo si no se ha reiniciado adecuadamente. En este punto, el LCD no puede interpretar de forma confiable los comandos enviados por el microcontrolador.

```
cd_write4bits(0x03);  // Enviar 0x03 tres veces para inicializar
```

```
  delay(5);
  lcd_write4bits(0x03);  // Repetir el comando de inicialización
  delay(5);
  lcd_write4bits(0x03);
  delay(5);
```

El controlador del LCD asume que está en modo de 8 bits tras el encendido. Por lo tanto, la secuencia de inicialización debe comenzar asegurándose de que el controlador está sincronizado con el microcontrolador.

La especificación del controlador HD44780 requiere que se envíe un comando de inicialización (0x03) tres veces al LCD. Este proceso sirve para garantizar que el controlador esté completamente preparado y sincronizado para aceptar comandos en modo de 4 bits.

Además, se ha implementado un conjunto de operaciones muy extenso para usar el módulo LCD, los cuales se podrán encontrar implementados y comentados en el codigo anexado a la memoria. Las operaciones son: void lcd_init();

```
void lcd_command(uint8_t cmd);
void lcd_write(uint8_t data);
void lcd_setCursor(uint8_t col, uint8_t row);
void lcd_print(const char *str);
void lcd_write4bits(uint8_t nibble);
void lcd_clear();
void lcd_home();
void lcd_displayOn();
void lcd_displayOff();
void lcd_cursorOn();
void lcd_cursorOff();
void lcd_blinkOn();
void lcd_blinkOff();
void lcd_scrollDisplayLeft();
void lcd_scrollDisplayRight();
void lcd_createChar(uint8_t location, uint8_t charmap[]);
void pulseEnable();
bool lcd_checkIfBusy();
void lcd_actualizarDisplay();
```

Módulo Teclado

El módulo del teclado implementa la gestión de entradas mediante un teclado analógico, permitiendo interpretar pulsaciones a través de un divisor de tensión. Para lograr esto, el sistema convierte los valores analógicos leídos en el pin

correspondiente en teclas discretas, asignando cada rango de voltaje a un botón específico. Esta conversión se realiza mediante la función leerTecladoFiltrado, que verifica de forma iterativa el rango del valor analógico y utiliza un contador de muestras consecutivas para filtrar rebotes y asegurar la detección precisa de la pulsación.

El diseño modular del código permite que cada pulsación activa eventos específicos según el modo de operación actual, como navegación en menús, ajustes de configuración o cambios de parámetros. La implementación de un filtro basado en conteo asegura una mayor fiabilidad en la detección de entradas, reduciendo la posibilidad de errores debidos a fluctuaciones o rebotes en el sistema. Este enfoque, junto con la integración de un potenciómetro para ajustes analógicos adicionales, proporciona una interfaz robusta y eficiente para interactuar con el sistema.

Módulo Sensores

Este módulo está compuesto de dos métodos. Uno está pensado para leer la intensidad luminica y devuelve una lectura entre 0 y 1023. El segundo está pensado para calcular la temperatura dentro de un rango en grados centigrados. Este rango está definido en el fichero sensors.h.

```
int getBrightness(){
  return customAnalogRead(SENSOR_LUZ);
}
float getTemperature(){//TODO comprobar si la comberison es correcta
  float T = customAnalogRead(SENSOR_TEMPERATURA);
  return T * (MAX_TEMPERATURE- MIN_TEMPERATURE)/1024;
}
```

Módulo UI

El módulo UI gestiona la interfaz de usuario del sistema, integrando la interacción entre el teclado, el display LCD y los modos de operación. La función principal de este módulo es lcd_actualizarDisplay, que limpia y actualiza la pantalla en función del estado actual del sistema.

Dependiendo del valor de la variable global modo_operacion, la función muestra información contextual, como la temperatura, los modos disponibles, o el estado del sistema. Por ejemplo, en el modo apagado, se muestra "Apagado" junto con la temperatura actual, mientras que en el modo principal se presentan las opciones de configuración y control disponibles.

Este módulo asegura que la información presentada al usuario sea clara y relevante, facilitando la navegación y el control del sistema.

Módulo Termostato

El módulo Termostato es el componente principal que integra y coordina los distintos subsistemas para ofrecer funcionalidades avanzadas como el control de temperatura y luminosidad. Este módulo utiliza dos controladores PID (Proporcional-Integral-Derivativo) configurados para regular el sistema con precisión.

La configuración inicial del módulo se realiza en la función setup, donde se inicializan los subsistemas clave y los controladores PID:

```
void setup() {
  Serial.begin(115200);
  while(!Serial){}
  lightbulbSetup();           // Configurar el sistema
  lcd_init();                 // Iniciar el LCD
  lcd_clear();                // Limpiar pantalla
  lcd_setCursor(0, 0);        // Configurar cursor
  configureADC(PIN_TECLADO);  // Configurar ADC
  configureTimer();           // Configurar temporizador
  // Configura los PID
  tPID.SetMode(AUTOMATIC);
  tPID.SetOutputLimits(0, MAX_TEMPERATURE);
  tPID.SetSampleTime(200);
  lPID.SetMode(AUTOMATIC);
  lPID.SetOutputLimits(0, MAX_LIGHT);
  lPID.SetSampleTime(100);
}
```

Los controladores PID ajustan los valores de salida en función de las lecturas de los sensores y los puntos de ajuste definidos por el usuario. La lógica principal del sistema está implementada en la función loop, que gestiona diferentes modos de operación:

```
void loop() {
  switch(modo_operacion){
    case MODO_TERMOSTATO:
      if(config == CONFIG_POTENCIOMETRO) Setpoint = readPotenciometer()/MAX_TEMPERATURE
+ DEFAULT_TEMPERATURE;
      Input = getTemperature();
      tPID.Compute();
      setLightPercent(Output/MAX_TEMPERATURE);
      break;
    case MODO_INTENSIDAD_LUZ:
      if(config == CONFIG_POTENCIOMETRO) Setpoint = readPotenciometer();
      Input = getBrightness();
      lPID.Compute();
      setLightPercent(Output/MAX_LIGHT);
      break;
```

```
    case MODO_MANUAL:
      if(config == CONFIG_POTENCIOMETRO) vertical = (readPotenciometer()*100)/1024;
      setLightPercent((float)(vertical%101)/100.0);
      break;
    case MODO_ENCENDIDO:
      setLightPercent(1);
      break;
    case MODO_APAGADO:
      setLightPercent(0);
      Input = getTemperature();
      break;
  }
  manejarTeclado();
  if(t + 250 < millis()){
    lcd_actualizarDisplay();
    t = millis();
  }
  Serial.print(Input);
  Serial.print(",");
  Serial.print(Output);
  Serial.print(",");
  Serial.println(Setpoint);
}
```

El módulo soporta cinco modos de operación:

1. **Modo Termostato**: Ajusta la temperatura deseada y regula la potencia del calefactor para mantenerla.
2. **Modo Intensidad de Luz**: Controla la intensidad luminosa ajustando el setpoint de luminosidad deseada.
3. **Modo Manual**: Permite al usuario seleccionar manualmente el porcentaje de potencia.
4. **Modo Encendido**: Activa el sistema al 100% de potencia.
5. **Modo Apagado**: Desactiva el sistema.

La integración de sensores, controladores PID y subsistemas asegura un funcionamiento robusto y eficiente, adaptándose a las necesidades del usuario en tiempo real.

3 Interfaz de usuario

La interfaz de usuario está compuesta por una serie de menús que incluyen opciones para acceder a los diferentes modos de uso. Para navegar por los menús, el usuario cuenta con un mando de cinco botones: uno para apagar la bombilla, dos para desplazarse arriba o abajo por los menús o para ajustar la intensidad de la bombilla, uno para aceptar o seleccionar, y otro para regresar al menú principal o a la pantalla de inicio (home).

Figura 3.1 menú home

La primera opción del menú principal lleva a un submenú que permite elegir entre regular la temperatura o la intensidad de la luz.

Figura 3.2 submenú Temperatura o luz

Los modos de temperatura y luz operan de manera muy similar, con una interfaz que se adapta para mostrar las unidades específicas según el contexto seleccionado:

- Modo Termostato (Temperatura): Muestra la temperatura actual (por ejemplo, 63.03) y un valor objetivo configurable (por ejemplo, 20.00).
- Modo Intensidad de Luz: Indica la intensidad actual de la luz (por ejemplo, 72.00) y un objetivo ajustable según las necesidades del usuario.

Figura 3.3 modo Temperatura

Figura 3.4 modo Intensidad de la luz

La segunda opción del menú principal permite encender la bombilla al máximo nivel posible configurado por el sistema. Este modo también puede ser utilizado como herramienta de diagnóstico para verificar si la bombilla funciona correctamente o si el sistema presenta fallas.

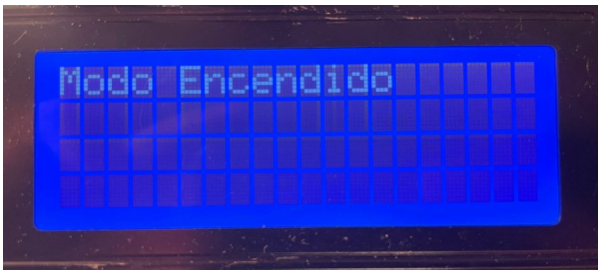


Figura 3.5 modo encendido

El modo de configuración ofrece la posibilidad de alternar entre el uso de teclas o un potenciómetro para ajustar los valores objetivo en los modos de temperatura, luz y el modo manual. Esto proporciona mayor flexibilidad para personalizar el funcionamiento del sistema según las preferencias del usuario.

Figura 3.6 modo configuración

La última opción del menú principal es el modo manual, que permite al usuario ajustar el porcentaje de encendido de la bombilla de manera precisa. Este modo opera de manera similar a un sistema *dimmer,* ofreciendo un control más detallado sobre la intensidad de la iluminación.

Figura 3.7 Modo manual

4 Conclusiones

El desarrollo de este proyecto ha permitido implementar un sistema de control de temperatura eficiente y funcional gracias a los conocimientos que se nos ha transmitido en la asignatura de Diseño con Microcontroladores.

La integración de las distintas funcionalidades ha permitido abordar el control de la temperatura, garantizando una estabilidad adecuada.

A través del diseño modular y la implementación de los diferentes algoritmos, hemos logrado afianzar el conocimiento transmitido a lo largo de la asignatura sobre tareas relacionadas con E/S en microcontroladores, así como otras tareas de sumo interés como la gestión de timers con microcontroladores, profundizando en conceptos fundamentales y adquiriendo un mayor entendimiento sobre la modulación por ancho de pulso (PWM) entre otras cosas.

Este proyecto ha supuesto un reto académico y técnico que ha fomentado nuestra capacidad de análisis, diseño y resolución de problemas, permitiéndonos comprender de manera práctica los principios de interacción entre hardware y software en un entorno de control embebido. Asimismo, el trabajo realizado ha proporcionado una visión más completa sobre la importancia de la planificación y la estructura en el desarrollo de sistemas basados en microcontroladores.

En este sentido, este proyecto ha fomentado y generado mayor interés en este campo de la Informática y la Electrónica abriéndonos nuevas oportunidades de conocimiento e interés.