E.T.S. de Ingeniería Industrial,
Informática y de Telecomunicación

Desarrollo de un Sistema de Control de Temperatura para su uso en un Biorreactor

Grado en Ingeniería Eléctrica y Electrónica

Trabajo Fin de Grado

Javier Huarte Larrayoz

Santiago Tainta Ausejo

Pamplona, 6 de septiembre de 2023

upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

Resumen

Este trabajo, desarrollado durante unas prácticas curriculares en la empresa Celignis Analytical, consiste en el desarrollo de un sistema para el control de la temperatura en el interior de un biorreactor. El sistema contará con un sensor de temperatura para su medida y un calefactor de silicona para su regulación. El procesado de la información estará basado en el uso de un microcontrolador Arduino Uno, que además contará con un sistema de interfaz a través de una pantalla LCD. Esta pantalla permitirá tanto la visualización de información en tiempo real como la programación de experimentos, pudiendo fijar intervalos temporales con diferentes temperaturas de acuerdo con los requisitos de este.

Palabras clave

Control PID; Biorreactor; Arduino.

Abstract

This project, developed during an internship in the company Celignis Analytical, aims to develop a temperature control system for a bioreactor. The system will include a temperature sensor and a heating jacket to precisely regulate the bioreactor's internal temperature. To process the data and provide a user-friendly interface, an Arduino Uno microcontroller will be employed, along with an LCD display. This display will enable real-time visualization of the data and facilitate the programming of various tests. Users will be able to set the test duration and desired temperature according to their requirements.

Keywords

PID Control; Bioreactor; Arduino.

Índice

Índice de figuras

Índice de tablas

1. Introducción

1.1. Contexto del Proyecto

Este trabajo fin de grado ha sido realizado durante la participación en el programa de movilidad Erasmus+. En dicho programa se han realizado prácticas curriculares con una duración de cuatro meses en Celignis Analytical, una empresa ubicada en Limerick, una de las principales ciudades de la República de Irlanda.

La empresa se trata de un laboratorio de análisis e investigación relacionado con la biomasa: para su oferta de análisis, recibe muestras de biomasa desde empresas u organizaciones que no disponen de los medios o conocimientos necesarios para realizar un análisis en profundidad, lo que permite a Celignis realizar un estudio en el que determinan la composición de la muestra, sus propiedades y sus mejores aplicaciones. Cuenta con una gran variedad de ofertas de análisis, desde muestras destinadas a ser utilizadas como biocombustible, hasta el análisis de su capacidad para ser empleadas en digestión anaerobia o como biochar [1]. Además de sus ofertas de análisis, Celignis también forma parte de múltiples proyectos de investigación dentro del marco de desarrollo sostenible de la Unión Europea, ya sean proyectos propios o realizados en colaboración con otras empresas.

Como parte de sus funciones en la empresa, Celignis encargó al estudiante el desarrollo e implementación de un sistema capaz de controlar la temperatura en el interior de un biorreactor, empleando un calefactor de silicona y un sensor PT-100 que se encontraban en desuso.

Figura 1: Logotipo de Celignis

1.2. Objetivos

El objetivo de este proyecto es el **desarrollo de un sistema de control de temperatura para el interior de un biorreactor**. Para ello, se implementará un **lazo de control con un algoritmo PID** que regulará un sistema de actuación basado en un calefactor y ajustará su respuesta en función de la información recibida a través de un sensor de temperatura, siendo el objetivo principal conseguir un valor de temperatura estable lo más cercano posible al valor deseado y sin sobrepasar el valor marcado. Además, el sistema deberá contar con una **interfaz que permita al usuario programar la duración y temperatura** deseadas para un máximo de cinco etapas distintas.

1.3. Metodología

Dado que este trabajo ha sido realizado durante la participación en unas prácticas curriculares, el primer paso dado por el estudiante fue reunirse con su supervisor en la empresa para tratar los distintos proyectos posibles que podría realizar durante este periodo. Una vez planteada la idea del proyecto, se realizó un estudio de su viabilidad y un análisis de las necesidades de la empresa para el prototipo.

Durante las semanas siguientes, se realizó un estudio de los componentes necesarios para el diseño del prototipo, barajando diferentes opciones tanto para el sistema de medida como para el sistema de actuación. Tras decidir el hardware necesario, se realizó una reunión para la selección final los elementos a emplear, que posteriormente fueron adquiridos por la empresa. Una vez obtenidos los componentes, se realizó el montaje del prototipo y su programación, a continuación, comenzaron las pruebas de sintonía, empleando un biorreactor de 12 litros relleno de agua. Los progresos de estas pruebas se fueron comunicando en reuniones semanales con el supervisor de la empresa, que revisaba su funcionamiento y proponía algunas ideas para su mejora. Finalmente, el prototipo desarrollado fue evaluado por los empleados de la empresa para la validación de su correcto funcionamiento.

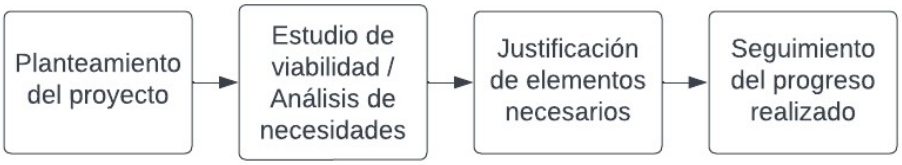

Figura 2: Etapas de la realización del proyecto en la empresa

1.4. Estructura del Documento

En este primer capítulo, se realiza una breve contextualización del trabajo de fin de grado, sus objetivos y la metodología seguida durante su desarrollo. Además, se presenta la estructura de esta memoria para proporcionar una visión general del contenido que se abordará a continuación.

En el capítulo 2, se describen los principales elementos de los que disponía la empresa de manera previa al diseño del prototipo: el biorreactor y el sistema de calefacción. También se mencionan las aplicaciones previstas para este sistema, lo que aporta una visión más completa del contexto en el que se desarrolló el proyecto.

El capítulo 3 enfoca los diferentes elementos que necesitará el prototipo y sus funciones, incluyendo el lazo de control. Este tema se profundiza en el capítulo 4, donde se presentan detalladamente los componentes seleccionados para realizar estas funciones y su funcionamiento. Todo lo relacionado con el software del sistema se presenta en el capítulo 5, donde se abordan aspectos como el entorno de desarrollo empleado, las librerías de Arduino utilizadas y sus funciones, así como la estructura del código realizado y su funcionamiento.

En el capítulo 6, se trata el montaje final del prototipo, tanto de su diseño mecánico como las pruebas de sintonía realizadas para obtener los parámetros del controlador PID. También se incluyen las pruebas llevadas a cabo por la empresa para evaluar su funcionamiento en un caso

real. Por último, el capítulo 7 consiste en un análisis de la satisfacción del resultado final del trabajo de fin de grado y se proponen mejoras potenciales que podrían ser implementadas en el futuro para optimizar su funcionamiento.

En cuanto a los anexos incluidos en el documento, el primero consiste en una tabla que contiene el presupuesto del prototipo, en la que se encuentran detallados los precios de cada uno de los componentes utilizados, así como el coste real del sistema para la empresa, ya que contaba con múltiples de los elementos necesarios de antemano. A continuación, el Anexo B incluye la totalidad del código implementado. Finalmente, el Anexo C incluye una breve guía dirigida al usuario final del dispositivo con instrucciones sobre como configurar el programa y la correcta conexión de los elementos, en caso de que sea necesario realizar alguna tarea de mantenimiento.

2. Antecedentes

2.1. Biorreactor

Un biorreactor es un recipiente o sistema diseñado para mantener un ambiente biológicamente activo en unas condiciones controladas (temperatura, presión, oxigenación, PH…) para su uso en diversos procesos de biotecnología (ya sean aerobios o anaerobios) en los que se busca el crecimiento de microorganismos / células vegetales o animales. Tienen la capacidad de transformar materias primas en productos mediante la multiplicación de células y/o microorganismos como hongos, levaduras y bacterias. [2]

El modelo usado en Celignis es el reactor de tanque agitado continuo (CSTR), en el que se emplea un motor para agitar los contenidos del tanque durante la duración del proceso. La tapa se cierra herméticamente, lo que permite generar un ambiente aislado en el interior del biorreactor. Para poder medir la temperatura en el interior cuenta con una apertura en la que se puede introducir un sensor como el Pt-100, manteniéndolo aislado del interior del tanque (*Sample Pipe* en la Figura 3). [3]

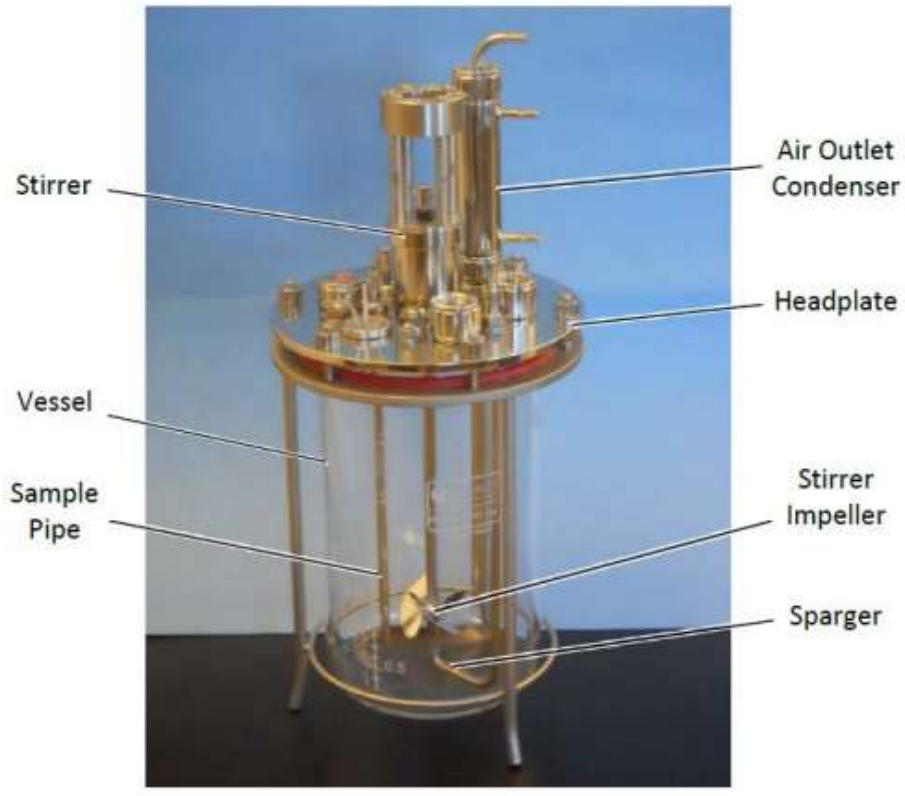

Figura 3: Fotografía de un biorreactor con sus partes principales [3]

2.2. Sistema de Calefacción

Mantener una temperatura interna estable es de vital importancia para poder conseguir un correcto funcionamiento del biorreactor. Para ello, dos de los métodos más comunes empleados para aportar calor al tanque son los baños termostáticos y las mantas calefactoras de silicona. Como puede verse a la derecha en la Figura 4, el primer método consiste en sumergir el biorreactor en un baño de agua a la temperatura deseada, en este caso, solo se controla la temperatura del baño termostático, por lo que podría se podría llegar producir un error en la temperatura interna del biorreactor. En el segundo método, tal y como se puede ver a la izquierda, el calefactor se coloca directamente en contacto con el biorreactor. La temperatura interna del biorreactor es monitorizada mediante un sensor con el cual se controla la potencia del calefactor, para así asegurar que la temperatura interna es la deseada.

En el contexto de este proyecto, se ha empleado el segundo método, usándose una manta calefactora con una potencia máxima de 200 W. La desventaja del modelo de calefactor utilizado es que es incapaz de regular su consumo de potencia, ya que únicamente cuenta con un enchufe para ser alimentada a una tensión de 110 V_{AC}. Por ello, será necesario implementar un sistema de actuación con el que regular su funcionamiento de modo que se obtenga la temperatura deseada en el interior del tanque.

Figura 4: Configuración de un biorreactor con manta calefactora (izquierda) y baño termostático (derecha)

2.3. Aplicación del Sistema

Celignis emplea biorreactores en diversos tipos de análisis y proyectos de investigación. En el caso de este prototipo, el biorreactor cuya temperatura se desea controlar será utilizado para llevar a cabo procesos de extracción de xilano a partir de materias primas de origen vegetal. Esta tarea se enmarca en el proyecto de investigación *EnXylaScope* [4], el cual es un proyecto diseñado e ideado por Celignis en colaboración con una docena de socios adicionales y un presupuesto de más de seis millones de euros. *EnXylaScope* forma parte del programa "*Horizon*

Europe", una iniciativa de la Unión Europea que busca financiar proyectos de desarrollo sostenible con un presupuesto de 95.500 millones de euros.

El objetivo del proyecto es el desarrollo de productos para el consumidor libres de plásticos líquidos derivados de combustibles fósiles gracias al descubrimiento de nuevas enzimas que actúan sobre la cadena lateral del xilano, las cuales se pretenden aplicar para desarrollar nuevos productos a base de xilano. Celignis se encarga de la extracción de xilano soluble en agua para convertirlo en formas insolubles en agua con alta capacidad de retención de agua mediante las enzimas desarrolladas en el proyecto, con el objetivo de emplear el xilano modificado como alternativa a los plásticos líquidos que se utilizan en productos cosméticos y de cuidado personal.

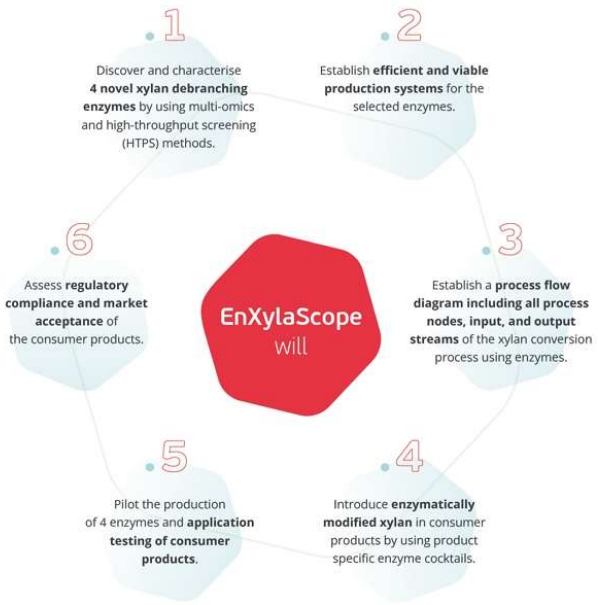

Figura 5: Objetivos de EnXylaScope [4]

Además de para la extracción de xilano, Celignis utiliza sus biorreactores para realizar pruebas de digestión anaeróbica [5], un proceso muy prometedor para la gestión de residuos orgánicos, ya que permite reducir el volumen de residuos a la vez que se generan biogases como el metano, que pueden ser posteriormente utilizados como combustibles renovables.

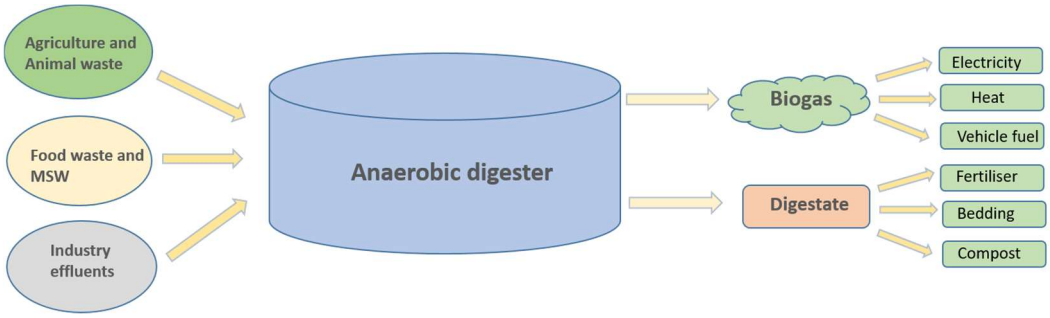

Figura 6: Proceso de digestión anaerobia [5]

Este proceso es utilizado en su oferta de análisis del potencial de biometano (BMP) de una muestra. El BMP está considerado como la máxima cantidad de biometano que se produce a

partir de la materia prima. Para determinar este potencial se realiza una prueba de laboratorio en el que se mezcla sustrato orgánico con un inóculo anaerobio en un biorreactor incubado a una temperatura fija durante un periodo de tiempo determinado. El volumen de biogás producido durante este proceso es monitorizado y posteriormente se analiza su composición con el fin de determinar su BMP.

3. Estructura General del Sistema

3.1. Introducción

En esta sección se va a presentar la estructura general del sistema. En la Figura 7 está representado un esquema general de los bloques que van a conformar el mismo. Como se puede observar, existen tres bloques diferentes conectados al microcontrolador:

- Bloque de medida y acondicionamiento de temperatura: empleando un sensor de temperatura, mide y acondiciona la señal para enviarla al microcontrolador a través de un bus de comunicación.
- Bloque de actuación: a partir de una salida del microcontrolador regula la potencia de un calefactor para obtener la temperatura deseada en el biorreactor.
- Interfaz: permite al usuario interactuar con el sistema a través de una pantalla y una serie de botones.

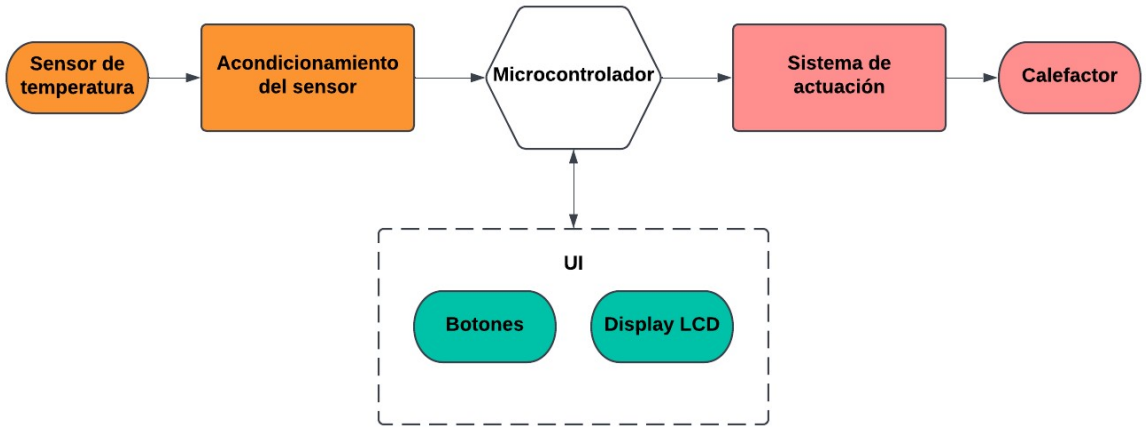

Figura 7: Esquema del Sistema

3.2. Medida y acondicionamiento de temperatura

Este bloque se compone del sensor de temperatura ubicado en el interior del biorreactor y su circuito de acondicionamiento. Su función es enviar al microcontrolador información precisa y constante sobre la temperatura en el interior de este, de modo que el lazo de control pueda realizar los cambios necesarios en el sistema de actuación. Se busca que el sensor cuente con las siguientes características:

- Envío de información en tiempo real al microcontrolador empleando un bus digital.
- Precisión de al menos 1 ℃.
- Resolución de al menos 0,1 ℃.

3.3. Actuación

Una vez se ha establecido la temperatura deseada para el programa, el calefactor debe mantener la temperatura interior del biorreactor a ese nivel. Para ello, es necesario emplear un sistema de actuación que regule la temperatura del calefactor controlando una señal de potencia en AC. Se han evaluado las siguientes opciones:

- Control ON / OFF
    - Relé
    - Optoacoplador
- Control de fase AC
    - TRIAC

3.3.1. Control ON / OFF

Este método de control es el más sencillo y extendido, pero presenta claras desventajas en aplicaciones que requieren precisión, ya que únicamente se controla el encendido y apagado del calefactor, sin poder regular su potencia. Por ello, la respuesta nunca se estabiliza a un valor preciso, oscilando constantemente entre los rangos de histéresis fijados, como se observa en la Figura 8. Los elementos que se han estudiado para este tipo de control son el relé y el optoacoplador, ya que ambos son capaces proporcionar aislamiento galvánico entre el circuito DC del microcontrolador y el circuito AC utilizado por el calefactor.

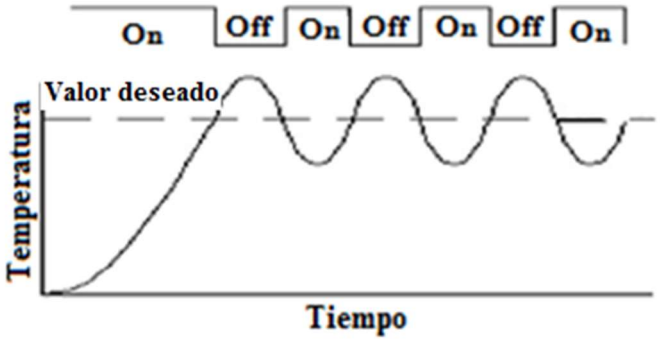

Figura 8: Respuesta del control ON / OFF

- **Relé**

Se trata de un interruptor accionado por un electroimán de forma que, cuando la bobina recibe suficiente corriente eléctrica, crea un campo magnético que convierte el núcleo en un imán. Este imán atrae hacia si una armadura que empuja los contactos del interruptor, cerrando así el circuito (o abriéndolo, en el caso de que se deseen emplear los contactos NC). En la Figura 9 se observan los principales elementos de un relé.

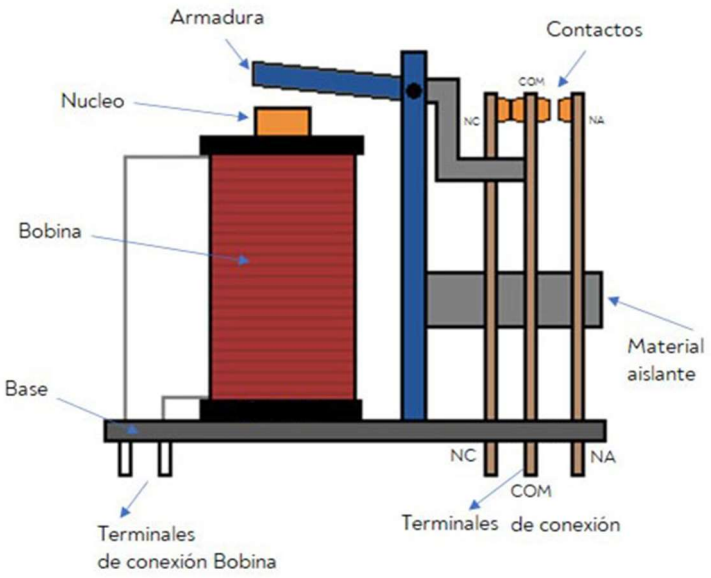

Figura 9: Modelo de un relé

- **Optoacoplador**

Su funcionamiento es muy similar al relé, la principal diferencia es que, en vez de un imán, emplea un diodo LED para saturar un dispositivo optoelectrónico (normalmente un fototransistor o un fototriac). Como se puede observar en la Figura 10, su principal ventaja es el aislamiento entre los circuitos de control y los de potencia, ya que no están directamente en contacto, además de no estar sometido a desgaste mecánico como los relés. Es el fotodetector el que actúa como interruptor, cerrando el circuito cuando el diodo LED lo satura.

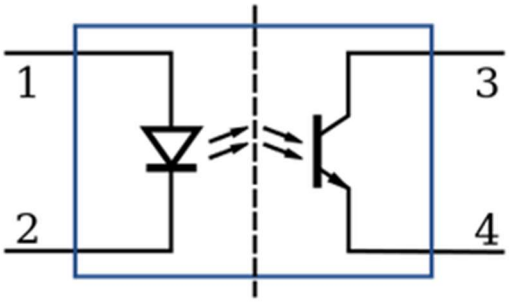

Figura 10: Modelo del optoacoplador

3.3.2. Control de fase AC mediante TRIAC

El TRIAC (o triodo para corriente alterna) es un dispositivo semiconductor de la familia de los tiristores. A diferencia del tiristor convencional, el TRIAC permite el flujo de corriente en ambas direcciones, es decir, es capaz de conmutar una señal de corriente alterna. Su estructura interna se asemeja al circuito resultante de conectar dos SCR (rectificador controlado de silicio) en direcciones opuestas. Como se puede observar en la Figura 11, el TRIAC cuenta con tres electrodos: dos terminales principales (MT1, MT2; o ánodo 1 y 2) y una puerta (gate), que controla el disparo del dispositivo.

Figura 11: Modelo del TRIAC

En la Figura 12 se puede observar el efecto de un TRIAC al instalarlo entre una fuente de alimentación de corriente alterna y una carga cualquiera, analizándose como afecta a las formas de onda de la señal en la carga y entre los terminales del TRIAC en dos condiciones diferentes. En la Figura 12 (a), el TRIAC se encuentra apagado durante los primeros 30° de cada semiciclo, comportándose como un interruptor abierto. Durante este periodo, el voltaje de la línea cae a través de los terminales principales del TRIAC, sin aplicar voltaje a la carga. Como se observa, en esta zona del semiciclo el voltaje en la carga es $V_{carga} = 0$, conociéndose esta zona como ángulo de retardo de disparo. Una vez transcurrido el ángulo de retardo, el TRIAC se dispara, empezando a funcionar como un interruptor cerrado, por lo que todo el voltaje de la línea es aplicado a la carga. Está zona se llama ángulo de conducción. El valor eficaz del voltaje quedará determinado por la duración del ángulo de conducción, ya que cuanto mayor sea, más cercano estará al voltaje de línea. La Figura 12 (b) muestra las mismas señales, pero con un mayor ángulo de retardo, $\theta = 120º$, lo que producirá un valor eficaz del voltaje de carga considerablemente inferior. [6]

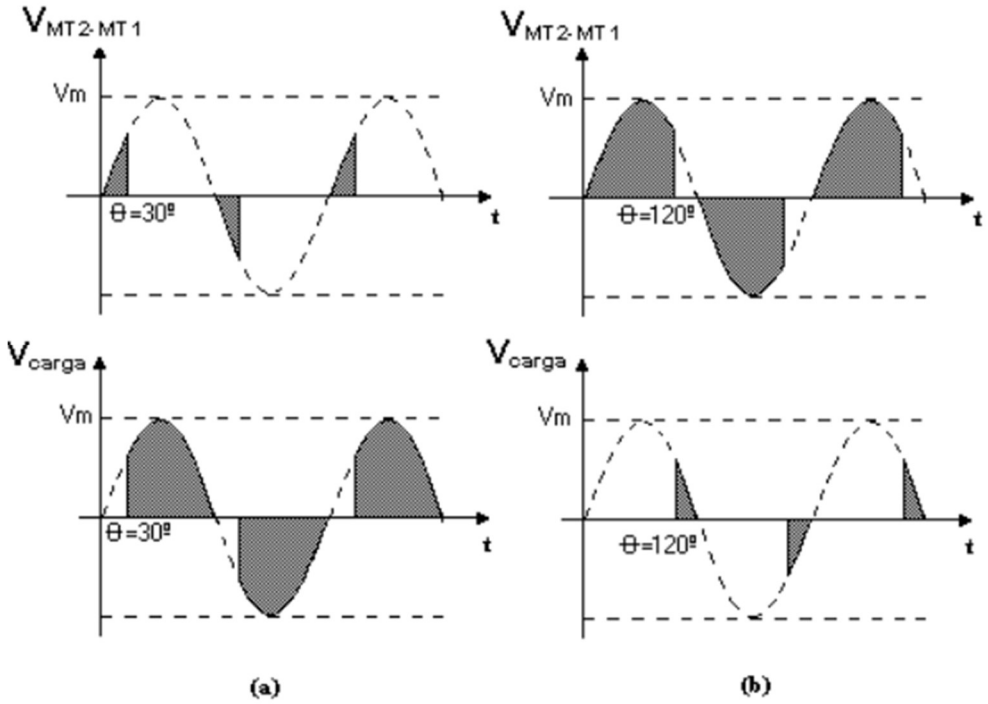

Figura 12: Efecto del TRIAC en la señal AC [6]

El valor eficaz del voltaje en la carga queda determinado por:

$$V_{rms}(\theta) = \frac{V_p}{\sqrt{2\pi}}\sqrt{\pi - \theta + \frac{1}{2}sin(2\cdot\theta)} \qquad \textit{Ec. 1}$$

donde:

- θ: ángulo de retardo, expresado en radianes.
- V_p: voltaje pico de la señal.

De este modo, es posible utilizar un algoritmo PID que controle el disparo del TRIAC, pudiendo así modificar la potencia recibida por una carga que requiera alimentación en AC para conseguir un lazo de control estable y preciso.

3.4. Control de lazo cerrado PID

El lazo de control va a ser implementado mediante un controlador PID (controlador proporcional, integral y derivativo) que se ejecutará en el microcontrolador. Se trata de un algoritmo sencillo y especialmente robusto, por lo que su uso está muy extendido en diversas aplicaciones (reguladores de velocidad, piloto automático, etc.). Para su correcto funcionamiento se necesitan:

- Un sensor, que determine el estado del sistema.
- Un controlador, que genera la señal que gobierna al actuador.
- Un actuador, que modifica el sistema de manera controlada.

3.4.1. El PID tradicional

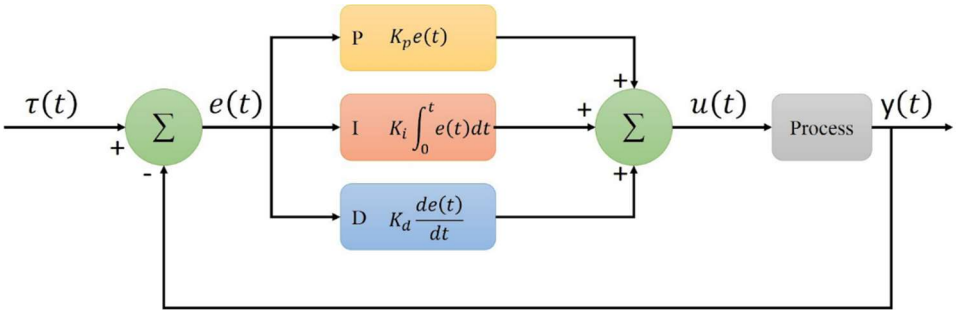

Figura 13: Estructura de un lazo de control con controlador PID

Como se observa a partir de la Figura 13, la ecuación básica del algoritmo PID es:

$$Output = K_p e(t) + K_I \int e(t)dt + K_D \frac{d}{dt} e(t) \qquad \text{Ec. 2}$$

donde:

- $e = Setpoint - Input$: es la diferencia entre el valor deseado (Setpoint) y el valor medido (Input), es decir, el error.
- K_p: ganancia proporcional del controlador.
- K_I: ganancia integral del controlador.
- K_D: ganancia derivativa del controlador.

El **término proporcional** actúa para lograr que el error en el régimen estacionario se aproxime a cero. En la Figura 14 se observa el efecto de un controlador proporcional con diferentes valores de ganancia frente a la misma señal. Como se puede observar, valores altos generan una respuesta más rápida y con menor error permanente (aunque nunca se llega a anular), pero producen mayor oscilamiento y tiempo de estabilización, pudiendo llegar a hacer el sistema inestable. [7]

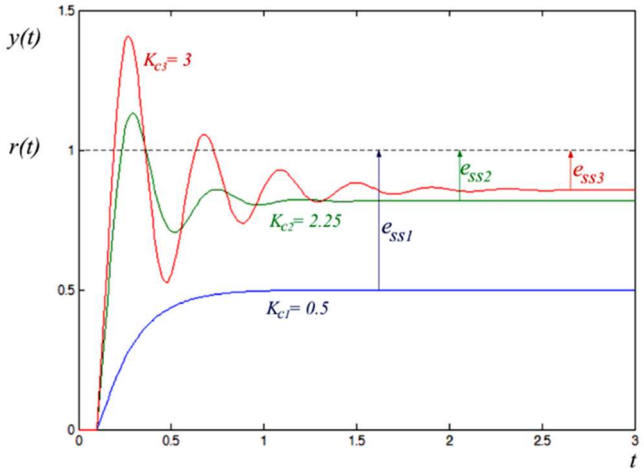

Figura 14: Respuesta de un controlador P con diferentes ganancias [7]

El **término integral** se encarga de eliminar el error permanente en la respuesta. En función del valor que se le dé a la ganancia integral, se modifica la constante de tiempo integral:

$$T_i = 1/K_i$$

Este tiempo se define como "aquel que debe transcurrir para que la acción integral alcance (iguale o repita) a la acción proporcional" [7]. El inconveniente de este elemento es que, a pesar de eliminar el error permanente, agrega un polo en el origen, lo que incorpora un elemento de desestabilización. Como se puede ver en la Figura 15, en la respuesta del controlador proporcional-integral se consigue eliminar el error, pero también tiene una menor estabilidad relativa al obtenerse una mayor sobreoscilación y tiempo de estabilización.

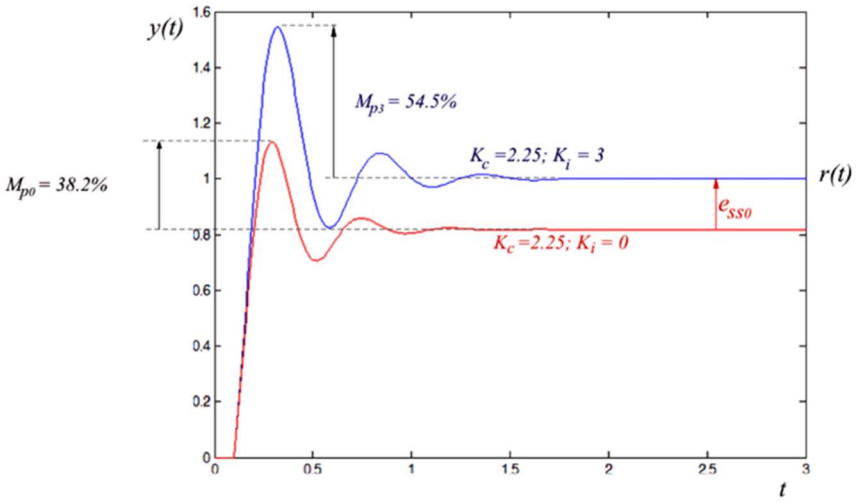

Figura 15: Respuesta de un controlador PI con diferentes ganancias [7]

El **término derivativo** se incorpora para contrarrestar la inestabilidad introducida por el término integral. En la Figura 16 se aprecia claramente la mayor estabilidad relativa del controlador PID completo en la respuesta (representada en negro): elimina completamente el error permanente,

mientras que conserva una buena velocidad de crecimiento y reduce considerablemente la máxima sobreoscilación M_p y el tiempo de estabilización.

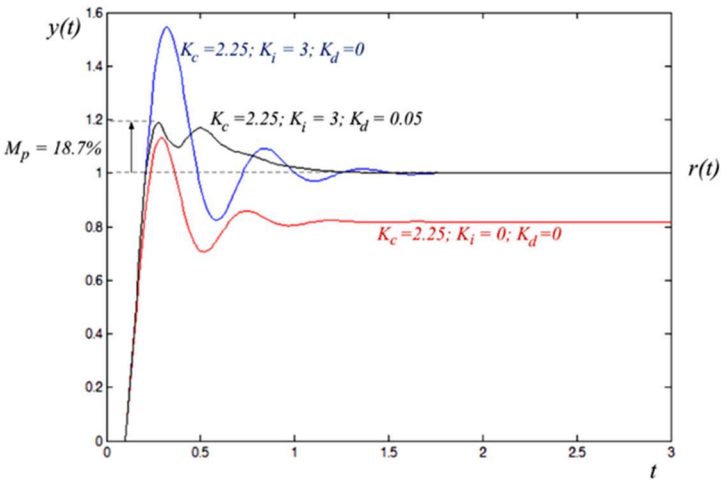

Figura 16: Respuesta de un controlador PID con diferentes ganancias [7]

Un inconveniente de este modelo de algoritmo PID es la existencia de los llamados "Integrating Process" [8]. En un sistema de control lo habitual es que la entrada ante un valor de salida (Output) fijo se autorregule a un valor determinado, por ejemplo, el flujo de agua a través de una tubería. Estos procesos se conocen como "Self-Regulating Process" y puede verse un ejemplo en la Figura 17 izquierda, donde ante un Output de escalón el Input se establece a un valor fijo tras un tiempo de estabilización. Sin embargo, en el caso de un "Integrating Process", el output del PID controla la pendiente del cambio del Input. Esto hace que ante una entrada escalón el Input no se fije en un valor, creciendo este de forma constante con una pendiente determinada por el valor del Output, tal y como se puede ver en la Figura 17 derecha.

Como se puede ver en la Figura 17, la diferencia entre ambos tipos de proceso al introducir un escalón en el output es evidente: mientras que en el "Self-Regulating Process" el Input se estabiliza en un valor determinado, en el "Integrating Process" se produce una pendiente ascendente, por lo que el valor del Input continuará aumentando mientras el Output tenga ese valor.

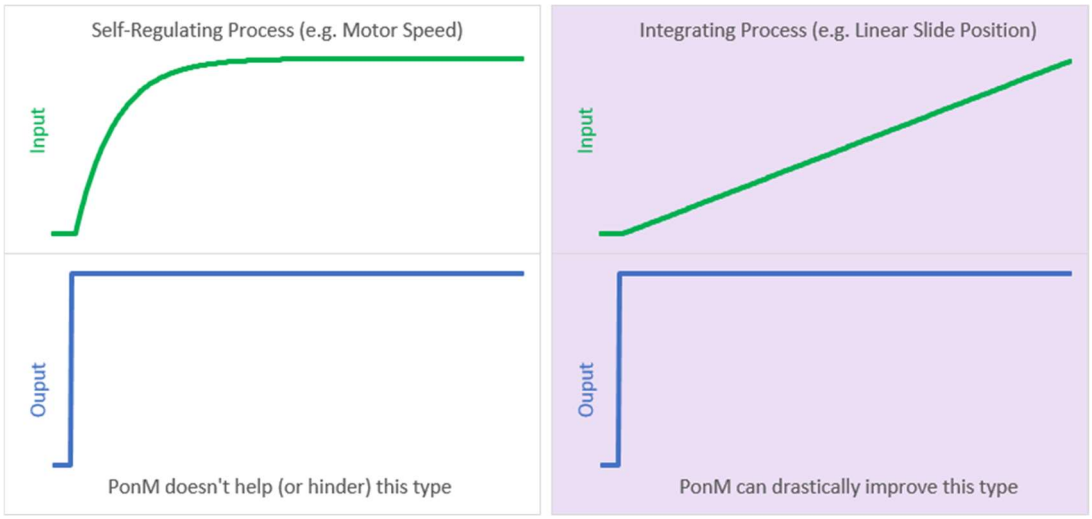

Figura 17: Comparación de la respuesta a un escalón en la salida de ambos tipos de procesos [9]

Un ejemplo claro de ambos tipos de proceso son los formados por los siguientes sistemas de tuberías. El primero, mostrado en la Figura 18, consiste en un sistema de control de flujo de agua por una tubería. Debido a las propiedades de este sistema, al introducir un cambio en la valvula de control con el sistema en modo manual (lazo abierto) la respuesta se estabiliza rápidamente (Figura 19) [8].

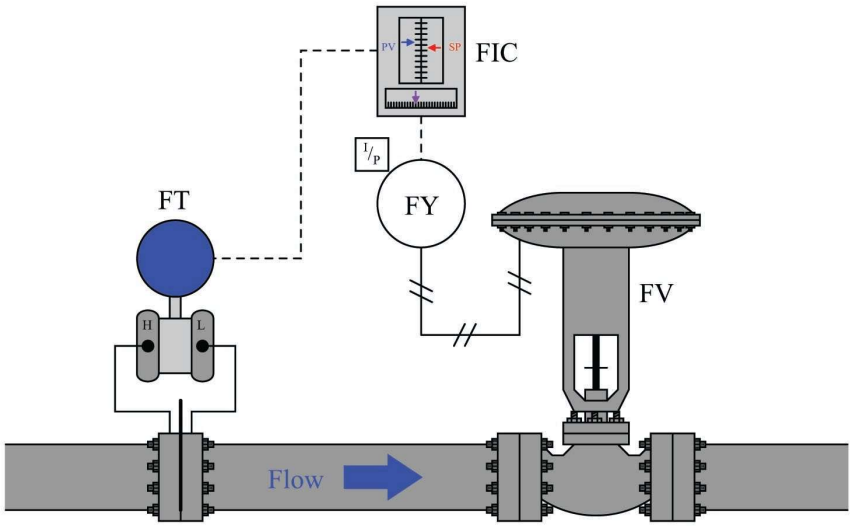

Figura 18: Sistema de tuberías que conforma un "Self-Regulating Process" [8]

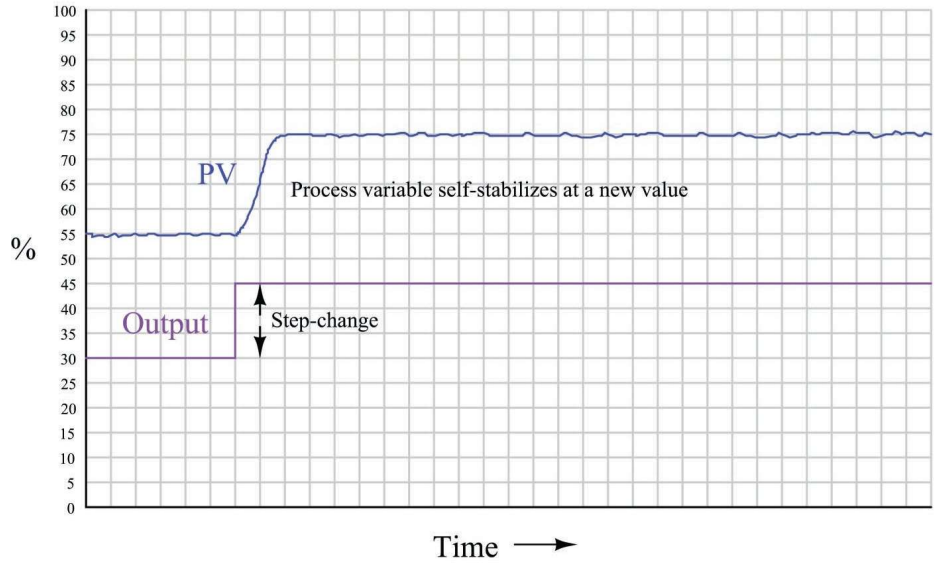

Figura 19: Evolución del Input en un "Self-Regulating Process" [8]

Por otro lado, el sistema mostrado en la Figura 20 forma un "Integrating Process". Como se puede observar, en este caso se está controlando el nivel de agua en un tanque, ajustando para ello el flujo de la tubería de entrada, mientras que la tubería de salida tiene un flujo constante. En este caso, al realizar un cambio en la válvula de control de flujo con el controlador en modo manual (lazo abierto) el nivel de líquido en el tanque cambiará con una pendiente proporcional a la diferencia entre los flujos de entrada y salida (Figura 21) [8]. **El sistema de control de temperatura con el que se va a trabajar se corresponderá con un "Integrating Process"**.

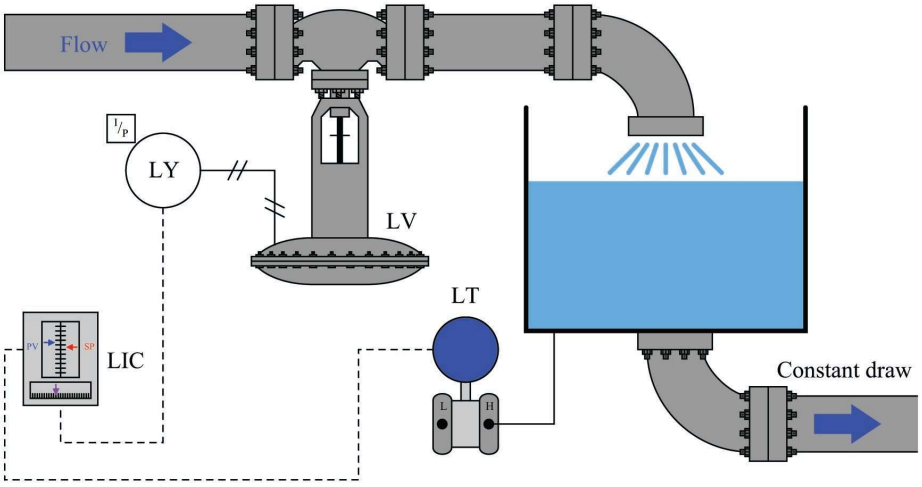

Figura 20: Sistema de tuberías que forma un "Integrating Process" [8]

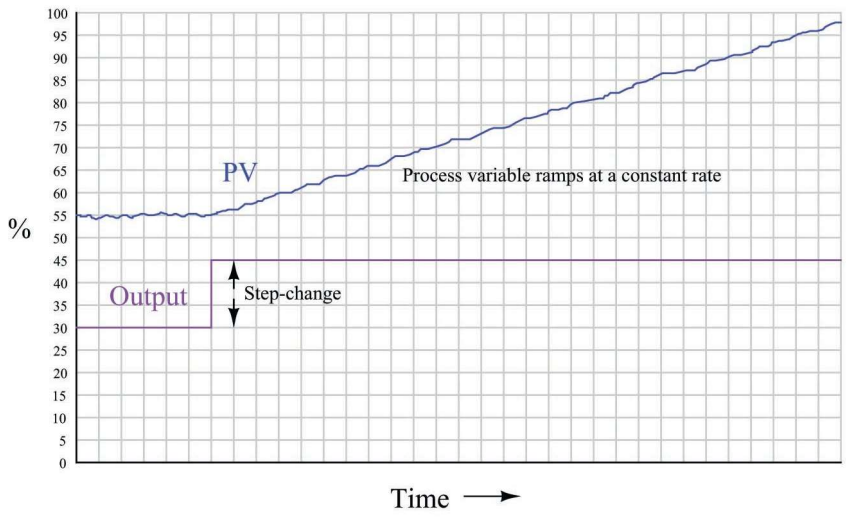

Figura 21: Evolución de la respuesta en un "Integrating Process" [8]

El problema generado por este último tipo de procesos es que utilizando el control PID tradicional siempre se generará sobreoscilación, independientemente de los parámetros que se elijan [9]. Esto, para ciertos procesos que se desean realizar en el biorreactor, puede suponer un fallo irreparable.

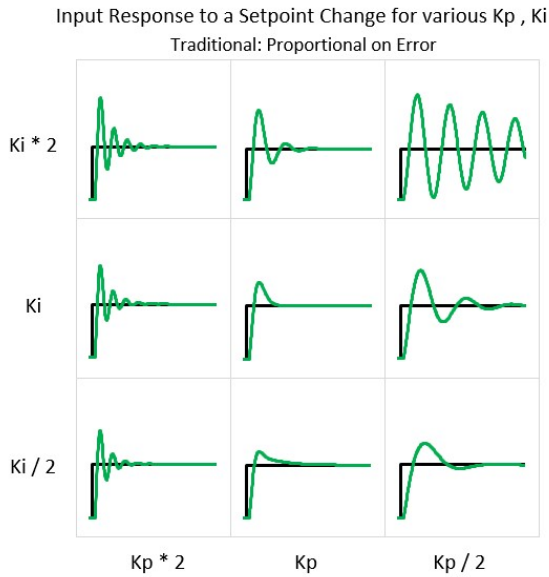

Figura 22: Respuesta del sistema al cambio del Setpoint para varios parámetros de Kp, Ki con un PID tradicional [9]

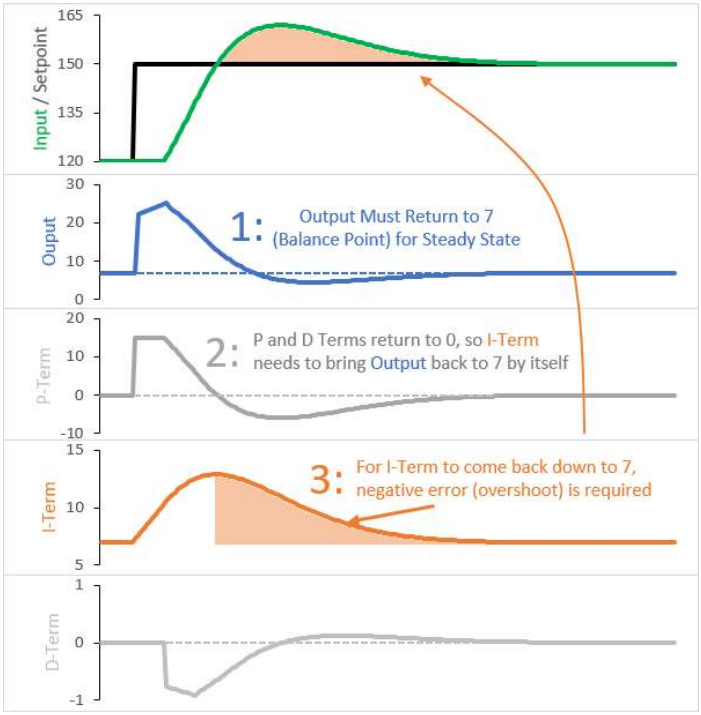

Figura 23: Respuesta a un cambio del Setpoint en un PID tradicional [9]

Para entender mejor la razón de esta sobreoscilación se puede observar el efecto de los diferentes parámetros de un PID tradicional al cambiar el Setpoint de un proceso integrador en la Figura 23. Se realizan las siguientes observaciones [9]:

- Durante el régimen permanente, el termino integral es el único responsable del output final.
- A pesar de que el Setpoint es distinto, el output vuelve al mismo valor. Este valor es generalmente conocido como “punto de balance”: el output que resulta en una pendiente del input igual a 0. En el caso de un control de temperatura, corresponde al punto en el que se genera suficiente calor para compensar las pérdidas al exterior.

Esto explica la existencia de la sobreoscilación, y por qué siempre se producirá con un PID tradicional: cuando el Setpoint cambia, el error existente causa que el termino integral crezca para tratar de reducirlo. Para mantener el proceso estable, el Output debe volver al punto de balance. La forma de que esto suceda es que el termino integral disminuya, pero la única manera de lograr esto es tener un error negativo, por lo que es necesario estar por encima del Setpoint.

Por tanto, para evitar este problema es necesario cambiar el comportamiento del PID: sustituir el termino proporcional en función del error por uno en función de la medida permitirá evitar la sobreoscilación.

3.4.2. El PID con Proporcional en Función de la Medida

Como se ha comentado en el apartado anterior, para conseguir que la sobreoscilación en la respuesta de un "Integrating Process" como con el que se va a trabajar sea evitable es necesario cambiar el modelo del lazo de control. Para ello, el Proporcional en función de la medida **(PonM)** [9] cambia el funcionamiento del PID, haciendo que el termino proporcional observe el valor del Input, en vez de la medida del error, como se observa al comparar Ec. 3 con Ec. 2:

$$Output = -K_p[Input(t) - Input_{init}] + K_i \int e(t)dt - K_d \frac{dInput}{dt} \qquad \text{Ec. 3}$$

El PonM cambia fundamentalmente lo que el termino proporcional hace en el control: en vez de ser una fuerza impulsora, como el termino integral, se convierte en una fuerza resistiva, al igual que el termino derivativo. Esto significa que con PonM cuanto mayor sea el valor de K_p más conservador será el controlador. En la Figura 24, se observa cómo afecta el control PonM al mismo sistema de la Figura 23. [9]

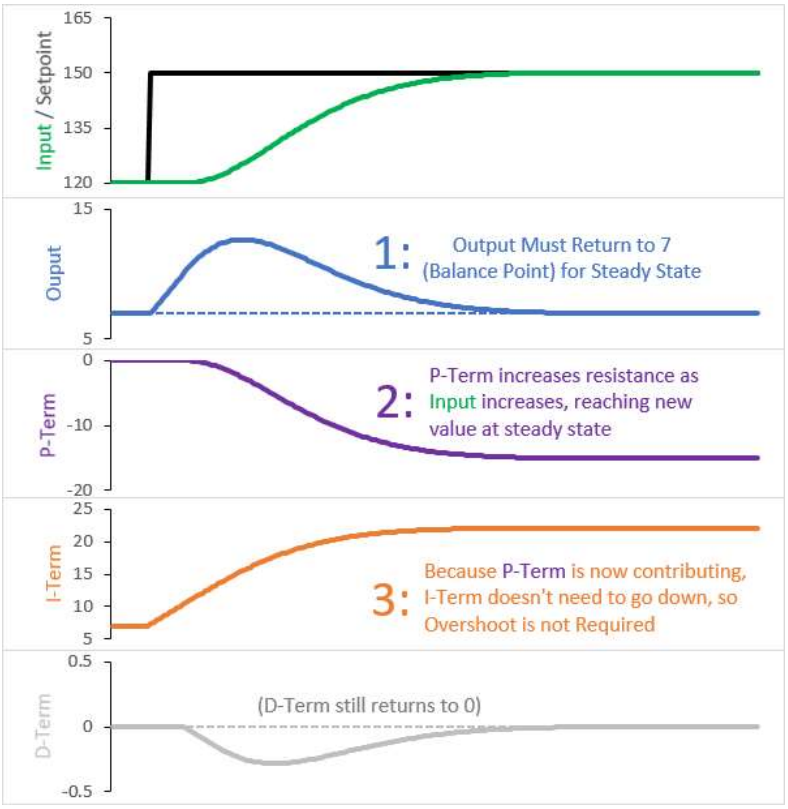

Figura 24: Respuesta a un cambio del Setpoint en un PID con PonM [9]

Los cambios más significantes son los siguientes:

- El termino proporcional es ahora una fuerza resistiva: cuanto más incrementa el Output, más negativo se vuelve su valor.
- En el PID tradicional, el termino derivativo se vuelve cero en el nuevo Setpoint, pero ahora continúa teniendo un valor.

La clave del funcionamiento del PonM es el hecho de que el termino proporcional no vuelve al valor cero: por tanto, el termino integral no tiene que volver al punto de balance por sí solo, sino que lo hace en conjunto con el proporcional. Esto significa que el termino integral no se tiene que reducir, por lo que no es necesario que se produzca sobreoscilación. En la Figura 25 se observa la respuesta del sistema frente a un cambio del Setpoint dependiendo de sus parámetros K_p, K_i con un PID PonM. Como se observa, a diferencia del sistema de la Figura 22 ahora es posible obtener una respuesta en la que se elimina completamente la sobreoscilación. [9]

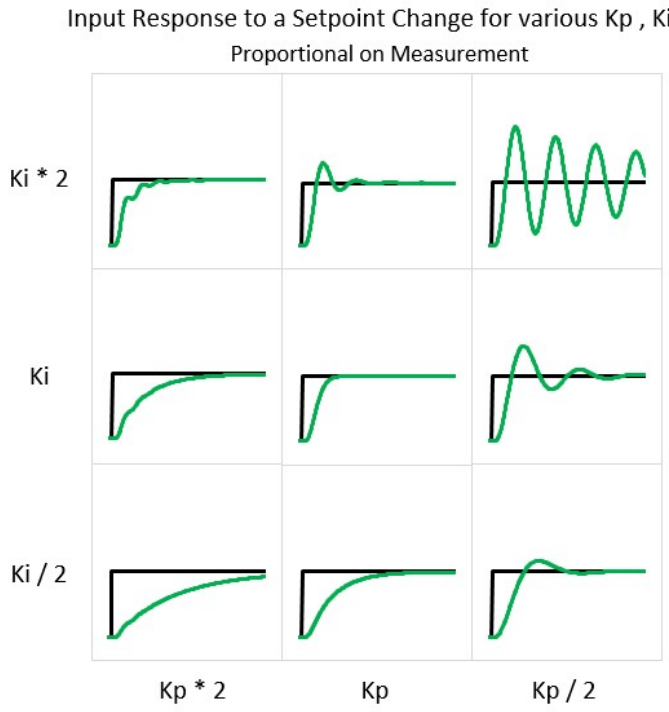

Figura 25: Respuesta del sistema al cambio del Setpoint para varios parámetros de Kp, Ki con un PID con PonM [9]

3.4.3. Métodos de sintonía

Existen una gran cantidad de métodos para obtener los parámetros de un controlador PID basados en diferentes metodologías en función del modelo disponible del sistema. Sin embargo, dado que en este proyecto no se cuenta con un modelo, ha sido necesario emplear métodos empíricos. Estas reglas ayudan a sintonizar correctamente un controlador PID cuando no se dispone del modelo, es decir, ayudan a fijar los valores de K_p, K_I y K_D para obtener un modelo estable del sistema. A continuación, se presentarán brevemente los diferentes métodos probados.

- **Prueba y error**

En este método, se asignan valores iniciales para K_p, K_i, K_d, que se van modificando en función del comportamiento de la respuesta siguiendo los siguientes criterios:

- K_p elevado genera sobreoscilación, pero hace que la respuesta sea más rápida

- K_i es necesario para eliminar el error permanente del proporcional, pero un valor muy elevado vuelve al sistema inestable
- K_d suaviza la respuesta para reducir la sobreoscilación, pero valores elevados ralentizan el sistema y lo pueden volver inestable

- **Regla de Ziegler-Nichols**

Ziegler y Nichols proponen el siguiente método para sintonizar el PID [10]: se comienza fijando los parámetros integral y derivativo con los valores $K_i = 0$ y $K_d = 0$. Utilizando únicamente el control proporcional, se aumenta el valor de K_p desde 0 hasta que se alcanza el punto crítico K_{cr}, donde la respuesta exhibe oscilaciones constantes, como las representadas en la Figura 26. Tanto K_{cr} como su periodo correspondiente P_{cr} (T_{cr}) se obtienen de manera experimental. De acuerdo con Ziegler-Nichols, los valores finales de los parámetros integral y derivativo se obtienen mediante las ecuaciones de la Tabla 1.

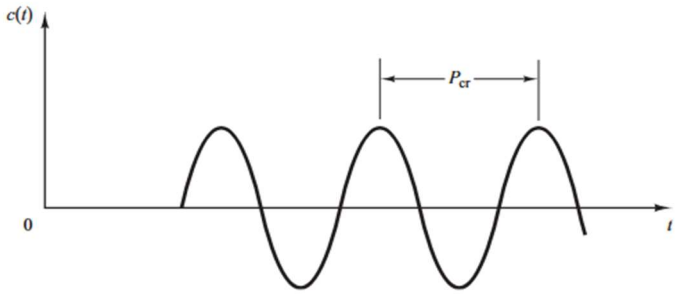

Figura 26: Oscilación sostenida con P_{cr} [10]

Control Type	K_p	K_i	K_d
P	$0{,}50K_{cr}$	$-$	$-$
PI	$0{,}45K_{cr}$	$0{,}54K_{cr}/T_{cr}$	$-$
PID	$0{,}60K_{cr}$	$1{,}2K_{cr}/T_{cr}$	$3K_{cr}T_{cr}/40$

Tabla 1: Valores de sintonización para diferentes controladores con el método de Ziegler-Nichols [10]

- **Método del relé (Åström-Hägglund)**

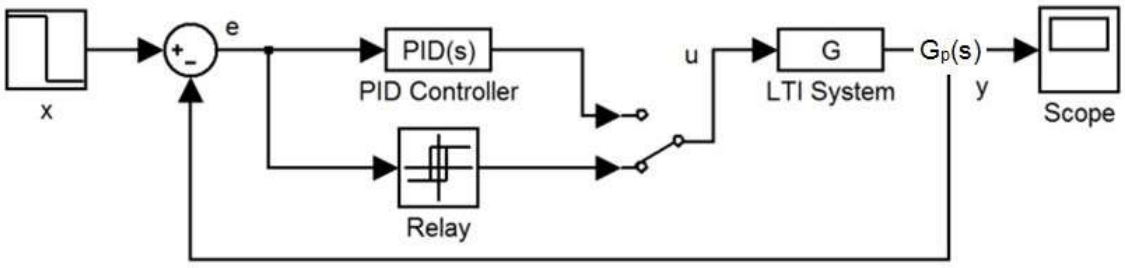

Figura 27: Esquema del método del relé [11]

En este método, los parámetros K_{cr} y P_{cr} se obtienen sustituyendo el controlador PID por un relé una vez alcanzado el régimen permanente. De este modo, el sistema se convierte en uno de control ON/OFF y se obtiene una oscilación constante en la respuesta (Figura 28) similar a la del método de Ziegler-Nichols, de la cual se pueden obtener los parámetros $T_u = P_{cr}$ y $K_{cr} = (4 * b)/(\pi * a)$. Utilizando la Tabla 1, se obtienen así una buena aproximación a los parámetros del PID. [11] [12]

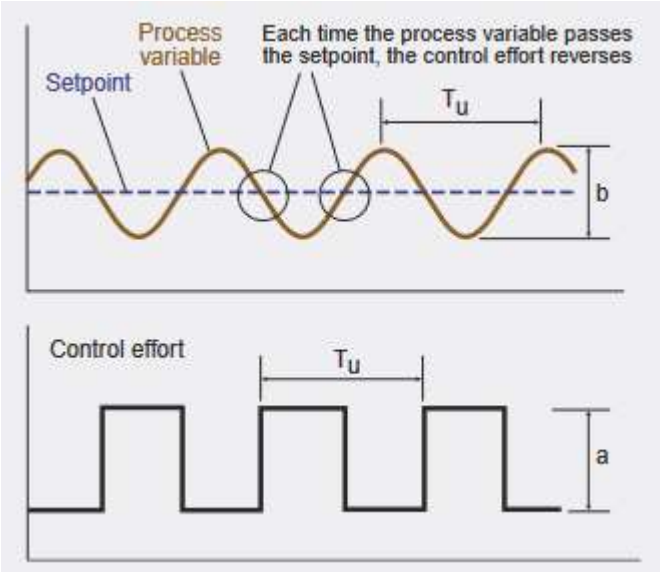

Figura 28: Funcionamiento del método del relé [12]

3.5. Interfaz

La interfaz permitirá al usuario interactuar con el sistema para poder configurarlo y establecer la programación de diferentes intervalos a cierta temperatura en función de la prueba a realizar. Deberá contar con una pantalla que refleje la información deseada y una serie de botones para facilitar la programación de la temperatura y duración deseadas para el proceso.

4. Elementos Hardware

4.1. Introducción

En el capítulo anterior se ha presentado la estructura general del sistema que se va a desarrollar. A continuación, se van a ver los diferentes elementos físicos que se emplearán para implementar los bloques que conforman el sistema y realizar las funciones necesarias para su correcto funcionamiento (detección de temperatura, interfaz con el usuario, sistema de actuación…). En este apartado se hablará con mayor profundidad sobre el funcionamiento y la razón de su elección para cada uno de estos elementos:

- Microcontrolador: Arduino Uno
- Sensor de temperatura: Pt-100
- Acondicionamiento del sensor: Adafruit MAX31865
- Sistema de actuación: Control de fase mediante TRIAC
- Pantalla LCD
- Calefactor

4.2. Microcontrolador

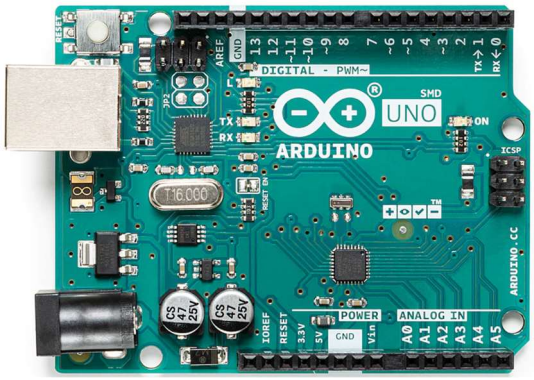

Figura 29: Arduino UNO

El microcontrolador será el elemento encargado de gestionar todo el sistema: ejecutará el controlador PID para el lazo de control del calefactor, a la vez que recibe datos del sensor y los muestra en un display LCD. Se ha optado por emplear una placa Arduino [13], ya que se trata de una plataforma de software libre muy extendida, pensada para su empleo en una gran variedad de proyectos electrónicos. Por ello, es sencillo encontrar hardware diseñado para su uso con este tipo de placas junto con diferentes librerías, facilitando así el diseño.

Existen múltiples modelos diferentes de placas Arduino, con diferentes arquitecturas, funcionalidades y número de pines. En este proyecto, el modelo que se va a emplear es el Arduino Uno, debido a su disponibilidad en la empresa. Sus principales características son:

- Basada en un microcontrolador ATmega328p
- 14 entradas / salidas digitales, 6 de las cuales se pueden usar como salida PWM de 8 bits.
- 6 entradas analógicas con una resolución de 10 bits.
- Conector USB para la programación.
- Botón de reinicio

- Múltiples métodos de alimentación:
    - Puerto USB
    - Puerto de alimentación jack que permite alimentar con un voltaje entre 7 y 12V
    - Puerto V_{IN} que permite alimentar con un voltaje entre 6 y 12V

4.3. Sensor de temperatura: Pt-100

Figura 30: Sensor Pt-100

El sensor que se va a emplear para la medición de temperatura en el interior del reactor es una termorresistencia (RTD), más concretamente, un Pt-100 de tres cables. Se trata de un sensor metálico con coeficiente térmico positivo, es decir, este tipo de sensor aumenta su resistencia al aumentar la temperatura. El principio de funcionamiento de este tipo de sensor es sencillo: al aumentar la temperatura de un metal, sus átomos aumentan su energía cinética, dificultando el movimiento de los electrones por el campo eléctrico. Esto provoca una disminución de la conductividad del material, es decir, aumenta su resistividad (ρ) y con ella la resistencia (R), que vendrá dada por la Ec. 4:

$$\boxed{R = \rho * \frac{l}{A}} \qquad \text{Ec. 4}$$

donde *l* es la longitud del hilo metálico, *A* la sección del hilo y ρ la resistividad del material.

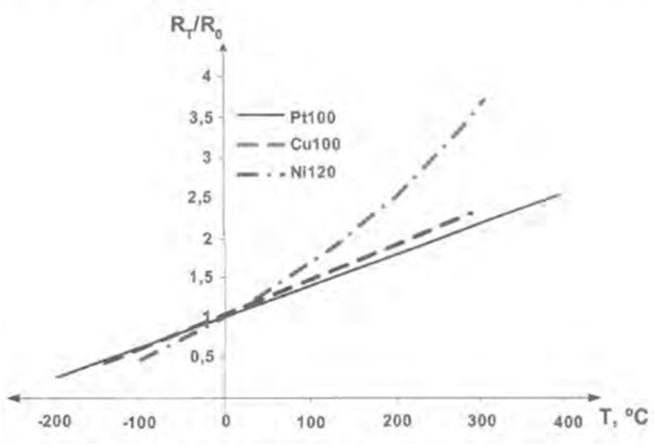

Figura 31: Curvas de calibración normalizadas de 3 RTDs [14]

Las RTDs pueden ser construidas de diversos metales, lo que afecta a sus características (sensibilidad, linealidad, etc.). En la Figura 31, se han representado las curvas de calibración para tres RTDs de diferentes materiales. Como se puede observar, la RTD de platino es la que peor sensibilidad tiene. Sin embargo, cuenta con la mayor linealidad de todas, lo que convierte a los sensores de platino en los más utilizados para la mayoría de las aplicaciones. El modelo matemático [14] que mejor se ajusta a está curva es Ec. 5:

$$R_T = R_0 * (1 + \alpha * \Delta T + \beta * \Delta T^2 + \gamma * \Delta T^3 + \ldots) \quad \text{Ec. 5}$$

Donde R_0 es la resistencia de la RTD a $T = T_0\ °K$, ΔT es la variación de la temperatura $(T - T_0)$ y $\alpha, \beta, \gamma \ldots$ son los coeficientes del modelo para los términos ideal, cuadrático, cúbico, etc... Dado que los valores de β y γ son del orden de 10^4 y 10^9 veces más pequeños que el valor de α para el platino, el modelo matemático se puede aproximar a la Ec. 6:

$$\boxed{R_T = R_0 * (1 + \alpha * \Delta T)} \quad \text{Ec. 6}$$

donde α es el coeficiente térmico de la RTD (K^{-1}) y R_0 es el valor de la resistencia de la RTD para $T = 0\ °C$.

La ventaja de utilizar un modelo de Pt-100 con tres o cuatro cables frente a uno de dos es que se reduce el error introducido en la medida debido a la resistencia del cable. Debido a que los modelos de cuatro cables pueden resultar caros, lo más común es encontrar RTDs de tres cables a pesar de no tener un nivel de exactitud tan alto, ya que si los tres cables son iguales se elimina el error casi por completo.

4.4. Acondicionamiento del sensor de temperatura

Para poder tomar medidas precisas de la temperatura es necesario emplear un circuito de acondicionamiento capaz de convertir las variaciones de resistencia a un rango de variaciones de voltaje adecuado. Para la aplicación en este trabajo se ha determinado que las características mínimas deseadas son una resolución de 0,1 °C y una precisión de 1 °C. Estos parámetros dependerán del circuito de acondicionamiento empleado, habiéndose barajado dos opciones: construir un puente de Wheatstone con un amplificador de instrumentación, o adquirir el sistema completo en una tarjeta de adquisición. Inicialmente se decidió emplear el puente de

Wheatstone, pero finalmente se optó por la otra opción, debido a que era más económica y fácil de adquirir para la empresa.

4.4.1. Puente de Wheatstone

El puente de Wheatstone es un circuito eléctrico capaz de medir el valor de una resistencia desconocida, en este caso el sensor Pt-100, mediante el equilibrio de los brazos del puente, obteniéndose así un alto nivel de precisión. Al medir la diferencia de voltaje entre ambos brazos (V en la Figura 32), es posible calcular el valor aproximado de la resistencia variable mediante la Ec. 7.

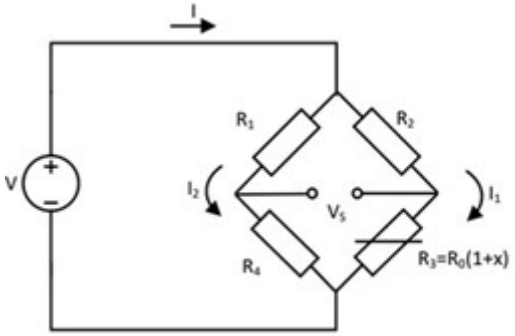

Figura 32: Modelo del Puente Wheatstone

$$V_S = V\left(\frac{R_3}{R_2 + R_3} - \frac{R_4}{R_1 + R_4}\right) \qquad \text{Ec. 7}$$

Donde *V* es el voltaje suministrado por la fuente de alimentación, que en este caso será Arduino (5 V), R_1, R_2, R_4 son resistencias y R_3 es el sensor Pt-100. Si calculamos el valor de V_S a una temperatura máxima de 100 °C, cuando el sensor toma un valor de 138,5 Ω, y se emplean unos valores de resistencia R1=R2=6.800Ω y R4=100Ω, se obtiene que la salida tendrá un valor $V_S = 27{,}3\ mV$.

Por tanto, este circuito tiene una sensibilidad de unos pocos $\mathrm{mV/°C}$, no teniendo el rango adecuado para poder conectarlo a un conversor A/D. Es necesario amplificar la señal en voltaje usando un amplificador de instrumentación (INA) capaz de convertir la señal al rango de funcionamiento de Arduino: 0-5 V.

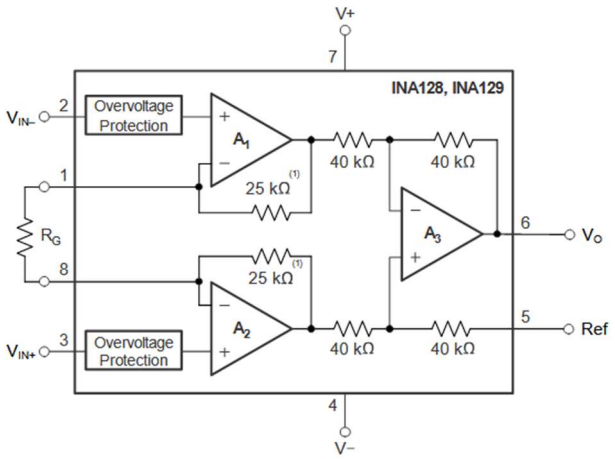

Figura 33: Esquema del INA128P [15]

$$\boxed{G = 1 + \frac{50\ k\Omega}{R_G}} \qquad \textit{Ec. 8}$$

En la Figura 33 se observa el esquemático del INA128P [15], que ha sido elegido para amplificar la señal. Este amplificador cuenta con una ganancia regulable a través de la resistencia externa R_G de acuerdo con la Ec. 8. Dado que se busca una ganancia de 183 V/V para ajustar el voltaje de salida al rango de operación del conversor A/D, el valor de la resistencia será:

$$R_G = \frac{50\ k\Omega}{G - 1} = 267\ \Omega \qquad \textit{Ec. 9}$$

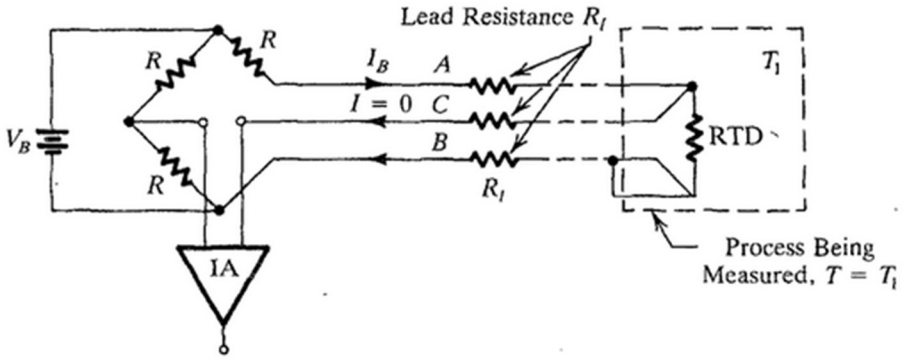

Figura 34: Conexión de un RTD de 3 cables con Puente Wheatstone [16]

La Figura 34 muestra la conexión del circuito de acondicionamiento con el sensor de tres cables, la salida del amplificador de instrumentación se conectaría con una de las entradas analógicas de Arduino. El conversor A/D de Arduino cuenta con 10 bits, por tanto, en un rango de temperaturas de 0 °C a 100 °C se obtiene la siguiente resolución:

$$Resolución = (100°C - 0°C)/(2^{10}) = 0{,}097°C/bit$$

Dado que se busca una resolución de 0,1 °C, el conversor A/D de Arduino es suficiente, por lo que no es necesario comprar uno externo.

4.4.2. Tarjeta de adquisición Adafruit MAX31865

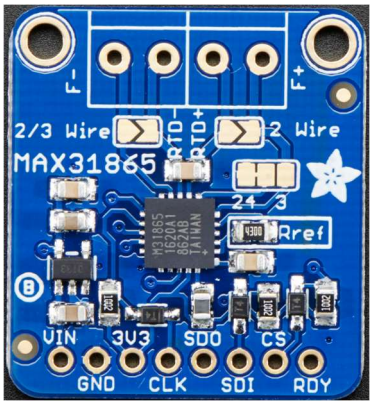

Figura 35: Adafruit MAX31865

Finalmente, se optó por no emplear el puente de Wheatstone, debido a que su precio era más elevado que adquirir una tarjeta de adquisición. Esto se debía a los recursos y logística de la empresa: dado que no contaba con ninguno de los elementos de antemano, era necesario adquirirlos de numerosos vendedores de Amazon, no disponiendo además de equipamiento para trabajar con dispositivos electrónicos (soldadores, osciloscopios, multímetros…). Esta tarjeta es un sistema completo para la medición de temperatura compuesto de una pequeña PCB que incluye todo el acondicionamiento requerido para obtener el valor de temperatura medido a través de sensores resistivos de temperatura en configuraciones de dos, tres o cuatro hilos. [17]

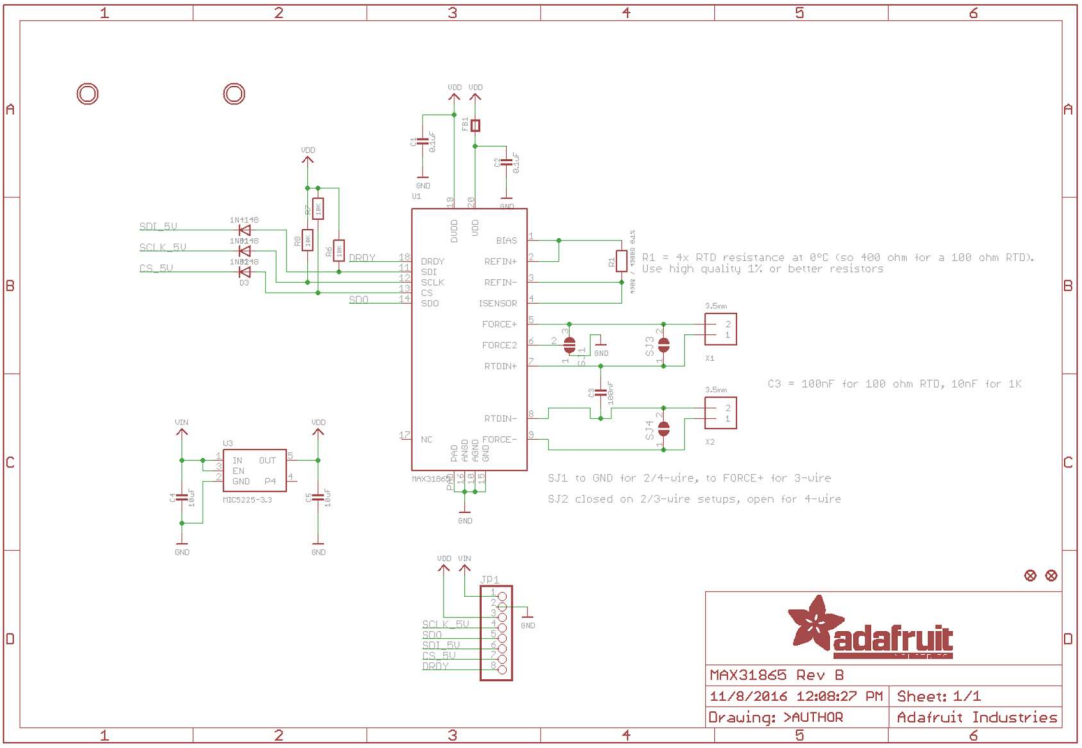

Figura 36: Esquemático del Adafruit MAX31865 [17]

Como se observa en la Figura 36, el esquemático de la placa cuenta con tres elementos principales:

- **Circuito MAX31865** [18]**:** circuito integrado que proporciona la excitación para el sensor y convierte el valor de resistencia medido a digital, enviándolo a través de un bus SPI a Arduino.
- **Regulador de tensión:** dado que el chip soporta un voltaje máximo de alimentación de 3,6V, es necesario reducir el voltaje suministrado por Arduino a su valor típico de trabajo, es decir, de 5V a 3,3V.
- **Pinout:** incluye los pines de alimentación y el bus SPI.
    - *Pines de alimentación*
        - *Vin:* Alimenta la placa empleando los 5V proporcionados por - Arduino. Dado que el chip MAX31865 se alimenta a 3,3V, este pin está conectado al regulador de tensión.
        - *3V3:* Se trata de la salida del regulador de tensión a 3,3V. En caso de que se quiera, puede proporcionar hasta 100 mA a través de esta salida.
        - *GND:* Tierra.
    - *Pines SPI:* pines que conectan el chip MAX31865 con Arduino para la transmisión de la información.
        - *SCK:* SPI Clock Pin, reloj del bus SPI.
        - *SDO:* Serial Data Output, envía los datos del chip a Arduino.
        - *SDI:* Serial Data Input, envía los datos de Arduino al chip.

- *CS:* Chip Select pin, se pone a 0 para activar el chip e iniciar una transacción SPI.
- *RDY:* Indica que el chip está listo para enviar datos a Arduino. Se puede utilizar para realizar las lecturas en cuanto se dispone del valor medido. No es necesario usarlo.

Dado que el dispositivo es válido para RTD de dos, tres o cuatro cables, es necesario configurarlo en función del uso que se le vaya a dar. Para ello, la placa cuenta con los siguientes "jumpers", que configuran el circuito en torno al RTD utilizado, como se observa en la Figura 37.

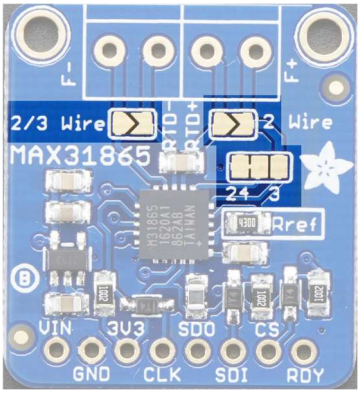

Figura 37: Pads de configuración del Adafruit MAX31865

Como se puede observar en la Figura 37, por defecto la placa está configurada para funcionar con sensores de cuatro cables. Dado que el sensor que se va a emplear es un Pt100 de tres cables, será necesario soldar el jumper marcado como "⅔ wire", cortar la unión del pad "24" y soldar el pad 4 con el 3. En la Figura 38 se demuestra cómo estos cambios forman el circuito de la Figura 40:

- SJ1: FORCE+ y FORCE2 están cortocircuitados.
- SJ3: No es necesario, se usa para RTDs de dos cables.
- SJ4: RTDIN- y FORCE- están cortocircuitados.

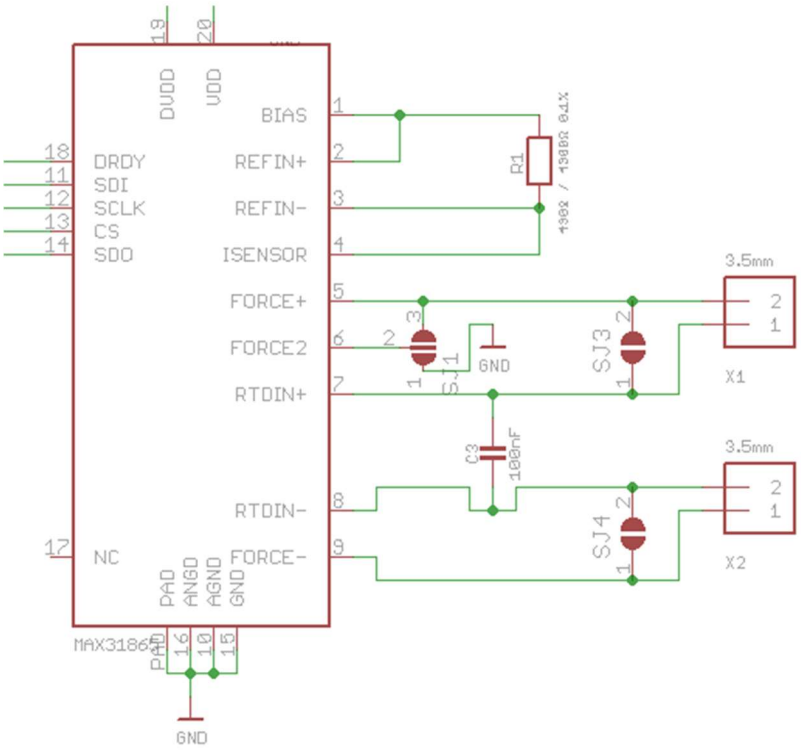

Figura 38: Esquemático de la conexión del MAX31865 [17]

A continuación, se va a hablar en más detalle sobre el chip que se emplea en la placa de desarrollo para el acondicionamiento de la PT-100, el MAX31865 [18]. Este circuito incorpora la excitación y el acondicionamiento necesario para conocer la variación de resistencia producida en un sensor y está orientado a su uso con diferentes sensores de temperatura, especialmente PT-100 y PT-1000. Sus características principales son:

- Conversión A/D con 15 bits de resolución.
- Resolución de la medida: 0,03125 °C.
- Precisión: 0,5 °C.
- Tiempo de conversión máximo: 21 ms.
- Sistema de detección de fallos integrado.
- Compatible con sensores de dos, tres o cuatro cables.

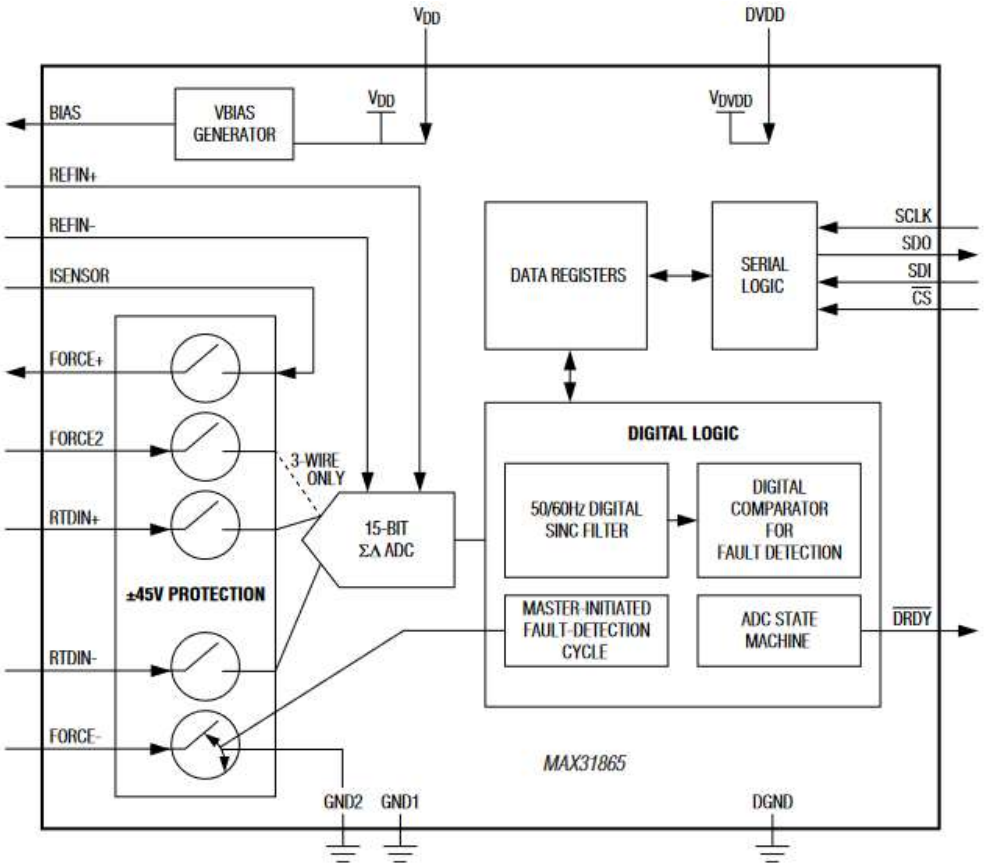

Figura 39: Diagrama de bloques del MAX31865 [18]

En la Figura 39 se observa el diagrama de bloques que representa la estructura interna del circuito. Como se puede observar, las conexiones con la RTD cuentan con una protección ante voltajes de ±45 V y están unidas al conversor A/D, al igual que la R_{REF}. El resultado de la conversión pasa por el filtro de rechazo de ruido y sistema de detección de fallos. Los resultados de la lógica digital se escriben en los registros de datos para posteriormente ser leídos por el Arduino mediante los pines de lógica SPI.

Sobre la estructura interna del MAX31865 cabe destacar el método empleado para la linealización del RTD sin hacer uso de una fuente de corriente. En su lugar, se utiliza un divisor de tensión entre el RTD y la resistencia externa de precisión, R_{REF}. Como se puede observar, ambas resistencias se encuentran conectadas entre sí mediante los terminales $FORCE+$ y $ISENSOR$. Por tanto, la corriente que circula por RTD es el resultado de la relación R_{RTD}/R_{REF}, lo que facilita obtener el valor de la resistencia del sensor empleando Ec. 10 [19]:

$$R_{RTD} = \frac{(ACD\ CODE\ \cdot R_{REF})}{2^{15}} \qquad \text{Ec. 10}$$

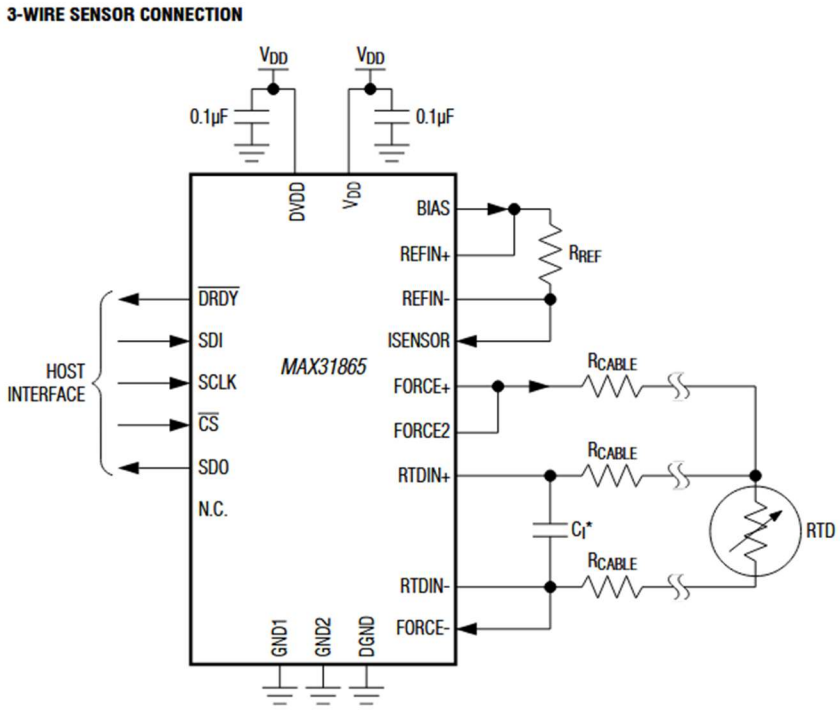

Figura 40: Esquema de la conexión del MAX31865 con un RTD de 3 cables [18]

En la Figura 40 se observa la configuración del MAX31865 para su uso con un sensor de tres cables. De acuerdo con las especificaciones de la datasheet [18], los valores de los elementos utilizados en esta conexión deben ser los siguientes:

- C_1: 100nF para un Pt100
- R_{REF}: El valor óptimo para la resistencia de referencia de un RTD de platino es cuatro veces su resistencia a 0ºC, es decir, 400Ω. En el circuito se ha aproximado a 430Ω, con una tolerancia del 0,1%.

El circuito cuenta con 8 registros de 8 bits, que contienen los datos de configuración, conversión y estado. La programación se realiza eligiendo la dirección del registro deseado y leyendo o escribiendo sus valores. A continuación, se explicará brevemente la función de cada uno de los registros.

REGISTER NAME	READ ADDRESS (HEX)	WRITE ADDRESS (HEX)	POR STATE	READ/WRITE
Configuration	00h	80h	00h	R/W
RTD MSBs	01h	—	00h	R
RTD LSBs	02h	—	00h	R
High Fault Threshold MSB	03h	83h	FFh	R/W
High Fault Threshold LSB	04h	84h	FFh	R/W
Low Fault Threshold MSB	05h	85h	00h	R/W
Low Fault Threshold LSB	06h	86h	00h	R/W
Fault Status	07h	—	00h	R

Tabla 2: Registros del MAX31865 [18]

- **Registro de configuración (00h)**

Indica, entre otras cosas, el tipo de conexión del sensor, la frecuencia del filtro de rechazo, etc. En la Tabla 3 se establece que configura cada bit:

D7	D6	D5	D4	D3	D2	D1	D0
V_{BIAS} 1 = ON 0 = OFF	Conversion mode 1 = Auto 0 = Normally off	1-shot 1 = 1-shot (auto-clear)	3-wire 1 = 3-wire RTD 0 = 2-wire or 4-wire	Fault Detection Cycle Control (see Table 3)		Fault Status Clear 1 = Clear (auto-clear)	50/60Hz filter select 1 = 50Hz 0 = 60Hz

Tabla 3: Registro de Configuración (00h) [18]

- D7: Activa/Desactiva la alimentación del sensor. Cuando no se están realizando conversiones, se puede poner a 0 para reducir disipación de potencia. Se debe cambiar a 1 antes de realizar una conversión, o dejarlo encendido si se van a realizar continuamente (modo automático).
- D6: Activa/Desactiva las conversiones de manera continua a una ratio de 50/60 Hz.
- D5: Si el modo de conversión no está en automático, cambiar este bit a 1 comienza una conversión.
- D4: Especifica el tipo de sensor utilizado (2, 3 o 4 cables).
- D3-D2: Controla el modo del ciclo de detección de fallos para los bits D5-D3 del registro del estado del error (07h). En la Tabla 4 se observan las posibles configuraciones de estos bits.
- D1: Poner este bit a 1 devuelve todos los bits del registro de estado de error (07h) a 0.
- D0: Selecciona la frecuencia del filtro de rechazo de ruido incorporado en el chip entre 50 y 60 Hz.

D3	D2	CONFIGURATION REGISTER WRITE (BINARY)	WRITE ACTION	READ MEANING
0	0	XXXX00XXb	No action	Fault detection finished
0	1	100X010Xb	Fault detection with automatic delay	Automatic fault detection still running
1	0	100X100Xb	Run fault detection with manual delay (cycle 1)	Manual cycle 1 still running; waiting for user to write 11
1	1	100X110Xb	Finish fault detection with manual delay (cycle 2)	Manual cycle 2 still running

Tabla 4: Posibles configuraciones para los bits D3-D2 [18]

- **Registros del valor RTD (01h-02h)**

Contienen el valor de la relación de resistencia entre el sensor y la resistencia de referencia, almacenándose en 2 registros de 8 bits, RTD MSBs (01h) y RTD LSBs (02h). Estos registros almacenan en los 15 bits más significativos el valor de la relación, mientras que el bit 0 del registro RTD LSBs indica si se ha detectado algún fallo:

REGISTER	RTD MSBS (01h) REGISTER								RTD LSBS (02h) REGISTER							
Bit	D7	D6	D5	D4	D3	D2	D1	D0	D7	D6	D5	D4	D3	D2	D1	D0
RTD Resistance Data	MSB	—	—	—	—	—	—	—	—	—	—	—	—	—	LSB	Fault
Bit Weighting	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	—
Decimal Value	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1	—

Tabla 5: Registros del valor RTD (01h-02h) [18]

Una vez se envían los valores de la conversión A/D a Arduino, se puede convertir al valor de temperatura siguiendo el siguiente proceso:

1. Calcular el valor resistivo del sensor empleando la donde ACD Code son los 15 bits resultado de la conversión A/D en los registros de RTD Data y R_{REF} el valor de la resistencia de referencia (430 Ω)

2. Una vez conocido el valor del sensor, obtener el valor de la temperatura es sencillo gracias a sus propiedades, ya sea mediante cálculos o mirando una tabla de referencia. En este caso, se realizan los cálculos en el programa de Arduino con la Ec. 11, empleando el método matemático directo descrito en [20]:

$$T_{RTD}(R_{RTD}) = \frac{-A + \sqrt{A^2 - 4B\left(1 - \frac{R_{RTD}}{R_0}\right)}}{2B} \qquad \text{Ec. 11}$$

donde para una Pt-100: $A = 3{,}9083 * 10^{-3}$ °C^{-1}, $B = -5{,}775 * 10^{-7}$ °C^{-2}, R_{RTD} es el valor de resistencia calculado previamente y R_0 es la resistencia de la RTD a 0 °C.

- **Registros del límite de error (03h-06h)**

Con estos registros se establecen los valores máximo y mínimo de los registros 01h-02h. Los resultados de la conversión RTD se comparan con los límites especificados en estos registros y, en caso de superarlos, generar los errores almacenados en los bits D7-D6 del registro de estado del error (07h). Los registros 03h y 05h marcan los límites superior e inferior del registro RTD MDBs (01h), mientras que los registros 04h y 06h hacen lo mismo con el registro RTD LSBs (02h).

REGISTER	HIGH FAULT THRESHOLD MSB (03h) REGISTER LOW FAULT THRESHOLD MSB (05h) REGISTER								HIGH FAULT THRESHOLD LSB (04h) REGISTER LOW FAULT THRESHOLD LSB (06h) REGISTER							
Bit	D7	D6	D5	D4	D3	D2	D1	D0	D7	D6	D5	D4	D3	D2	D1	D0
RTD Resistance Data	MSB	—	—	—	—	—	—	—	—	—	—	—	—	—	LSB	X
Bit Weighting	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	—
Decimal Value	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1	—

Tabla 6: Registros del límite de error (03h-06h) [18]

- **Registro de estado del error (07h)**

Indica si se ha detectado algún fallo y donde se ha producido, activando el bit correspondiente del registro. Es posible inicializar el registro a 0 activando el bit D1 del registro de configuración (00h). En la Tabla 7 se pueden observar los diferentes errores que se pueden detectar:

- En cada conversión A/D:
  - D7: Se produce cuando el valor de la conversión es igual o superior al establecido, en los registros (03h-04h).
  - D6: Se produce cuando el valor de la conversión es igual o inferior al establecido, en los registros (05h-06h).
- Al iniciar el ciclo de detección de errores en el registro de configuración (00h).
  - D5: Se produce si el voltaje en REFIN- es mayor que el 85% de V_{BIAS}
  - D4: Se produce si el voltaje en REFIN- es menor que el 85% de V_{BIAS} cuando el input FORCE- está abierto.
  - D3: Se produce si el voltaje en RTDIN- es menor que el 85% de V_{BIAS} cuando el input FORCE- está abierto.
- En cualquier momento:
  - D2. Se activa si el voltaje que reciben los pines FORCE+, FORCE2, RTDIN+, RTDIN-, o FORCE- es superior a V_{DD} o inferior a GND1.

D7	D6	D5	D4	D3	D2	D1	D0
RTD High Threshold	RTD Low Threshold	REFIN- > 0.85 x V_{BIAS}	REFIN- < 0.85 x V_{BIAS} (FORCE- open)	RTDIN- < 0.85 x V_{BIAS} (FORCE- open)	Overvoltage/ undervoltage fault	x	x

Tabla 7: Registro de estado del error (07h) [18]

4.5. Sistema de actuación

Como se ha explicado en el punto 3.3.2, los TRIAC pueden ser empleados para controlar el valor eficaz del voltaje de una señal de corriente alterna mediante control de fase. Se ha adquirido una placa para reducir el voltaje en alterna introducido en el calefactor. Esta placa emplea un TRIAC e incluye la circuitería necesaria para que sea fácilmente controlable desde Arduino mediante una señal PWM [21]. Además, cuenta con varios elementos adicionales aparte del TRIAC para asegurar así un correcto control de fase. En la Figura 41 se muestran estos elementos diferenciados.

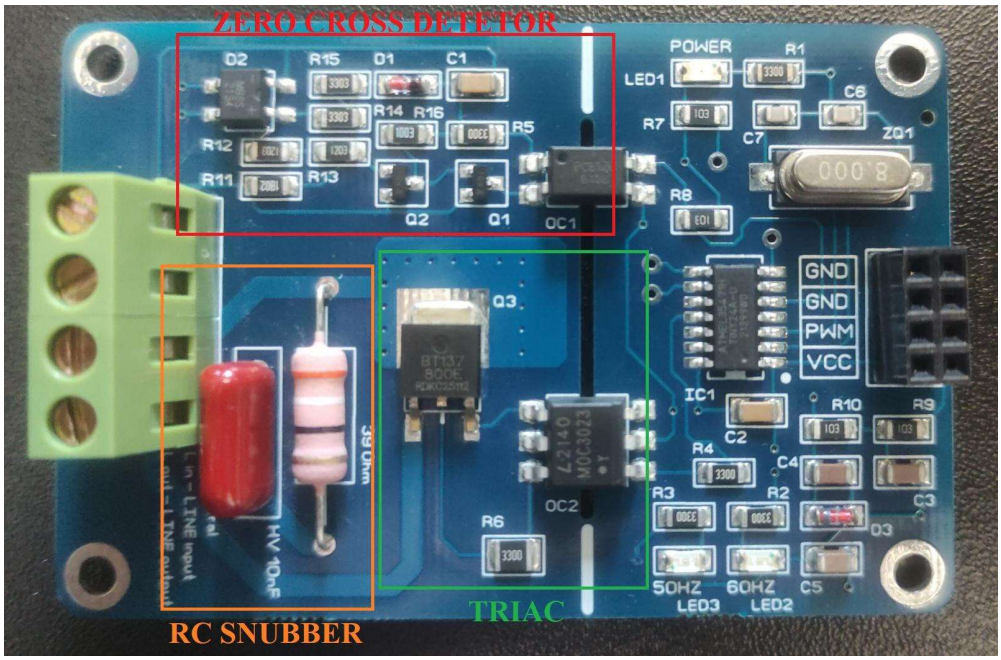

Figura 41: Placa de control de fase AC

Como se puede observar en la Figura 41, la placa está claramente separada en dos zonas, la de la izquierda contiene el circuito de potencia, mientras que la parte derecha contiene los elementos requeridos para el funcionamiento del microcontrolador que envía la señal de disparo al TRIAC (oscilador, filtro RC, LEDs, etc.). Este microcontrolador recibe a su vez una señal PWM desde Arduino, con la cual se le comunica la potencia deseada en la carga.

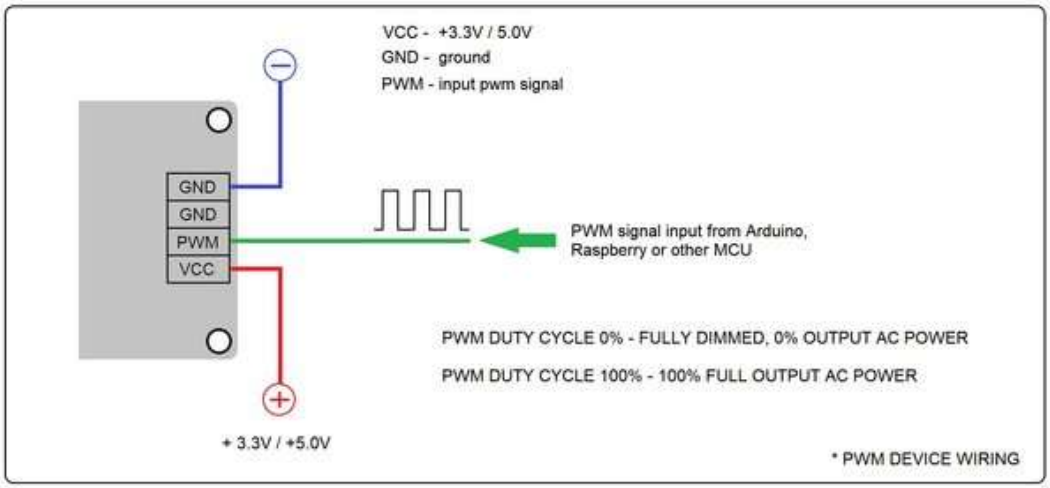

Figura 42: Conexión de la placa de control de fase con Arduino [21]

El funcionamiento de la señal PWM es simple: para emular el envío de una señal analógica mediante un pin digital (salida únicamente de 0 o 5 V) Arduino cuenta con una funcionalidad que le permite ajustar los tiempos de encendido y apagado de la señal digital, de este modo, se consigue una salida con un valor medio equivalente a una señal analógica con un rango de valores de 0 a 255, donde 0 equivale a 0V y 255 a 5V (Figura 43).

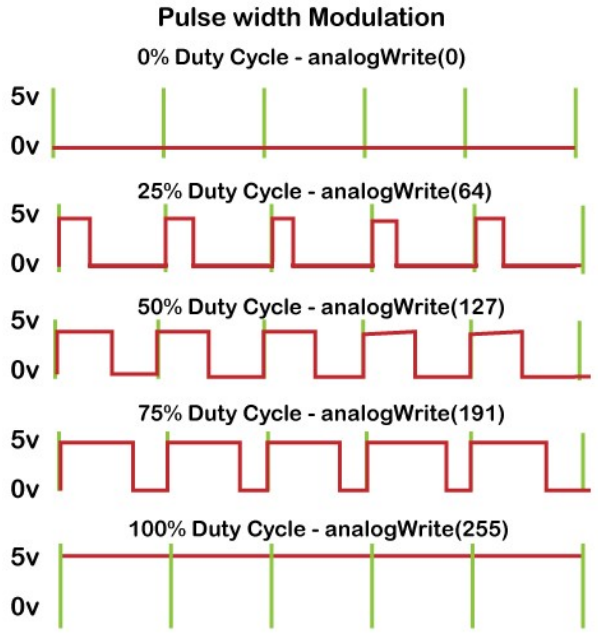

Figura 43: Señal PWM de Arduino

4.5.1. Detector de Cruce por Cero

A la hora de programar los disparos del TRIAC surge un problema: se busca que los ángulos de disparo sean precisos, por lo que el microcontrolador debe disponer de algún modo para saber con certeza cuándo empezar a contar. Para ello, se emplea un circuito conocido como Detector de Cruce por Cero. Como indica su nombre, este circuito alerta al controlador cada vez que una señal senoidal cambia de semiciclo, de modo que recibe la información necesaria para activar el TRIAC en el momento adecuado.

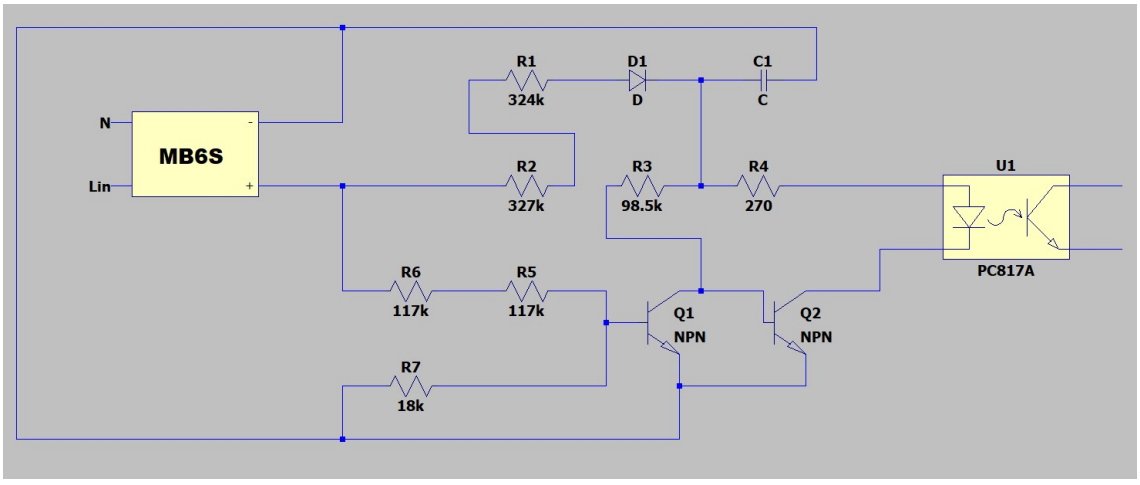

Figura 44: Esquemático del circuito detector de cruce por cero

Su funcionamiento es sencillo: utilizando un puente rectificador, se convierte la señal AC en DC y se conecta con un optoacoplador. Cuando la señal baja por debajo de su tensión mínima (es decir, la señal alterna se aproxima a 0 V), el LED en el interior del optoacoplador se apaga, mandando una señal digital al controlador. En la Figura 45 está representada la señal AC original y la entrada / salida del optoacoplador.

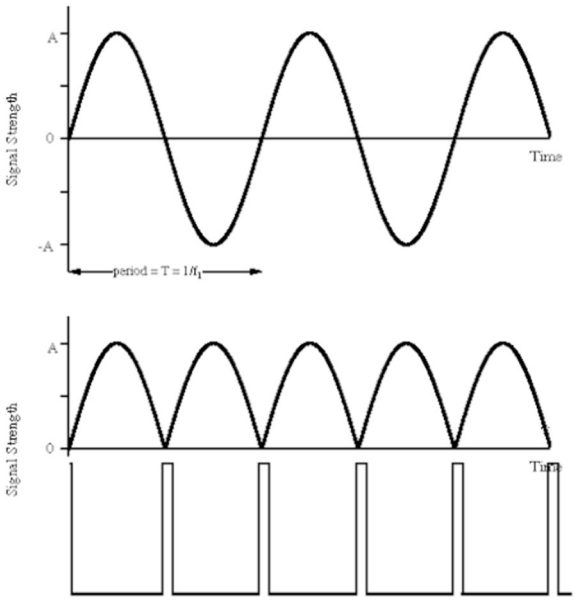

Figura 45: Funcionamiento del Detector de Cruce por Cero

4.5.2. Snubber RC

Se trata de un circuito habitualmente usado junto a TRIACs que cumple las siguientes funciones [22]:

- Mejora del apagado del TRIAC: cuando se apaga, la corriente pasa por cero y el voltaje de alimentación se vuelve a aplicar instantáneamente por la estructura. En ciertas condiciones, el componente no es capaz de bloquear este voltaje y se vuelve a encender espontáneamente. El snubber aumenta el tiempo de respuesta, reduciendo el pico producido por dV/dt_{OFF} y evitando que se encienda.
- Supresión de voltaje de transición rápida causado por el ruido, lo que puede llegar a causar el disparo del TRIAC.
- En caso de carga inductiva: limitación de sobrevoltaje en la conmutación de apagado. El snubber limita la pendiente del incremento de tensión, manteniendo el sobrevoltaje dentro de los límites permitidos por el TRIAC (Figura 46).

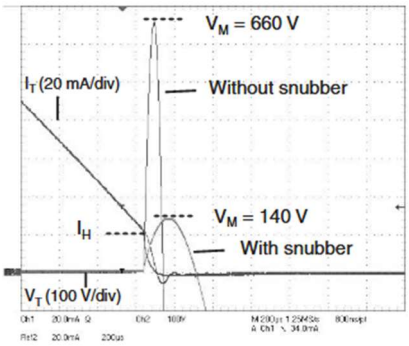

Figura 46:Sobrevoltaje durante el apagado del TRIAC con y sin circuito Snubber (R=10nF , R=2,7 kΩ) [22]

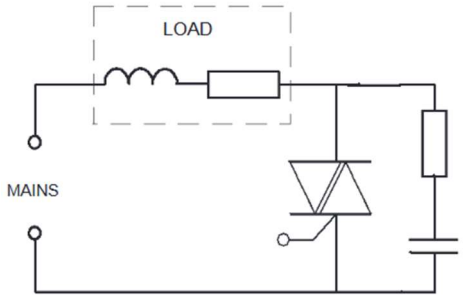

Figura 47: Esquema del circuito Snubber RC junto al TRIAC

Como se observa en la Figura 47, el circuito es sencillo, ya que consiste únicamente en una resistencia y condensador en serie conectados al TRIAC en paralelo. El diseño del circuito depende del factor de amortiguamiento (ξ): se buscan factores bajos, de ese modo, el circuito se puede optimizar reduciendo el valor del condensador, disminuyendo el coste del circuito. La elección del condensador se toma mediante la corriente RMS de la carga (Figura 48), suponiendo una carga puramente inductiva y que la tensión del voltaje que se vuelve a aplicar está limitada a $2\ V/\mu s$.

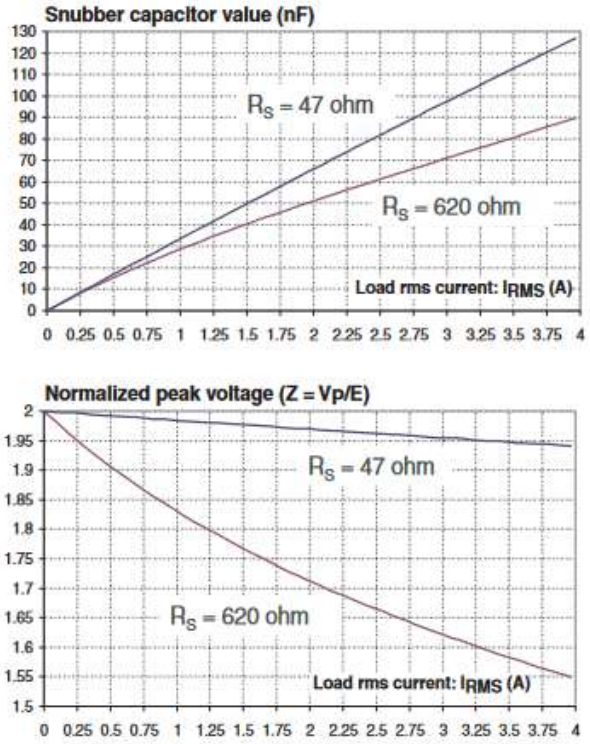

Figura 48: Valor del condensador del Snubber y pico de voltaje normalizado en función de la resistencia [22]

Para una corriente RMS de carga dada y en función de la resistencia empleada (47Ω o 620Ω), la ratio entre la pendiente ascendente de voltaje normalizado $(K = dV/dt_{OFF}/(E * \omega_0))$ y el factor de amortiguamiento ξ queda definida por:

$$\boxed{\frac{K}{\xi} = 2 * \frac{L}{R_S + R} * \frac{(dV/dt_{OFF})}{E}} \qquad \textit{Ec. 12}$$

donde $L = \frac{V_{RMS}}{I_{RMS} * 2 * \pi * f}$.

El factor de amortiguamiento ξ se obtiene mediante la Figura 49, el valor del condensador utilizando Ec. 13, mientras que el voltaje pico se obtiene con la Figura 50.

$$\boxed{C_S = 4 * \frac{L}{(R + R_S)^2} * \xi^2} \qquad \textit{Ec. 13}$$

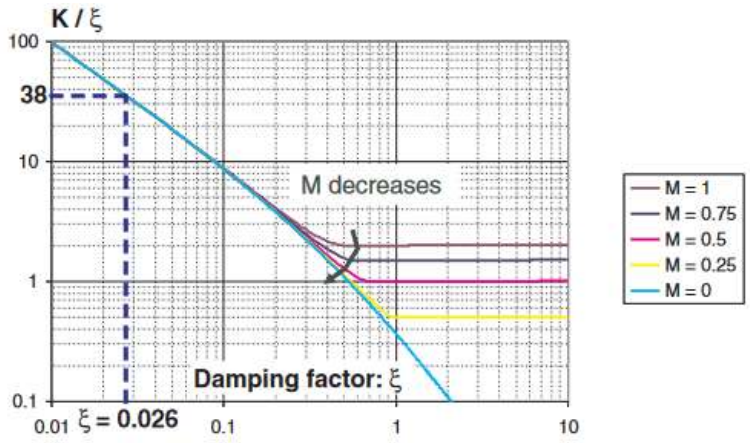

Figura 49: Relación entre pendiente ascendente del voltaje y el factor de amortiguamiento [22]

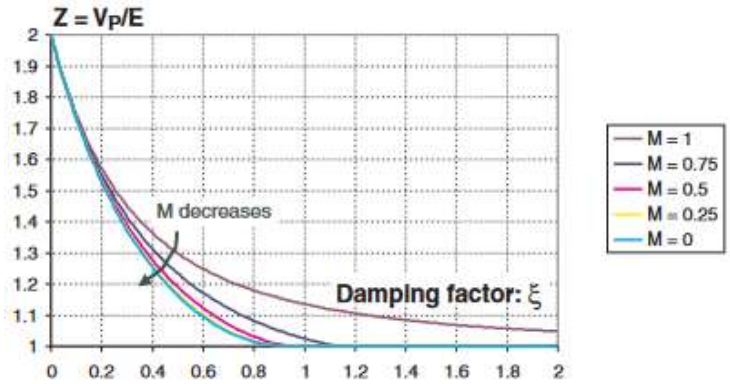

Figura 50: Relación entre Voltaje Pico (Z) con el factor de amortiguamiento [22]

4.5.3. Control del TRIAC

Se trata de un circuito sencillo, debido a la necesidad de separar el circuito de corriente alterna del de continua, el disparo del TRIAC se activa mediante un optotriac, un tipo de optoacoplador en el que el LED activa un fototriac en lugar de un fototransistor. Este disparo será producido por el microcontrolador de esta tarjeta en función del valor de la señal PWM que recibe del controlador PID de la placa Arduino.

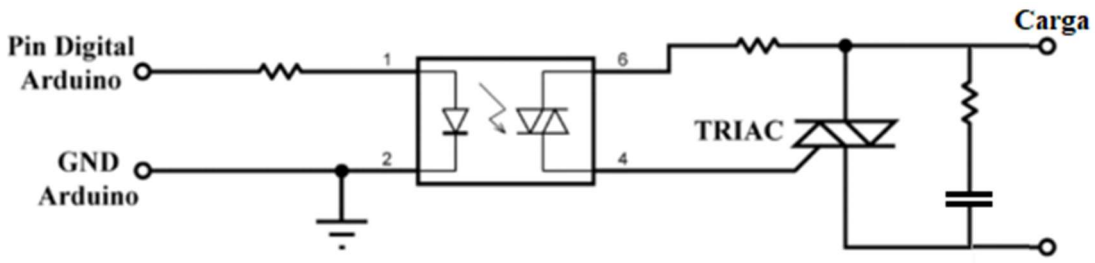

Figura 51: Esquema del circuito de control del TRIAC

4.6. Pantalla LCD

Este elemento se utiliza para la interfaz del usuario, empleándose para fijar la temperatura en el interior del biorreactor y la duración del programa, así como ofrecer información a tiempo real sobre la temperatura y tiempo restante. Se ha empleado un shield LCD de Arduino [23] que incluye múltiples botones, por lo que toda la interfaz necesaria está incluida en un sólo elemento. El shield usará siete pines digitales de Arduino para el control de la pantalla y un pin analógico para recibir la información sobre el botón pulsado.

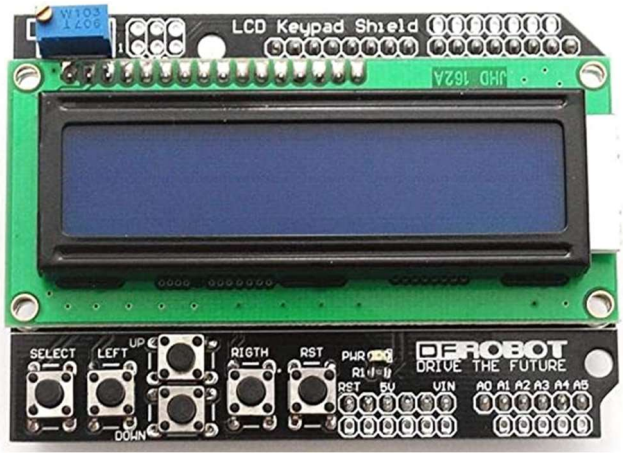

Figura 52: Shield LCD para Arduino

4.7. Calefactor

El modelo con el que se va a trabajar es un Applikon ZC8130HB07 [24]. Este elemento trabaja con unos valores nominales de 200W a 110 V_{AC}, por lo que será controlado por el sistema de actuación para regular la temperatura que produce en el biorreactor. Este modelo de silicona ha sido diseñado originariamente para su uso en un biorreactor de 7 litros, por lo que no tiene el tamaño adecuado para colocarse completamente alrededor de la pared exterior del biorreactor facilitado por la empresa. Su única conexión termina en un enchufe que se conecta a un transformador de 110 V_{AC}, por lo que es necesario reconfigurarla para que la señal eléctrica pase primero por la placa de control de fase.

Figura 53: Modelo del calefactor

5. Programación del Sistema

5.1. Introducción

En este apartado se va a hablar sobre el programa desarrollado para el sistema, las librerías empleadas y el entorno de desarrollo utilizado. El programa ha sido desarrollado como una máquina de estados, de modo que realice diferentes funciones en función del estado en el que se encuentra.

5.2. Entorno de desarrollo

Se ha empleado el software Arduino IDE, una aplicación desarrollada en java para facilitar la programación de los microcontroladores Arduino. Este programa cuenta con un editor de código que soporta los lenguajes C y C++, además de los mecanismos necesarios para compilar y enviar el programa al microcontrolador. La versión 2.0 del programa, dispone de un nuevo sistema de administración de librerías y una interfaz mejorada. También tiene la capacidad de recibir información a través del puerto serial al que se ha conectado el Arduino, lo que facilita controlar el correcto funcionamiento del programa.

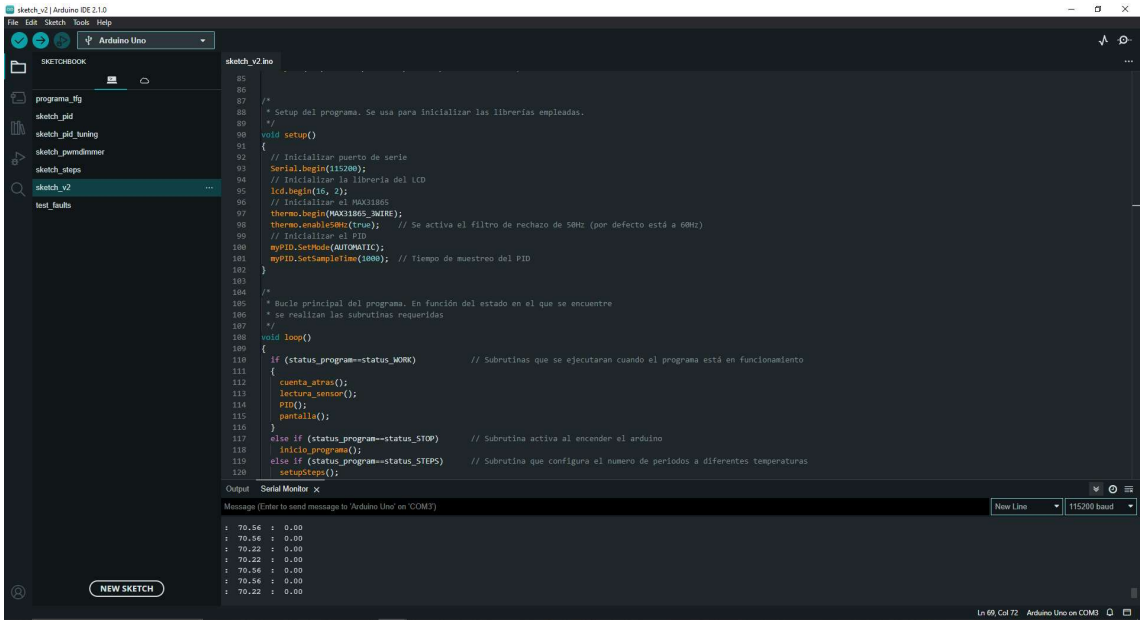

Figura 54: Entorno de desarrollo

5.3. Librerías empleadas

Se han empleado las siguientes librerías, correspondientes a los diferentes elementos hardware del sistema:

- LiquidCrystal: facilita la comunicación con la pantalla LCD.
- Adafruit_MAX31865: librería proporcionada por el fabricante para obtener el valor de la temperatura a partir del sensor Pt-100.
- PID_v1: Una de las librerías más extendidas para la programación de controladores PID.

5.3.1. LiquidCrystal

Se utiliza para enviar información al display LCD. Las funciones empleadas son las siguientes:

- *lcd.begin(16,2)*: inicializa la librería para un display 16x2.
- *lcd.setCursor(x,y)*: indica la posición del display donde se mostrará la información.
- *lcd.print(x,y)*: indica el mensaje o variable que se debe mostrar en el display.

5.3.2. Adafruit_MAX31865

Se comunica a través del bus SPI con el MAX31865 y convierte el valor de salida en su equivalente en grados Celsius. Además, permite filtrar el ruido en las medidas y ver el estado del sistema, pudiendo detectar errores. Las funciones empleadas son las siguientes:

- *thermo.begin(MAX31865_3WIRE)*: inicializa el sistema empleando la configuración de tres cables.
- *thermo.enable50Hz(true)*: activa el filtro de ruido a 50Hz, ya que por defecto se encuentra a 60Hz.
- *thermo.readRTD()*: lee el valor de 16 bits para la relación entre el RTD y la RREF generada por el MAX31865.
- *thermo.temperature(RNOMINAL,RREF)*: a partir los valores de la RREF y del sensor a 0ºC para calcular el valor de la temperatura junto al valor obtenido de *thermo.read().* El método empleado para el cálculo es el descrito en el apartado 4.4.3.
- *thermo.readFault()*: lee el registro 07h del MAX31865, que indica si se ha detectado algún error.

5.3.3. PID_v1

Implementa un algoritmo PID en el microcontrolador (tanto el tradicional como el PonM) y todas las funciones necesarias para configurarlo: tiempo de muestreo, valores de Output máximos y mínimos, modo de funcionamiento, etc.

- *PID(&Input, &Output, &Setpoint, Kp, Ki, Kd, P_ON_M, Direction)*: genera el control PID con los parámetros deseados, empleando el método PonM.
- *myPID.SetMode(AUTOMATIC)*: establece el modo de funcionamiento del PID; en *"MANUAL"* el controlador está apagado, la respuesta la indica el usuario.
- *myPID.SetSampleTime(1000)*: indica el tiempo de muestreo del controlador (en ms). Se utiliza un valor de 1 segundo para que sea igual al periodo de toma de medidas del sensor.
- *myPID.Compute()*: contiene el algoritmo PID, obtiene un nuevo output a la frecuencia indicada por *SetSampleTime()*.

5.4. Estructura del programa

Como se ha mencionado, el programa se ha dividido en cuatro estados que se activarán y desactivarán en función de la etapa en la que se encuentra el programa y las entradas del usuario. En este apartado se va a explicar el funcionamiento de cada uno de estos estados, las funciones que cumplen dentro del programa y las condiciones necesarias para pasar de uno a otro. Estos estados y como el programa va cambiando entre ellos se visualizan en la Figura 55.

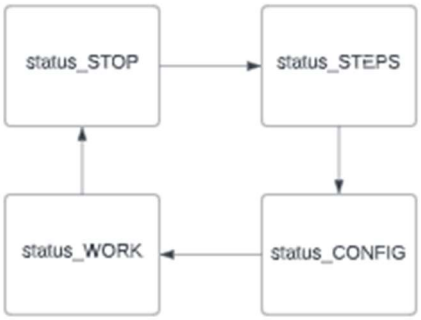

Figura 55: Diagrama de flujo de la máquina de estados

5.4.1. status_STOP

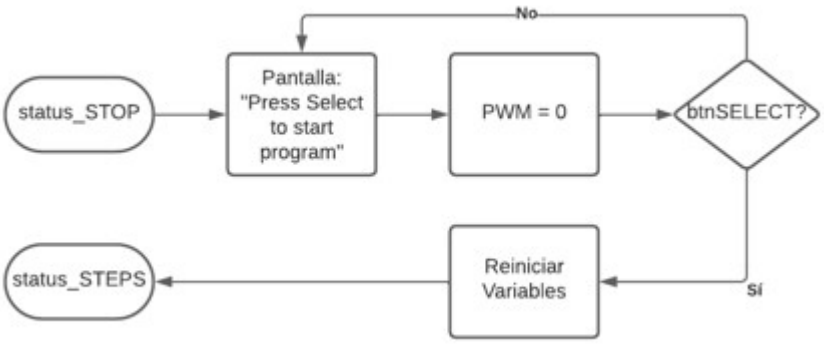

Figura 56: Diagrama de bloques de estado STOP

Se trata del estado por defecto del prototipo, en el que se encuentra al encenderse. Por tanto, no cuenta con muchas funciones. Únicamente se encarga de reiniciar las distintas variables empleadas en ejecuciones anteriores y se asegura de que el calefactor esté inicialmente apagado ($PWM = 0$). Para hacer la interfaz más atractiva al usuario, el display LCD cuenta con un mensaje con instrucciones sobre como empezar a configurarlo, en este caso, pulsando el botón *SELECT* del shield LCD, cambiando así el programa al siguiente estado.

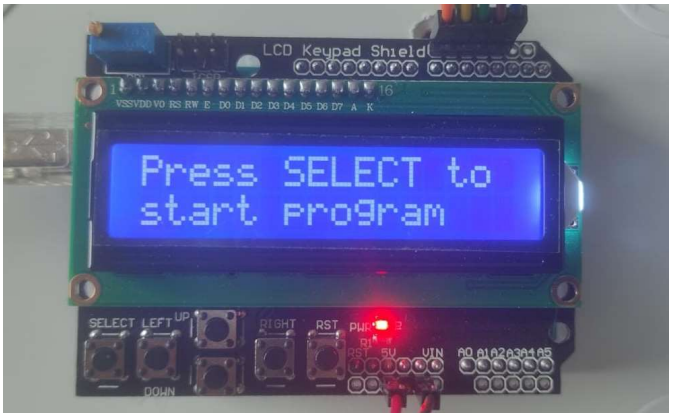

Figura 57: Aspecto del display LCD en el estado STOP

5.4.2. status_STEPS

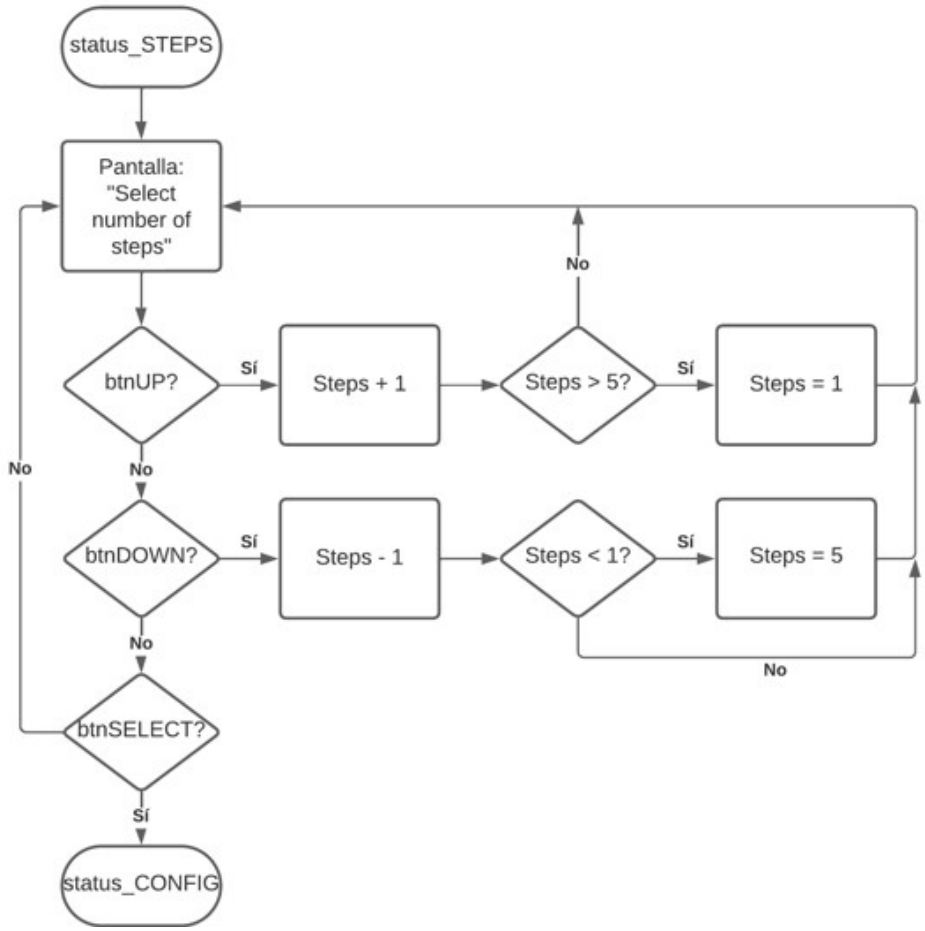

Figura 58: Diagrama de bloques del estado STEPS

A petición de la empresa, se implementó esta opción para la configuración del programa, que permite al usuario elegir el número de etapas con las que contará el programa final, de modo que sea posible programar diferentes duraciones para distintos escalones de temperatura. Su funcionamiento es sencillo: pulsando los botones *UP / DOWN* del shield LCD se aumenta / disminuye el número de etapas, con un número máximo de cinco.

Figura 59: Aspecto del shield LCD en el estado STEPS

5.4.3. status_CONFIG

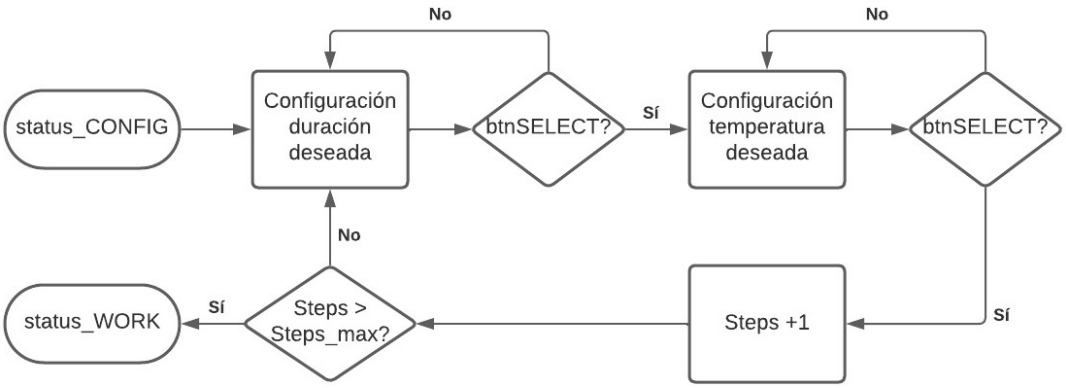

Figura 60: Diagrama de bloques del estado CONFIG

En este estado se configura la duración y temperatura deseadas. Para ello, se hace uso de los botones del shield LCD para modificar los dígitos de uno en uno. La configuración de la duración sigue un formato *HHH:MM:*SS y se realiza pulsando los botones *UP / DOWN* para modificar el valor del digito seleccionado, que parpadea para facilitar su visualización. Presionar *LEFT / RIGHT* cambia la selección al digito a su izquierda / derecha. Una vez se tiene la duración deseada, presionar *SELECT* comienza la configuración de la temperatura deseada, que sigue el mismo funcionamiento con un formato de dos dígitos y un decimal. La etapa que se está configurando se expresará por el numero correspondiente a la derecha de "Time", como se observa en la Figura 61.

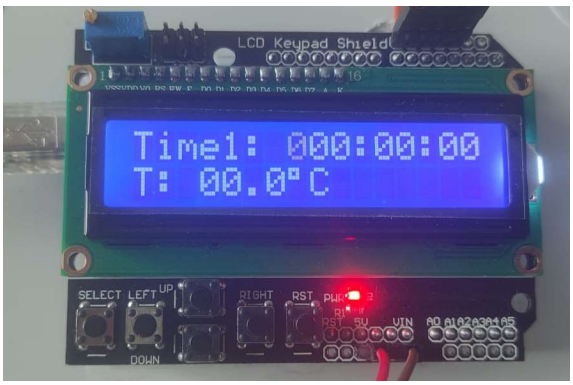

Figura 61: Aspecto del shield LCD durante el estado CONFIG

5.4.4. status_WORK

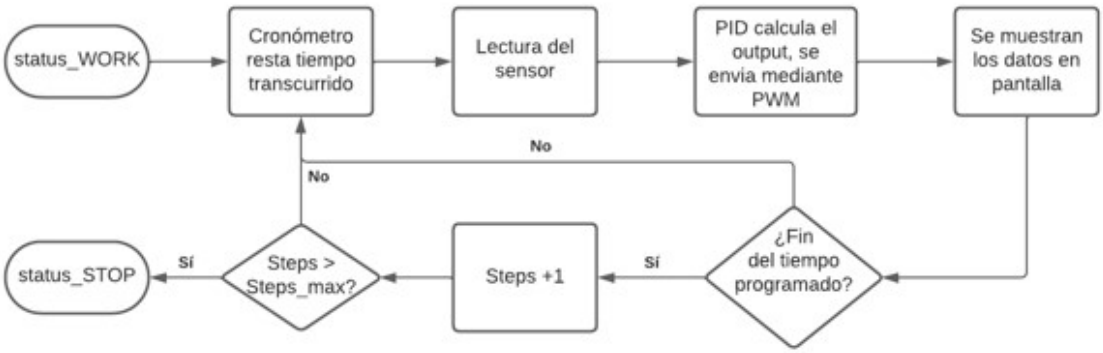

Figura 62: Diagrama de bloques del estado WORK

Una vez configurado el programa deseado, se llega a este estado, en el que el sensor RTD comienza a enviar los valores de la temperatura interna al controlador PID, que se encarga de calcular la potencia necesaria en el calefactor, enviándola en forma de una señal PWM a la tarjeta de control de fase AC. Los datos a tiempo real de tiempo restante y temperatura son mostrados en pantalla, junto a la temperatura objetivo, como se observa en la Figura 63. Además, a petición de la empresa, se añadió una funcionalidad al botón *UP*: al presionarlo, el valor de tiempo es sustituido por el valor del output PWM del controlador PID, permitiendo así al usuario saber que está sucediendo en su interior.

Los valores de temperatura y output del PID son calculados con un periodo de muestreo de 1 segundo, ya que al tratarse de un sistema que cambia lentamente no es necesario una frecuencia de muestreo mayor. Una vez la cuenta atrás llega a 0, se cambia a la siguiente etapa programada, o en caso de ser la última, se da por finalizado el programa, volviéndose al estado *status_STOP*.

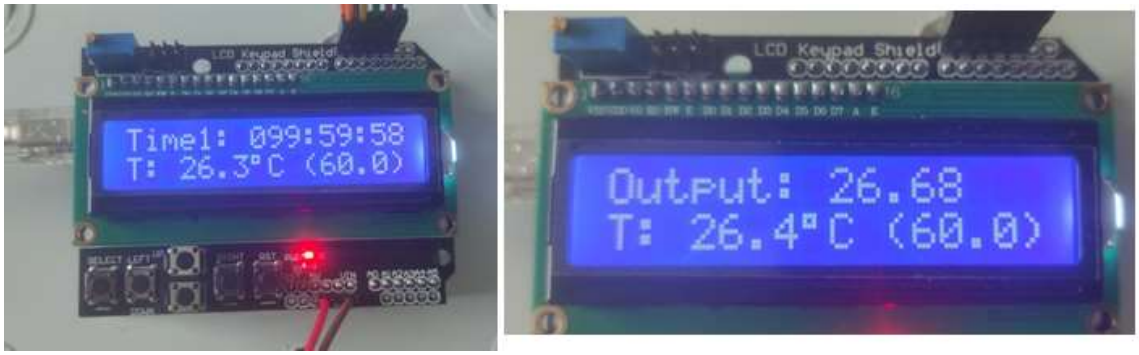

Figura 63: Aspecto del shield LCD durante el estado WORK

6. Pruebas y Resultados

En este capítulo se trata todo lo referido al montaje final del prototipo, en el que se incluyen el diseño mecánico y las pruebas de sintonía para lograr un funcionamiento óptimo en el controlador PID. También se van a tratar las pruebas realizadas por la empresa en sus procesos y los resultados de estas.

6.1. Montaje final

Debido a la necesidad de aislar el circuito de potencia (110 V_{AC}) del usuario, se decidió emplear una caja eléctrica *Mureva BOX ENN05007* [25], fabricada por Schneider Electric, mostrada en la Figura 64. Este modelo de caja tiene unas dimensiones 105x80x150 mm y se encontraba disponible en la empresa. En su interior cuenta con suficiente espacio para incluir tanto la tarjeta de control de fase AC como la tarjeta Adafruit Max31865, lo que evita la necesidad de tener que adquirir material adicional.

Figura 64: Caja eléctrica empleada en el prototipo

Para optimizar la organización de los cables en el interior de la caja se optó por separar el interior de la caja en dos partes, una para el acondicionamiento y otra para el actuación. Para el acondicionamiento se utilizó una placa para montaje por soldadura en la que se incorporó el Adafruit Max31865 (Figura 65), facilitando también la extracción de dicha tarjeta en caso de necesitar reemplazarla. Este circuito recibe su alimentación desde la placa Arduino, y de ella se alimenta el acondicionamiento del sensor y la parte de control de la tarjeta de control de fase, que se ubica en el otro extremo de la caja.

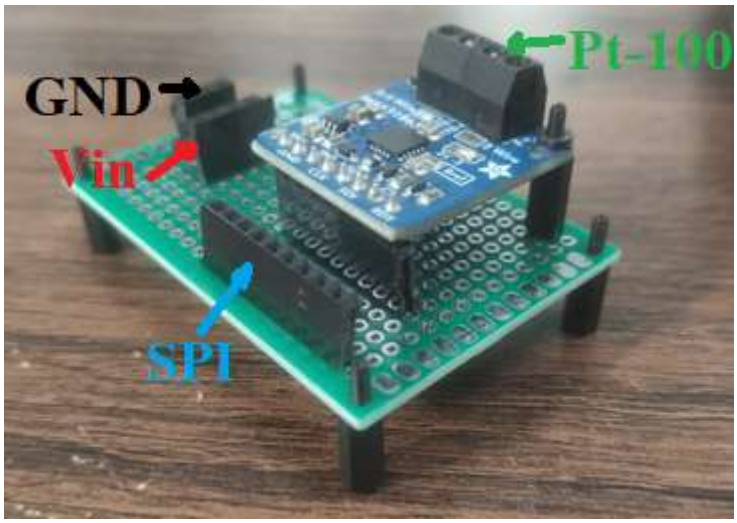

Figura 65: Placa de circuito construida

Todo el diseño mecánico ha sido realizado de modo que los diferentes componentes puedan ser fácilmente reemplazados o reutilizados en otros proyectos. Para fijar ambas tarjetas al interior de la caja eléctrica se añadió silicona a los soportes de las tarjetas, de este modo se mantiene la posibilidad de sacarlas de la caja desenroscando los soportes. La placa Arduino se instaló en el exterior de la caja, encima de la tapa, de modo que el usuario tuviera acceso al shield LCD para la configuración del programa (Figura 66).

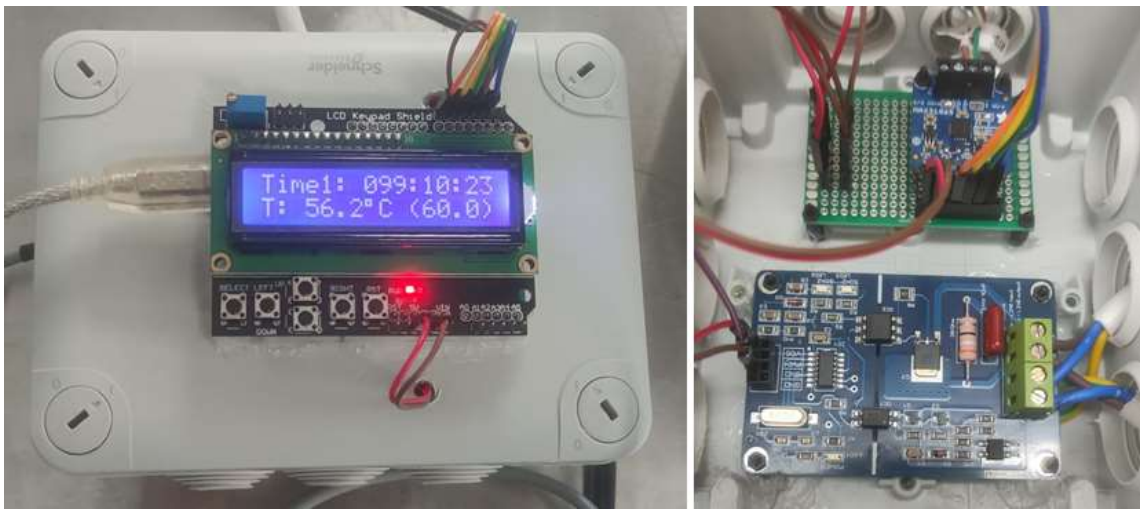

Figura 66: Exterior e interior de la caja eléctrica

Para poder acomodar el shield LCD, el resto de los componentes emplearon los siguientes pines, tal y como se observa en la Figura 67:

- **#13**: CLK, conectado al Adafruit MAX31865.
- **#12**: SDO, conectado al Adafruit MAX31865.
- **#11**: SDI, conectado al Adafruit MAX31865.
- **#2**: CS, conectado al Adafruit MAX31865.

- **#3**: PWM, conectado a la tarjeta de control de fase.

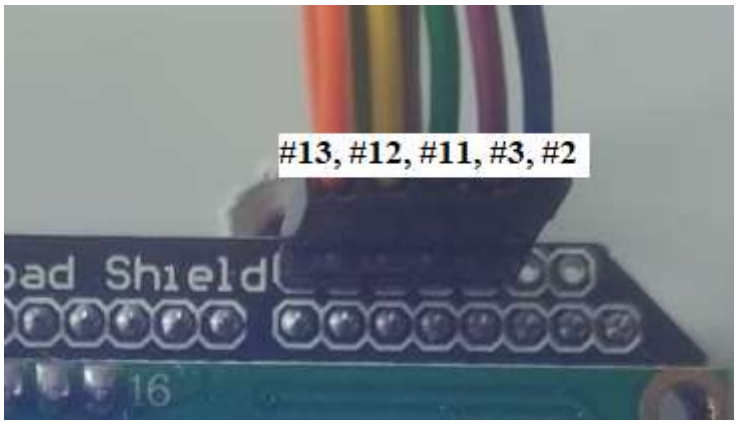

Figura 67: Reparto de los pines no utilizados por el shield LCD

Para la alimentación del calefactor, se empleó una regleta multicontactos que había sido modificada para estar conectado a la tarjeta de control de fase, de acuerdo con la conexión de la Figura 68. De este modo, la señal de entrada es regulada al valor deseado antes de llegar al calefactor.

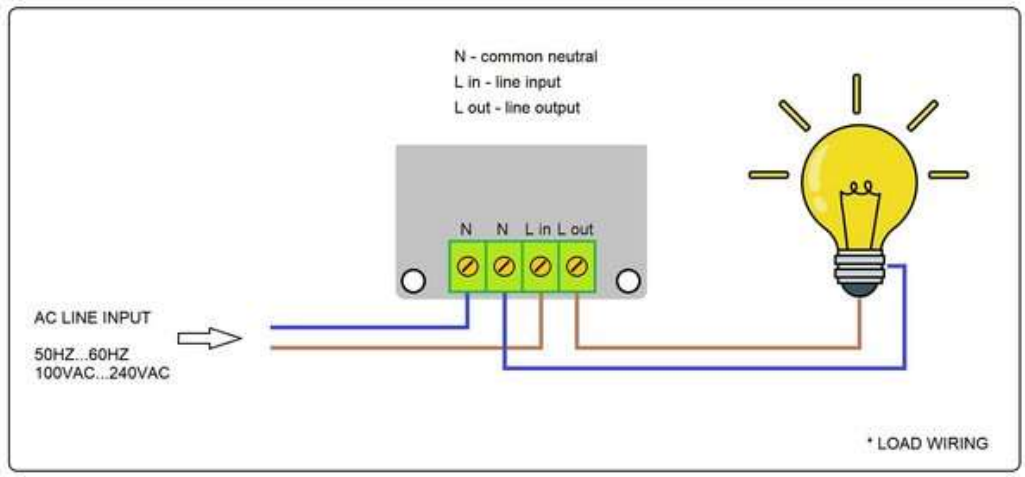

Figura 68: Conexión de la tarjeta de control de fase AC [21]

Las ventajas de usar una regleta en vez de conectar el calefactor directamente a la tarjeta de control de fase son varias: permite a la empresa desconectar el calefactor fácilmente en caso de que se desee usar en aplicaciones diferentes. Además, al contar con múltiples contactos, permite la conexión de varios calefactores en caso de que se quieran usar modelos de menor tamaño. En la Figura 69, se encuentra el aspecto final del prototipo: se puede observar la conexión de la regleta y del sensor con la caja eléctrica, mientras que el Arduino es alimentado por un adaptador de 5V conectado a un enchufe.

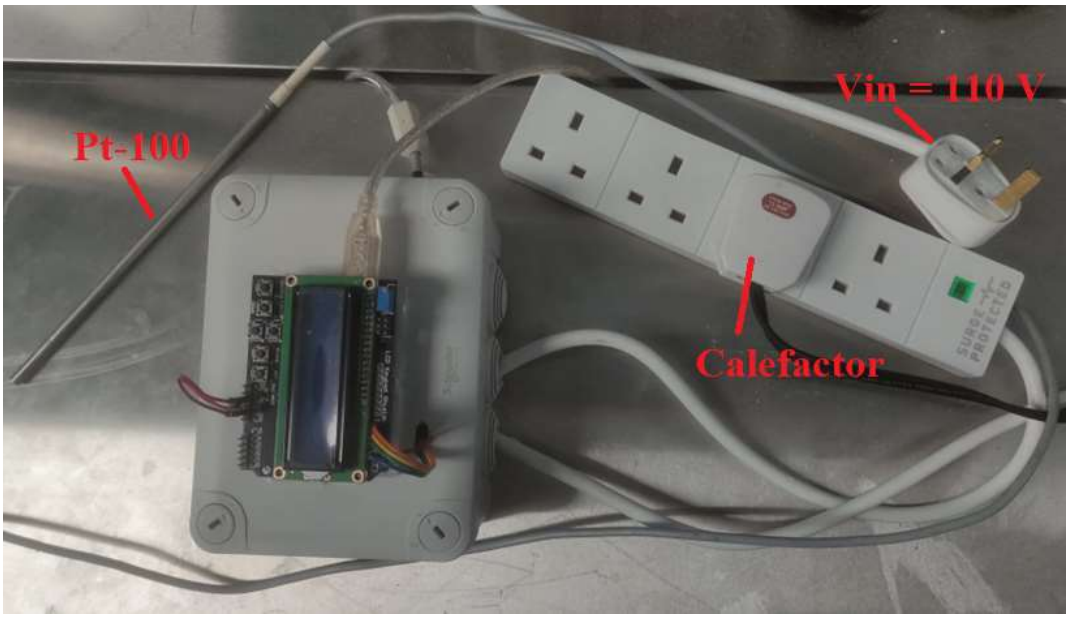

Figura 69: Aspecto final del prototipo

6.2. Pruebas de sintonía y estabilidad

Las siguientes pruebas fueron realizadas empleando el biorreactor para el que estaba destinado el uso del prototipo. Como ya se ha comentado anteriormente, se trata de un tanque con una capacidad máxima de 12L. Durante la realización de estas pruebas, se encontraba relleno de agua, con el montaje que se observa en la Figura 70. Para mantener una temperatura interna uniforme, el agua era agitada con el motor que se observa en la parte superior de la imagen, además, se utilizó una capa de aislamiento para aumentar la eficiencia del calefactor.

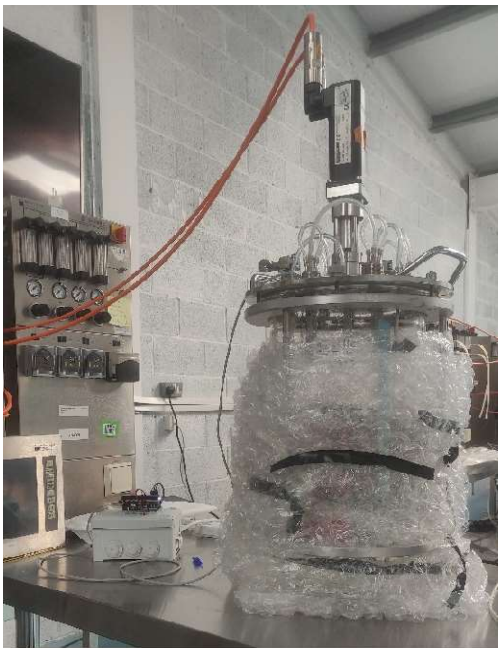

Figura 70: Montaje durante las pruebas

Una vez realizado el montaje necesario para realizar las pruebas, se realizó una primera prueba para comprobar su correcto funcionamiento, asignándole al controlador los siguientes valores arbitrarios: $K_p = 15;\ K_i = 0{,}15; K_d = 0$

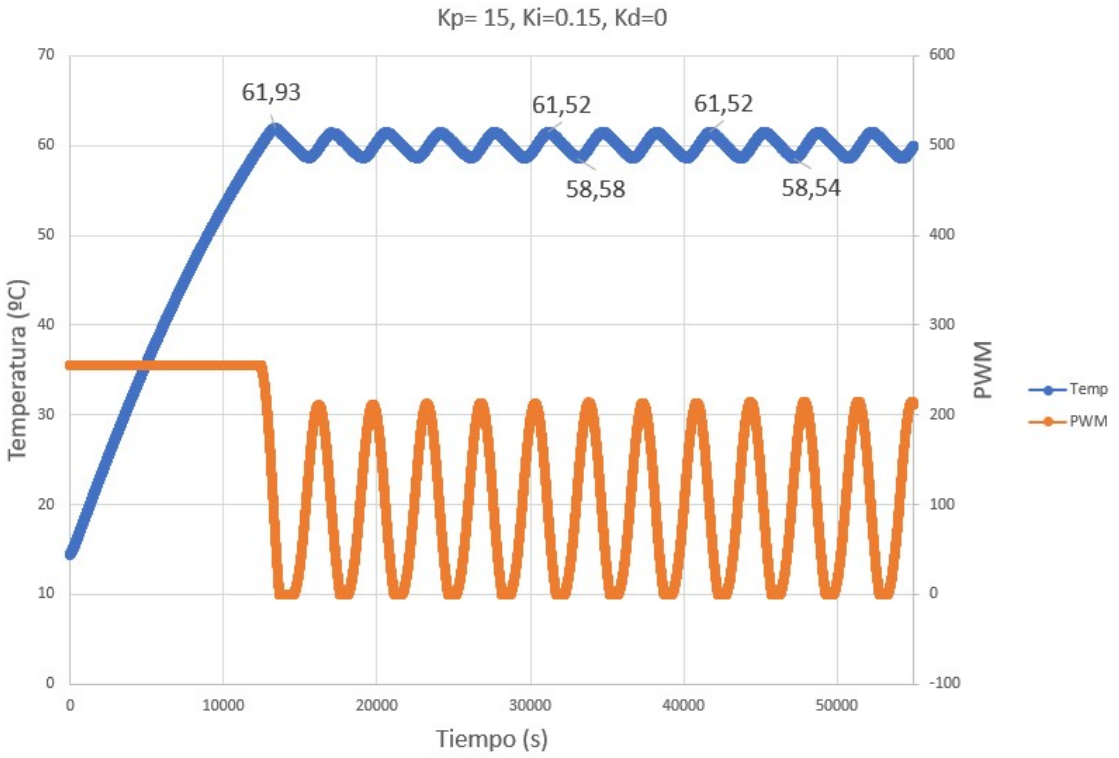

Figura 71: Respuesta del PID (Kp=15; Ki=0,15; Kd=0)

Como se puede observar en la Figura 71, el sistema tiene una respuesta oscilante con una variación entre 61,5ºC y 58,5ºC. La posible causa de esta oscilación es que el valor de K_i es demasiado alto. A continuación, se intentó emplear los métodos del relé y Ziegler-Nichols para comparar sus resultados, lo que debería facilitar obtener unos parámetros más adecuados en caso de que funcionasen correctamente.

- **Método del relé**

Las pruebas de sintonía se iniciaron empleando el método del relé descrito en el apartado 3.4.3, obteniéndose los siguientes valores para P_{cr} y K_{cr}:

P_{cr}	500
K_{cr}	0,4

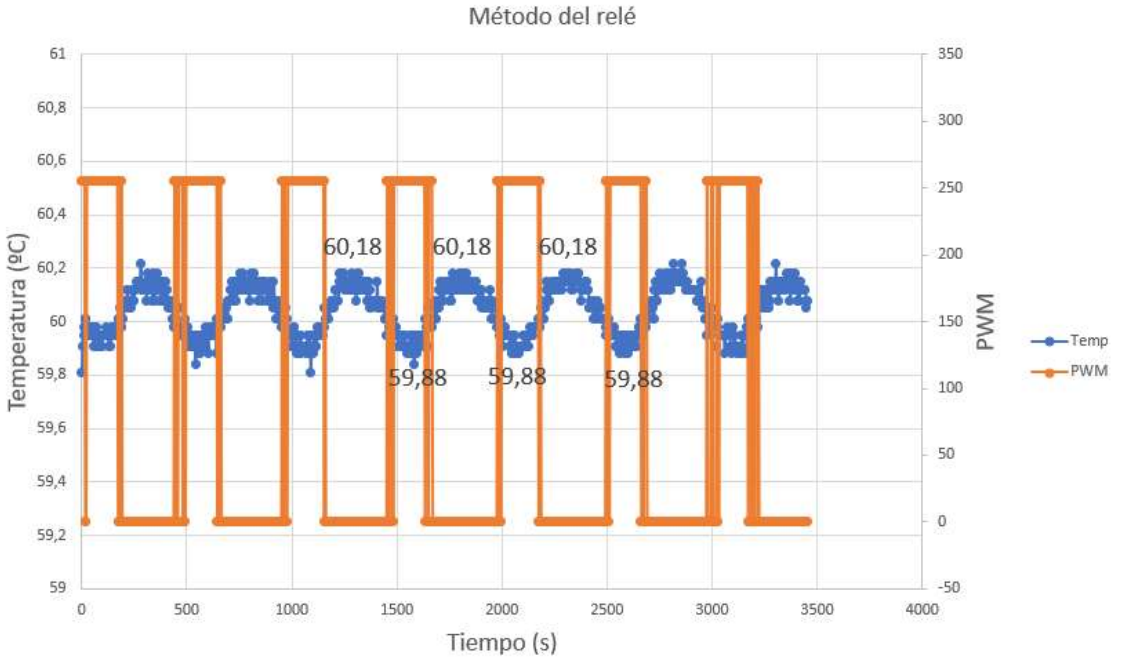

Figura 72: Resultados del método del relé

- **Método de Ziegler-Nichols**

Se aplica también el método de Ziegler-Nichols, en el cual se fijan $K_i = 0$ y $K_d = 0$. El valor del parámetro proporcional con el que se generaban oscilaciones sostenidas en el régimen permanente fue $K_p = 1000$. De este modo, los parámetros K_{cr} y P_{cr} son los siguientes:

P_{cr}	498
K_{cr}	1000

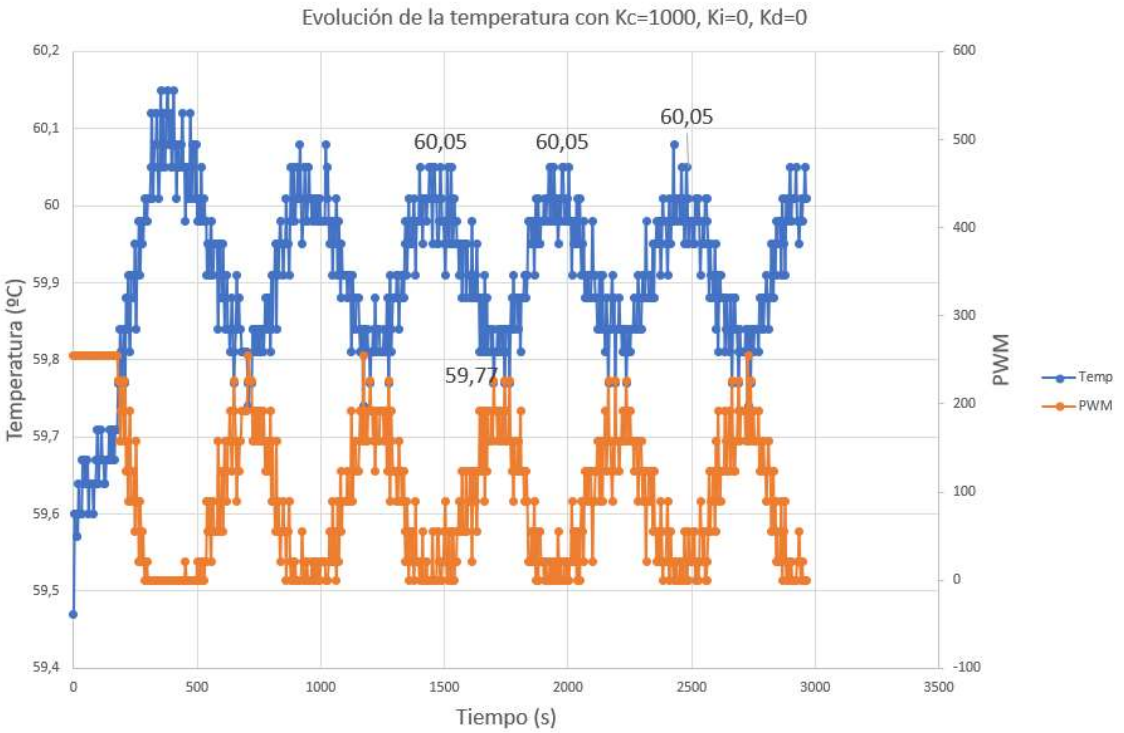

Figura 73: Resultados del método de Ziegler-Nichols

Como se observa, P_{cr} tiene un valor idéntico al emplear ambos métodos de sintonía. Sin embargo, el valor de K_{cr} difiere bastante, por lo que se optó por realizar las siguientes pruebas de sintonía con un valor de $K_i = 0{,}01$, similar al determinado por la Tabla 1, mientras se emplea un valor más conservador como $K_{cr} = 10$. Se ha decidido observar el comportamiento del controlador asignando el valor $K_p = 10$, que será ajustado mediante prueba y error para que el sistema tenga la respuesta deseada.

- $\boldsymbol{K_p = 10;\ K_i = 0,01;\ K_d = 0,5}$

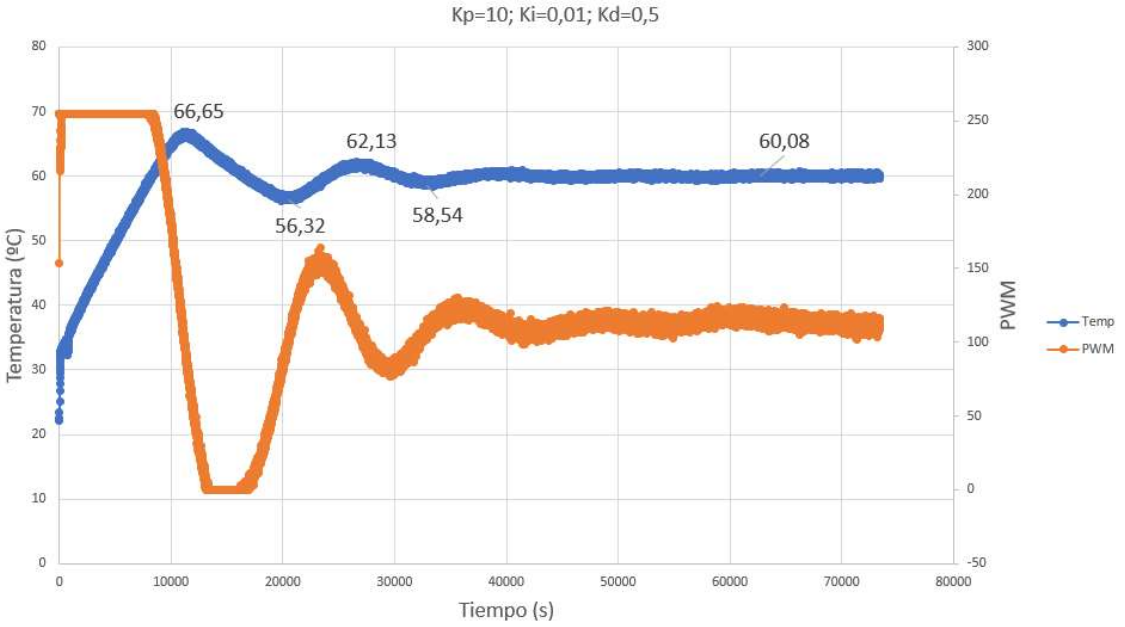

Figura 74: Respuesta del PID (Kp=10; Ki=0,01; Kd=0,5)

Se observa una clara mejoría con respecto a la prueba inicial ya que el sistema deja de presentar las oscilaciones existentes en el estacionario. Sin embargo, sigue teniendo varios problemas claros como la elevada sobreoscilación inicial de $66{,}65$ºC, la cual es inaceptable, ya que podría suponer un daño irreparable en ciertos procesos. Para corregir este problema, se opta por cambiar el modelo del controlador PID tradicional por un PID PonM (Apartado 3.4.2), lo que debería permitir eliminar completamente la sobreoscilación.

- $K_p = 10; K_i = 0,01; K_d = 5$ **(Proportional on measurement)**

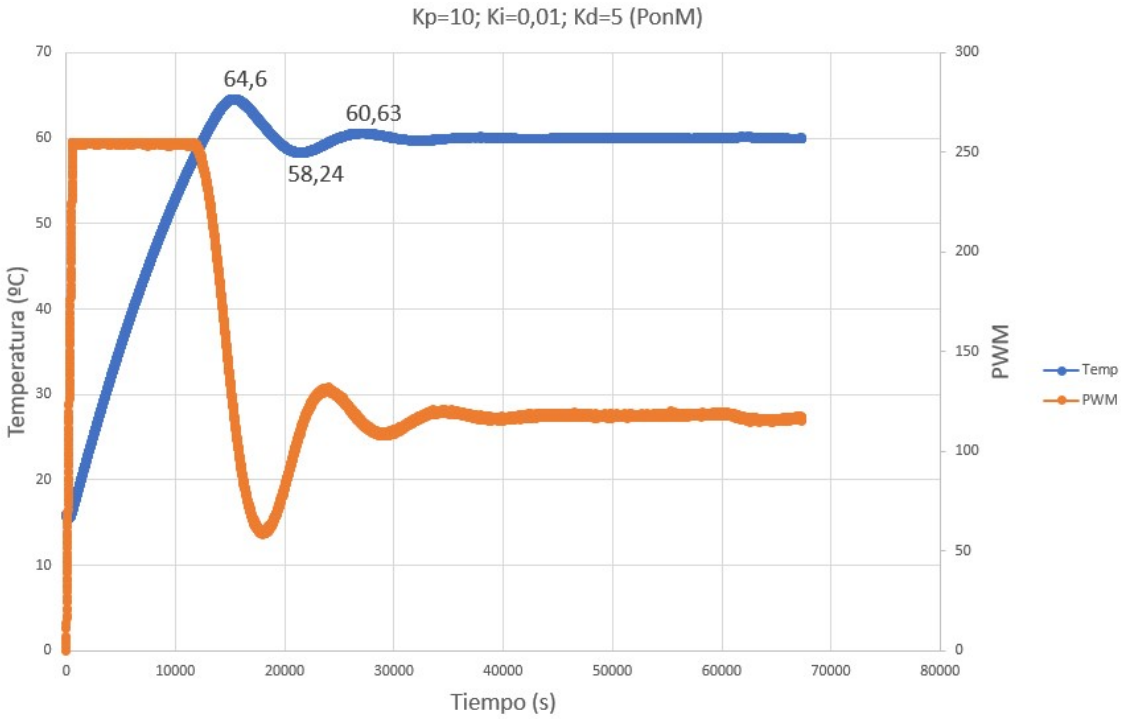

Figura 75: Respuesta del PID (Kp=10; Ki=0,01; Kd=5; P on M)

En la Figura 75 se observa una reducción en el oscilamiento de la respuesta con respecto al caso anterior, en el que el proporcional dependía del error, sin embargo, se sigo produciendo una sobreoscilación inaceptablemente alta. Como ya se ha explicado en el apartado 3.4.2, la gran ventaja de emplear un control con el proporcional en función de la medida es que permite eliminar la sobreoscilación en procesos integradores. De acuerdo con la Figura 25, la respuesta actual podría ser la resultante de los parámetros K_p, $K_i * 2$ o $K_p/2, K_i$. Se ha asumido que se trata del primer caso por lo que, para la siguiente prueba, se reducirá K_i a la mitad.

- $\boldsymbol{K_p = 10; K_i = 0,005; K_d = 5}$ $\boldsymbol{(Proportional\ on\ Measurement)}$

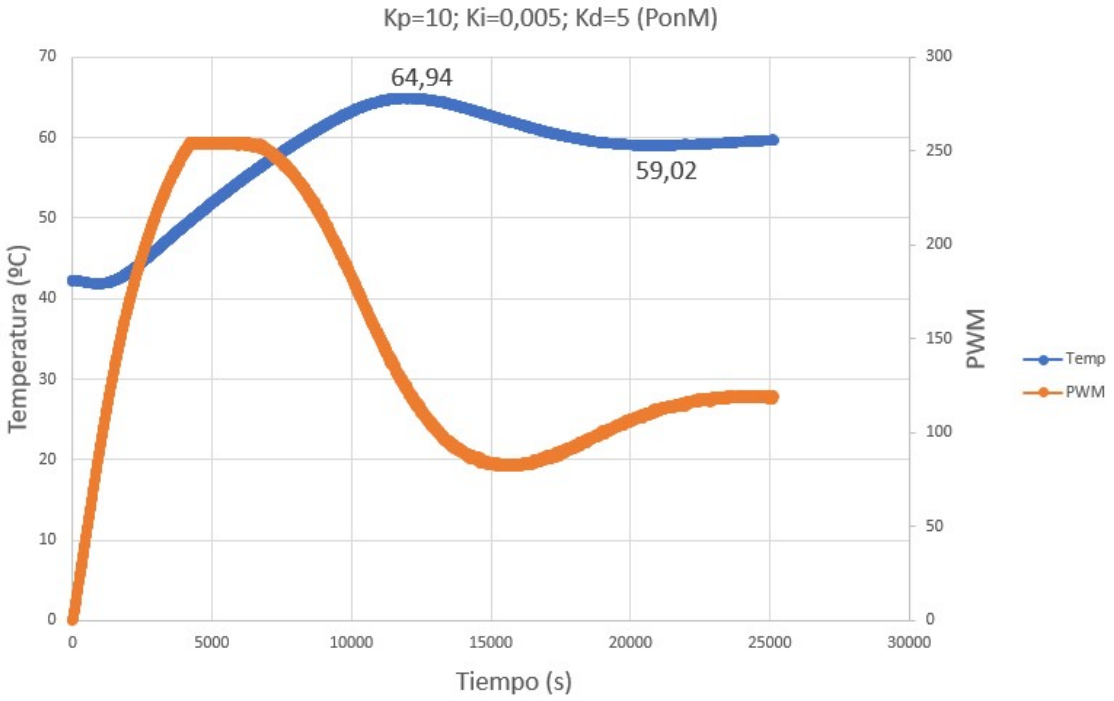

Figura 76: Respuesta del PID (Kp=10; Ki=0,005; Kd=5; P on M)

Con estos nuevos parámetros no se ha eliminado la sobreoscilación como se esperaba, lo que indica que los parámetros anteriores se correspondían al segundo caso que se había estudiado. Comparando la Figura 76 con la tabla de la Figura 25, se observa como la respuesta en esta prueba se asemeja a la respuesta en la posición $K_p/2$, $K_i/2$ de la tabla, por tanto, para eliminar la sobreoscilación es necesario doblar el valor de K_p.

- $\boldsymbol{K_p = 20; K_i = 0,005; K_d = 5}$ $\boldsymbol{(Proportional\ on\ Measurement)}$

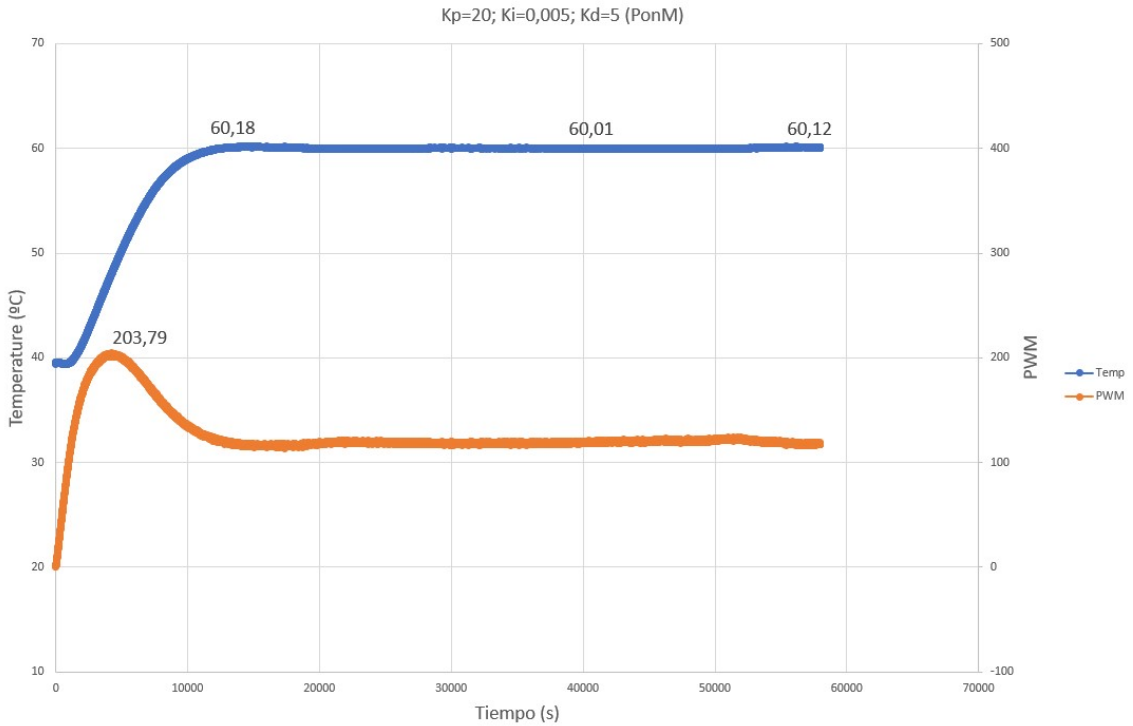

Figura 77: Respuesta del PID (Kp=20; Ki=0,005; Kd=5; P on M)

Como se puede observar, el comportamiento de la respuesta en la Figura 77 es bastante bueno: la sobreoscilación que se genera es despreciable y el valor de la temperatura se mantiene perfectamente estable durante toda la duración de la prueba. Sin embargo, se aprecia que la señal PWM no llega en ningún momento a su potencia máxima (255), por lo que el tiempo de estabilización es más lento de lo que podría llegar a ser. Para tratar de optimizar la velocidad de la respuesta, se prueba a aumentar el valor de K_i, lo que debería mover el control a la posición K_p, K_i de la Figura 25.

- $\boldsymbol{K_p = 20; K_i = 0,01; K_d = 5}$ $\boldsymbol{(Proportional\ on\ Measurement)}$

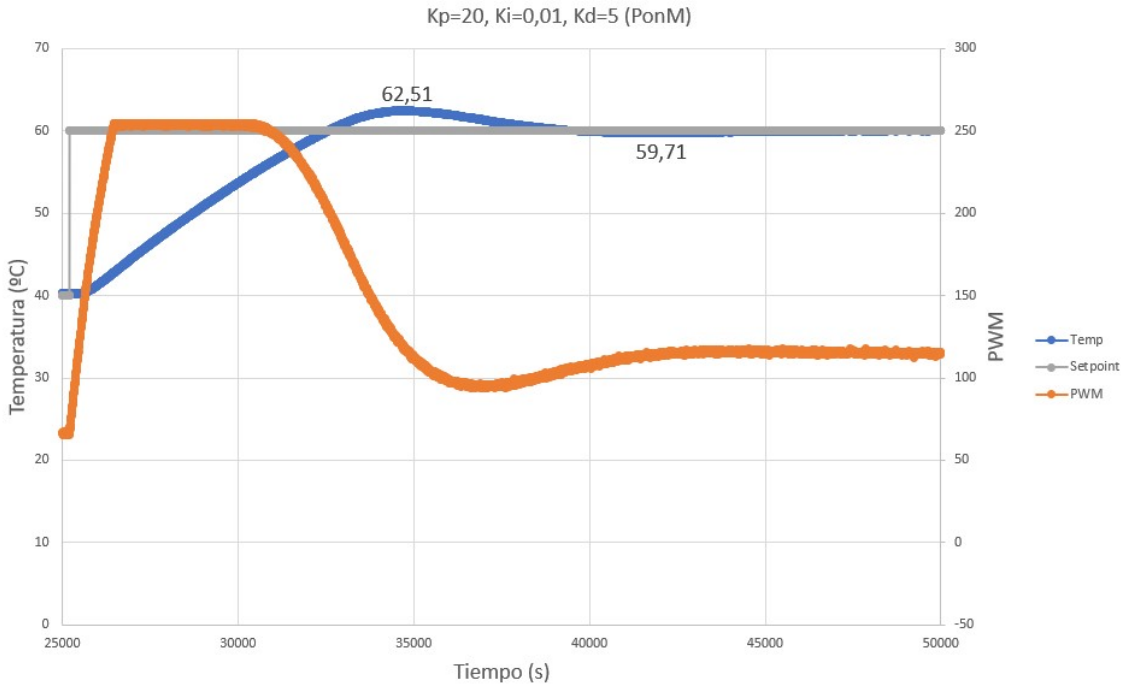

Figura 78: Respuesta del PID (Kp=20; Ki=0,01; Kd=5; P on M)

Como se observa en la Figura 78, aumentar el valor de K_i consigue que el output del calentador llegue a su valor máximo, reduciendo significativamente el tiempo de calentamiento. Sin embargo, el controlador no disminuye el valor de la salida a tiempo, por lo que se genera una sobreoscilación considerable, lo que hace que estos parámetros no sean aceptables para la aplicación que se va a dar al sistema. A continuación, se trata de reducir el valor de K_i ligeramente, para tratar de conseguir el comportamiento deseado.

- $\boldsymbol{K_p = 20; K_i = 0,0075; K_d = 5}$ $\boldsymbol{(Proportional\ on\ Measurement)}$

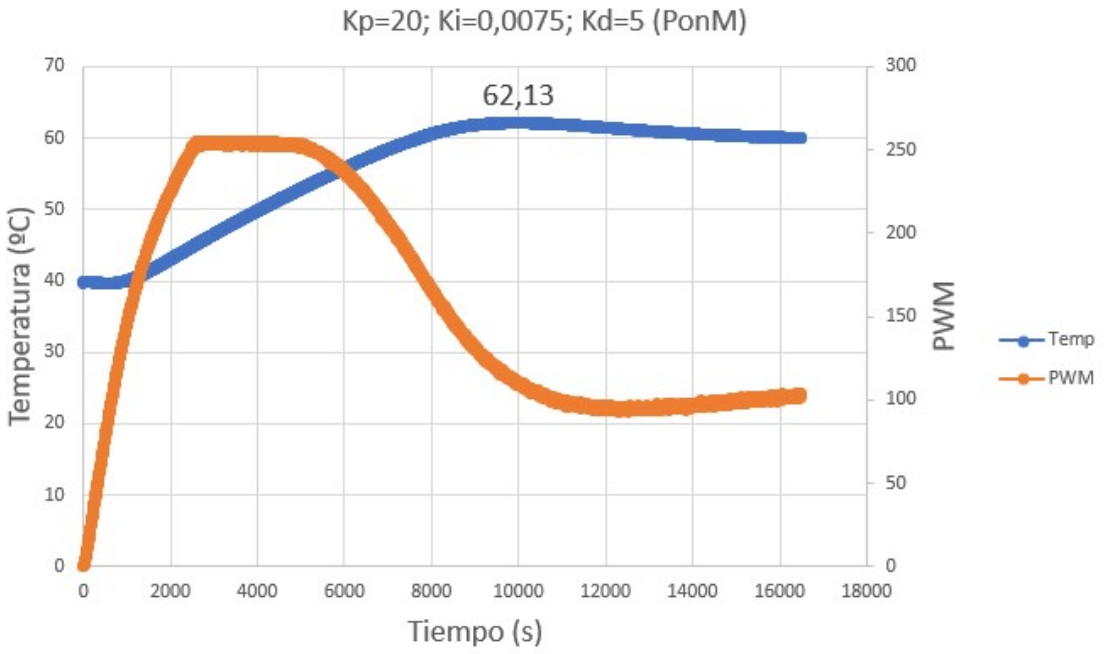

Figura 79: Respuesta del PID (Kp=20; Ki=0,0075; Kd=5, P on M)

Con los nuevos parámetros la sobreoscilación se ha reducido ligeramente, pero sigue estando a unos niveles no aceptables. Por tanto, se ha decidido mantener $K_i = 0{,}005$, ya que se considera un buen compromiso entre estabilidad y rapidez.

6.3. Pruebas finales

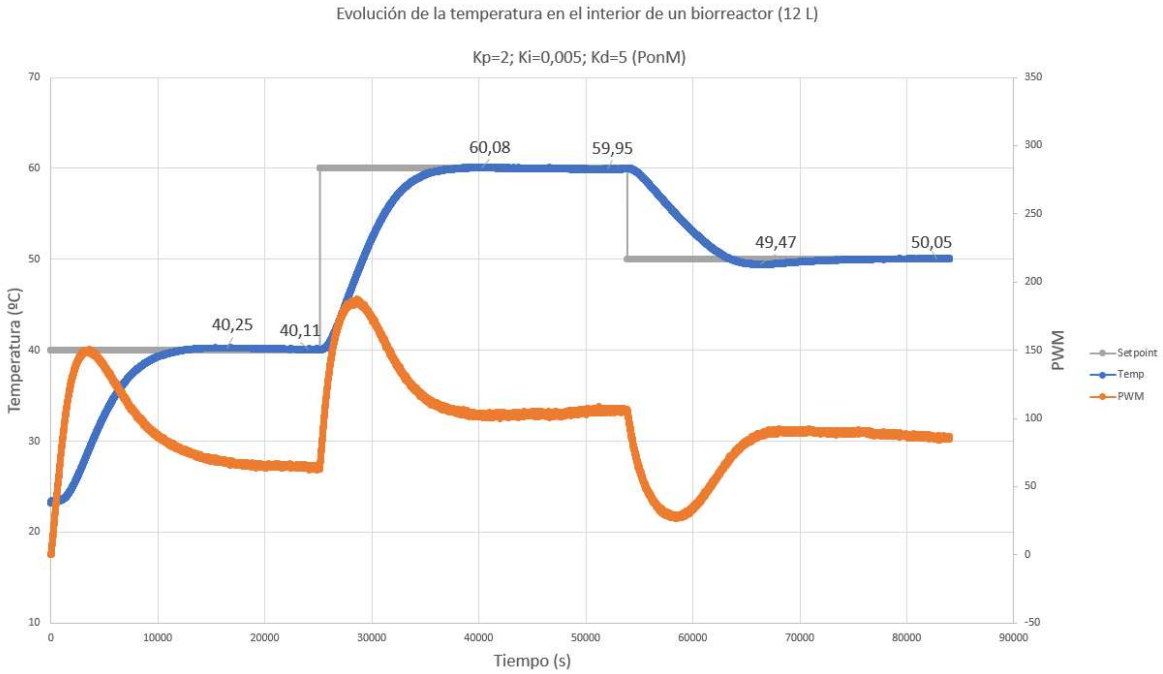

Figura 80: Evolución de la temperatura interna del biorreactor frente a diferentes valores del Setpoint

Tras realizar las pruebas mostradas anteriormente, finalmente se optó por emplear los parámetros $K_p = 20; K_i = 0{,}005; K_d = 5$. En la Figura 80, se observa la respuesta del controlador frente a diferentes escalones de temperatura en el interior de un biorreactor de 12 litros relleno de agua para realizar las pruebas. En todos ellos se puede ver que la sobreoscilación es despreciable y la respuesta permanece estable durante toda su duración al alcanzar el estado estacionario. Para obtener información más detallada, se analizan los siguientes parámetros [7]:

- $\boldsymbol{t_d}$: tiempo de retardo, aquel necesario para llegar al 50% del estado estacionario.
- $\boldsymbol{t_r}$: tiempo de crecimiento, tiempo que el sistema tarda en llegar del 10% al 90% del estado estacionario.
- $\boldsymbol{t_s}$ $(\mathbf{5\%})$: tiempo de establecimiento, tiempo que tarda la salida en alcanzar una banda ubicada en un valor igual al 5% del valor del estado estacionario.
- $\boldsymbol{M_p}$: máximo sobreimpulso, diferencia entre el máximo valor alcanzado y el valor del estado estacionario.

	$\boldsymbol{23-40\ ºC}$	$\boldsymbol{40-60\ ºC}$	$\boldsymbol{60-50\ ºC}$
$\boldsymbol{t_d}$ $(\boldsymbol{s})$	4440	3808	4488
$\boldsymbol{t_r}$ $(\boldsymbol{s})$	6652	6724	6648
$\boldsymbol{t_{s\,(5\%)}}$ $(\boldsymbol{s})$	8108	7076	6524
$\boldsymbol{M_p}$ $(\%)$	0,625	0,133	$-1{,}06$

Tabla 8: Especificaciones del sistema en el dominio temporal

Como se observa, el tiempo de establecimiento es relativamente largo, alrededor de 2 horas de media. Esto es principalmente causado por el propio sistema, ya que se están calentando casi 12 litros de agua con un calefactor de 200 W de potencia. Sin embargo, también se debe al

controlador, ya que se ha priorizado que no se genere sobreoscilación frente a que el tiempo de establecimiento sea corto. Esto es debido a que dependiendo del proceso en el que se esté utilizando el biorreactor, una sobreoscilación de tan solo un par de grados podría suponer el fracaso del experimento.

6.4. Pruebas de los usuarios

El prototipo ha sido incorporado por la empresa a su proceso de extracción de xilano como parte del proyecto *EnXylaScope* [4], expuesto en el apartado 2.3. En la Figura 81 se muestran las etapas del proceso de extracción de xilano que requieren el uso del biorreactor.

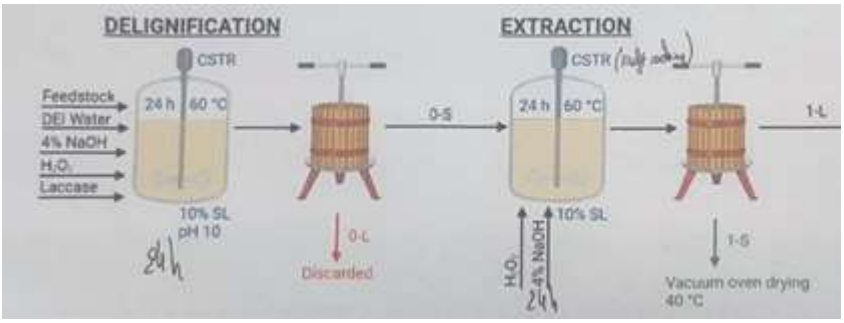

Figura 81: Pasos del proceso de extracción de xilano que requieren el biorreactor

Estas etapas son las siguientes:

1. Se mezcla la materia prima (previamente molida) con agua desionizada, hidróxido de sodio al 4%, peróxido de hidrógeno y una de las enzimas desarrolladas en el proyecto. Tras medir el pH de la mezcla, se cierra el biorreactor, se enciende el motor encargado de agitar la mezcla y se configura el controlador de temperatura a 60ºC.
2. Después de 24 horas, se vuelve a medir el pH y, a continuación, se extrae el exceso de líquido y se vuelve a añadir hidróxido de sodio y peróxido de hidrógeno. Tras medir el pH una vez más, el biorreactor se vuelve a cerrar y se configura el controlador de temperatura a 60ºC.
3. Al día siguiente, se extrae la mezcla del biorreactor y se prensa, extrayendo el líquido y dejando el xilano sólido. Este sólido es posteriormente tratado en un horno para secarse completamente antes de continuar el resto del proceso.

Figura 82: Biorreactor para la extracción de xilano en los pasos 1, 2 y 3

A petición de los usuarios, se añadió al prototipo la posibilidad de obtener también información sobre el output del PID, permitiendo así al usuario saber que está sucediendo en el controlador incluso cuando el prototipo no se encuentra conectado a un ordenador. Esta funcionalidad fue implementada en uno de los botones del shield LCD: al presionarlo, se sustituye la información sobre el tiempo restante con el valor del output.

Los resultados de este proceso fueron muy satisfactorios, por lo que la empresa introdujo el uso del prototipo en este proceso, el cual es realizado una vez por semana.

7. Conclusiones y Líneas Futuras

7.1. Conclusiones

En este trabajo se han desarrollado las habilidades de diseño y programación del estudiante, así como su capacidad de resolución de problemas, habiéndose desarrollado un prototipo operativo para un sistema de control de temperatura en un biorreactor. A pesar de las dificultades propias de haberlo realizado en una empresa de pequeño tamaño, como la falta de recursos, se han obtenido unos resultados muy satisfactorios, consiguiendose todos los objetivos marcados por la empresa para su correcto funcionamiento e introduciendose el uso del prototipo desarrollado en los procesos de trabajo de la empresa. Por esta razón, se solicitó al estudiante la construcción de dos prototipos adicionales previo a la finalización de su periodo de prácticas, lo que les permitirá aumentar la productividad del proceso de extracción de xilano al poder emplear múltiples biorreactores simultaneamente.

Adicionalmente, se ha podido observar la efectividad de utilizar un lazo de control PID PonM para evitar la sobreoscilación de la respuesta en sistemas que forman un “Integrating Process”, lo cual habría resultado imposible empleando un PID tradicional: en todas las pruebas realizadas con los parámetros finales se ha logrado obtener una sobreoscilación inferior al 1%, una variación tan pequeña que no podrá afectar negativamente los resultados de los diferentes procesos que los usuarios decidan realizar con el prototipo.

Figura 83: Los tres prototipos construidos por el estudiante

7.2. Líneas futuras

Debido a que el proyecto ha sido realizado durante la duración de unas prácticas curriculares, el periodo durante el cual se ha podido trabajar en el prototipo ha estado limitado por la duración de estas y las necesidades puntuales de la empresa. Esto ha impedido al estudiante aplicar una serie de mejoras en el prototipo, ya que requerían recursos no disponibles en la empresa o una cantidad considerable de tiempo para realizar pruebas. En este apartado se va a hablar de algunas de estas posibles mejoras que podrían ser implementadas en el futuro.

7.2.1. Segundo set de parámetros del PID

Debido a que se ha priorizado eliminar la sobreoscilación de la respuesta, el controlador actúa de manera muy cautelosa al comienzo del régimen transitorio, causando que el calefactor funcione por debajo de su potencia máxima y alargando su duración. Una mejora simple que podría reducir este problema es la inclusión de un segundo set de parámetros en el controlador PID. Estos parámetros se encontrarían activos cuando la temperatura interna se encuentra lo suficientemente alejada del Setpoint, haciendo que el controlador actúe de manera más agresiva antes de volver a los parámetros actuales, es decir, cuando el intervalo fuese lo suficientemente grande, como al principio del proceso, el Output del PID aumentará más rápido, llegando a valores del PWM superiores a los que alcanza con los parámetros más conservadores.

7.2.2. Mejora del diseño mecánico

Debido a los recursos limitados disponibles en la empresa, el diseño final no es muy refinado: la placa Arduino ha sido instalada en la cubierta de la caja eléctrica, causando que parte de los cables se encuentren en el exterior del prototipo. Empleando una impresora 3D sería posible obtener un diseño más compacto, eficiente y atractivo al usuario.

7.2.3. Incorporación de comunicación

Finalmente, se podría incorporar un módulo wifi como el ESP8266 01 WIFI [26] al prototipo. Esto permitiría a los usuarios comunicarse a distancia con el mismo desde un dispositivo externo como un teléfono móvil, facilitando la monitorización del proceso, ya que el módulo wifi enviaría al usuario información a tiempo real sobre la temperatura interna y tiempo restante en el proceso. También permitiría al usuario configurar el programa a distancia, sin necesidad de emplear los botones del shield LCD.

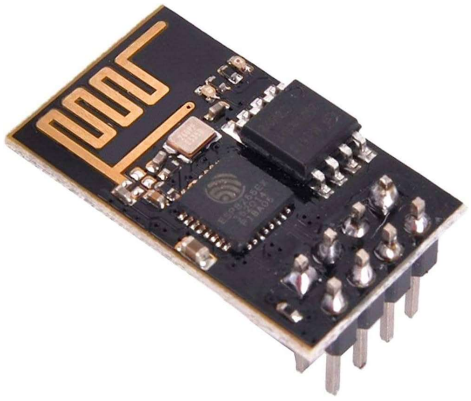

Figura 84: Módulo wifi ESP8266

8. Referencias

[1] Celignis Limited, “Biomass Analysis,” Celignis Analytical, 2023. [Online]. Available: https://www.celignis.com/services.php. [Accessed 19 July 2023].

[2] Tecnal, «Biorreactor: ¡Qué es y cuáles son sus productos!,» [En línea]. Available: https://tecnal.com.br/es/blog/347_biorreactor_que_es_y_cuales_son_sus_productos.

[3] J. Buford, J. McMillan, H. Doshi and M. Bryans, *SOP: Applikon ez-Control Bioreactor Controller Operation,* Blue Bell: Montgomery County Community College, 2017.

[4] Celignis Limited, “EnXylaScope - A research project at Celignis Biomass Lab.,” Celignis Analytical, 2021. [Online]. Available: https://www.celignis.com/project.php?value=8. [Accessed 2023].

[5] Celignis Limited, “Services for the Anaerobic Digestion Sector and RNG,” Celignis Analytical, [Online]. Available: https://www.celignis.com/ad.php. [Accessed 2023].

[6] G. E. Lima P., R. D. Mendez M. y A. R. Rojas B., «El Triac,» Monografias.com S.A., [En línea]. Available: https://www.monografias.com/trabajos14/triac/triac. [Último acceso: June 2023].

[7] D. Chuk, Los sistemas de primer orden y los Controladores PID, UNSJ, 2012.

[8] T. R. Kuphaldt, “Self-regulating, Integrating, and Runaway Process Characteristics,” in *Lessons In Industrial Instrumentation*, 2008, pp. 2413-2427.

[9] B. Beauregard, “Introducing Proportional on Measurement,” 20 June 2017. [Online]. Available: http://brettbeauregard.com/blog/2017/06/introducing-proportional-on-measurement/. [Accessed July 2023].

[10] K. Ogata, “Ziegler-Nichols Rules for Tuning PID Controllers,” in *Modern Control Engineering*, 5th ed., Pearson, 2009, pp. 568-577.

[11] S. Hornsey, “A Review of Relay Auto-tuning Methods for the Tuning of PID-type Controllers,” *Reinvention: an International Journal of Undergraduate Research,* vol. V, no. 2, 2012.

[12] V. VanDoren, “Relay Method Automates PID Loop Tuning,” September 2009. [Online]. Available: https://d1.amobbs.com/bbs_upload782111/files_36/ourdev_614499E39LAH.pdf. [Accessed July 2023].

[13] Arduino, “Arduino Uno Rev 3,” Arduino, [Online]. Available: https://store.arduino.cc/products/arduino-uno-rev3.

[14] M. A. Perez García, J. C. Alvarez Antón, J. C. Campo Rodríguez, F. J. Ferrero Martín y G. J. Grillo Ortega, «Sensores de temperatura de resistencia metálica,» de *Instrumentación Electrónica*, Thomson, 2004, pp. 207-225.

[15] Texas Instruments, *INA12x Precision, Low-Power Instrumentation Amplifiers,* 1995.

[16] E. Kennedy, “Resistance Temperature Detector (RTD),” in *Operational Amplifier Circuits: Theory and Applications*, Oxford University Press, 1995, pp. 483-490.

[17] Adafruit, “Overview | Adafruit MAX31865 RTD PT100,” Adafruit, [Online]. Available: https://learn.adafruit.com/adafruit-max31865-rtd-pt100-amplifier/.

[18] Maxim Integrated, *MAX31865 RTD-to-Digital Converter,* 2015.

[19] Maxim Integrated, “Achieve High-Accuracy Temperature Measurement in Your Precision Designs,” [Online]. Available: https://www.analog.com/media/en/reference-design-

documentation/design-notes/ds67-achieve-high-accuracy-temperature-measurement-in-your-precision-designs.pdf. [Accessed 18 July 2023].

[20] K. Grayson and T. Fukushima, “RTD Interfacing and Linearization Using an ADuC8xx MicroConverter,” Analog Devices, Inc, 2004. [Online]. Available: https://www.analog.com/media/en/technical-documentation/application-notes/AN709_0.pdf. [Accessed July 2023].

[21] Krida Electronics, “PWM 8A AC Light Dimmer Module 50Hz 60Hz TASMOTA,” Krida Electronics, [Online]. Available: https://www.tindie.com/products/bugrovs2012/pwm-8a-ac-light-dimmer-module-50hz-60hz-tasmota/.

[22] STMicroelectronics, “RC snubber circuit design for TRIACs,” 2007. [Online]. Available: https://www.st.com/resource/en/application_note/an437-rc-snubber-circuit-design-for-triacs-stmicroelectronics.pdf.

[23] tiendatec, «LCD KEYPAD SHIELD PARA ARDUINO,» [En línea]. Available: https://www.tiendatec.es/maker-zone/shields/842-lcd-keyboard-shield-para-arduino-8472496014038.html.

[24] Applikon, “Bio Bundles 1 -15 Liter (US Version) for Microbial Applications,” April 2003. [Online]. Available: https://archive-resources.coleparmer.com/Manual_pdfs/BioBundle%201-15L%20Microbial%20Hardware&InstallationManual.pdf. [Accessed 20 July 2023].

[25] Schneider Electric, «Hoja de Características del Producto: ENN05007,» 2018. [En línea]. Available: https://docs.rs-online.com/5519/0900766b8169c8a7.pdf.

[26] PromeTec, «Arduino y WIFI ESP8266,» [En línea]. Available: https://www.prometec.net/arduino-wifi/. [Último acceso: 20 julio 2023].

Anexo A: Presupuesto

En este Anexo se muestra el presupuesto del proyecto en caso de necesitar adquirir todos los componentes para la construcción de 1 y 10 unidades. También se muestra su coste real, ya que la empresa contaba con los componentes más costosos de antemano en estado de desuso.

Componente	Precio u.(€)	Precio 10 u.(€)	Coste real u.(€)
Adafruit MAX31865	15,00	150,00	15,00
Krida Electronics PWM AC dimmer	17,00	170,00	17,00
Arduino Uno Rev3	24,00	240,00	N/A
Mureva BOX ENN05007	9,07	90,70	N/A
PT100	32,00	288,90	30,54
Applikon ZC8130HB07	102,40	1024,00	N/A
Shield LCD	5,60	56,00	5,60
Total	205,07	2019,60	68,14

Tabla 9: Presupuesto del proyecto

Anexo B: Código del Arduino

```
#include <Arduino.h>
#include <LiquidCrystal.h>
#include <Adafruit_MAX31865.h>
#include <PID_v1.h>

#define DEBUG   //If you comment this line, the DPRINT & DPRINTLN lines are defined as blank.
#ifdef DEBUG    //Macros are usually in all capital letters.
   #define DPRINT(...)    Serial.print(__VA_ARGS__)     //DPRINT is a macro, debug print
   #define DPRINTLN(...)  Serial.println(__VA_ARGS__)   //DPRINTLN is a macro, debug print with new line
#else
   #define DPRINT(...)     //now defines a blank line
   #define DPRINTLN(...)   //now defines a blank line
#endif

// LCD
#define btnRIGHT  0
#define btnUP     1
#define btnDOWN   2
#define btnLEFT   3
#define btnSELECT 4
#define btnNONE   5
// Adafruit MAX31865
#define RREF      430.0 // Value of the Reference Resistor
#define RNOMINAL  100.0 // Value of the sensor at 0°C
// Estados sistema
#define status_STOP 0   // Initial state of the program (1)
#define status_STEPS 1  // State to determine the number of steps (2)
#define status_CONFIG 2 // Configuration of the lenght and desired temperature (3)
#define status_WORK 3   // State the Arduino is in when the PID is working (4)

// Indicate whether the program is configuring the duration or the temperature
#define s_TIME 0
#define s_TEMPERATURE 1

// Determine wether the display is showing the remaining duration or the PID output (PWM)
#define show_OUTPUT 0
#define show_TIME 1

// Output pin for the PWM used by the PID to control the AC dimmer
#define PIN_OUTPUT 3
// Max number of steps
#define MAX_STEPS 5

// Blinking interval (ms)
#define  BLINK_INTERVAL 500
// Button debounce interval (ms)
#define DEBOUNCE_DELAY 500

// Variables globales

// LCD display
LiquidCrystal lcd(8, 9, 4, 5, 6, 7);

// MAX31865
Adafruit_MAX31865 thermo = Adafruit_MAX31865(2, 11, 12, 13);

// Temperature value as measured by the Pt100
float temp=0;
```

```
59.
60. // Tells the program in which state it's in (Intial, configuration or
    working)
61. int status_program=status_STOP;
62. int modo_pantalla=show_TIME;       // Wether the display shows the remaning
    time or the PID output
63.
64. // Variables used for programming the time
65. unsigned long last_sec;             // Indicates the last moment a second
    was counted, used for the countdown
66. unsigned long last_sec_temp;        // Counts the seconds between the
    calculations of the temperature sensor
67. unsigned long current_millis;       // Current time value
68. unsigned long last_millis_blink=0;  // Indicates the last time the blinking
    state changed
69. unsigned long last_millis_PID=0;    // Indicates the last time the PID
    values were sent to the serial
70. unsigned long last_millis_temp=0;
71.
72. int parpadeo=0;             // Controls the blinking
73.
74. // Variables used in the subroutine to choose the number of steps
75. int num_steps=0;            // Indicates the current step
76. int total_steps=1;          // Indicates the number of steps
77.
78. // Variables used for the configuration of the duration and desired
    temperature
79. int time_or_temp=s_TIME;    // Indicates wether the value of the
    length/temperature is being changed
80. int pos_time=0;             // Indicates the positiong on the time vector
81. int pos_temp=0;             // Position of the temperature vector
82. float set_temp[MAX_STEPS];  // Temperature setpoint, calculated from the
    temperature vector
83. int v_time[7][MAX_STEPS];   // Time vector [hhh:mm:ss] shown on the display
84. int v_temp[3][MAX_STEPS];   // Temperature vector [xx.x °C] shown on the
    display
85.
86.
87. // PID
88. double Setpoint, Input, Output;
89. double Kp=20, Ki=0.005, Kd=5;
90. PID myPID(&Input, &Output, &Setpoint, Kp, Ki, Kd, P_ON_M, DIRECT);
91.
92.
93. /*
94.  * Program Setup. Initializes the different libraries.
95.  */
96. void setup()
97. {
98.   // Initialize serial port
99.   Serial.begin(115200);
100.     // Inicialize LCD library
101.     lcd.begin(16, 2);
102.     // Initialize Adafruit library (Pt100)
103.     thermo.begin(MAX31865_3WIRE);
104.     thermo.enable50Hz(true);    // Se activa el filtro de rechazo de 50Hz
    (por defecto está a 60Hz)
105.     // Inicializar el PID
106.     myPID.SetMode(AUTOMATIC);
107.     myPID.SetOutputLimits(30, 255); // Minumum output at 30: The heater
    doesn't really start to noticably warm up until around ~50 PWM, so this
    reduces the time it takes to heat up
108.     myPID.SetSampleTime(1000);  // PID sampling time
109.   }
110.
111.   /*
112.    *  Main loop of the program. Calls different subroutines depending on
113.    *  its current state
```

```
114.    */
115.   void loop()
116.   {
117.     if (status_program==status_WORK)            // Subroutines called when
   the temperature control is working
118.     {
119.       cuenta_atras();
120.       lectura_sensor();
121.       PID();
122.       pantalla();
123.       opcion_Pantalla();
124.     }
125.     else if (status_program==status_STOP)       // Subroutine activated
   when the Arduino is turned ON
126.             inicio_programa();
127.     else if (status_program==status_STEPS)      // Subrutina que configura
   el numero de periodos a diferentes temperaturas
128.             setupSteps();
129.     else if (status_program==status_CONFIG)     // Subrutinas que
   configuran el programa
130.     {
131.       pantallaConfig();
132.       setupProgram();
133.     }
134.   }
135.
136.   /*
137.    *      Subroutine active when the Arduino is turned ON or the previous
   program ends.
138.    *  Restarts most variables, makes the UI more attractive.
139.    */
140.   void inicio_programa()
141.   {
142.
143.     lcd.setCursor(0,0);
144.     lcd.print("Press SELECT to ");
145.     lcd.setCursor(0,1);
146.     lcd.print("start program   ");
147.     analogWrite(PIN_OUTPUT,0);  // Heater intially OFF
148.     if (read_LCD_buttons()==btnSELECT){
149.       status_program=status_STEPS;
150.       time_or_temp=s_TIME;      // Reiniciar variable
151.       lcd.clear();    // Se limpia la pantalla
152.       // Se reinician las variables
153.       memset(set_temp, 0, sizeof(set_temp));
154.             pos_time=0;
155.       pos_temp=0;
156.       num_steps=0;
157.       total_steps=1;
158.       memset(v_time, 0, sizeof(v_time));  // Se reinician los array
159.       memset(v_temp, 0, sizeof(v_temp));
160.       delay(DEBOUNCE_DELAY);     // Se espera medio segundo para evitar
   que se detecten como multiples pulsaciones  del botón
161.     }
162.   }
163.
164.   /*
165.    *  Subroutine where the number of steps is selected. Each of them
166.    *  will have a diferent duration and temperature setpoint.
167.    */
168.   void setupSteps()
169.   {
170.     current_millis=millis();
171.     lcd.setCursor(0,0);
172.     lcd.print("Select number of");
173.     lcd.setCursor(0,1);
174.     lcd.print("steps: ");
175.     if (current_millis-last_millis_blink>=BLINK_INTERVAL)
```

```
176.     {
177.       last_millis_blink+=BLINK_INTERVAL;
178.       if (parpadeo==0)
179.       {
180.         parpadeo=1;
181.         lcd.print(" ");
182.       }
183.       else
184.       {
185.         parpadeo=0;
186.         lcd.print(total_steps);
187.       }
188.     }
189.
190.     int lcd_key = read_LCD_buttons();  // read the buttons
191.     switch (lcd_key)               // Se realizan acciones diferentes en
    funcion del boton pulsado
192.     {
193.       case btnRIGHT:
194.       {
195.         break;
196.       }
197.       case btnLEFT:
198.       {
199.         break;
200.       }
201.       case btnUP:
202.       {
203.         total_steps++;
204.         if (total_steps>MAX_STEPS) total_steps=1;
205.         delay(DEBOUNCE_DELAY);
206.         break;
207.       }
208.       case btnDOWN:
209.       {
210.         total_steps--;
211.         if (total_steps<=0) total_steps=MAX_STEPS;
212.         delay(DEBOUNCE_DELAY);
213.         break;
214.       }
215.       case btnSELECT:
216.       {
217.         status_program=status_CONFIG;
218.         delay(DEBOUNCE_DELAY);
219.         lcd.clear();
220.         break;
221.       }
222.       case btnNONE:
223.       {
224.         break;
225.       }
226.     }
227.   }
228.
229.   /*
230.    *  Subroutine in charge of controlling the display to look like this:
231.    *      Time: hhh:mm:ss
232.    *      T= XX.X °C   (70.0)
233.    *  The temperature setpoint is between the parenthesis, while the real
    time value of the temperature is on the left.
234.    *  Pressing the UP button while the program is in this state will
    display the PID output instead of the duration.
235.    */
236.   void pantalla()
237.   {
238.     if (modo_pantalla==show_TIME)
239.     {
240.       lcd.setCursor(0,0);
```

```
241.       lcd.print("Time");
242.       lcd.print(num_steps+1);
243.       lcd.print(": ");
244.       //lcd.setCursor(6,0);
245.       for (int i=0;i<=2;i++)
246.       {
247.         if (i!=pos_time || time_or_temp==s_TEMPERATURE || status_program==
    status_WORK)
248.         {
249.           lcd.setCursor(7+i,0);
250.           lcd.print(v_time[i][num_steps]);
251.         }
252.       }
253.       lcd.setCursor(10,0);
254.       lcd.print(":");
255.       for (int i=3;i<=4;i++)
256.       {
257.         if (i!=pos_time || time_or_temp==s_TEMPERATURE || status_program==
    status_WORK)
258.         {
259.           lcd.setCursor(8+i,0);
260.           lcd.print(v_time[i][num_steps]);
261.         }
262.       }
263.       lcd.setCursor(13,0);
264.       lcd.print(":");
265.       for (int i=5;i<=6;i++)
266.       {
267.         if (i!=pos_time || time_or_temp==1 || status_program==status_WORK)
268.         {
269.           lcd.setCursor(9+i,0);
270.           lcd.print(v_time[i][num_steps]);
271.         }
272.       }
273.
274.       // Cuando el programa esta en funcionamiento, la temperatura que
    aparecera en pantalla es la que esta detectando el sensor Pt100
275.       //set_temp=v_temp[0]*10+v_temp[1]+v_temp[2]*0.1;      // Ya se
    calcula en la subrutina de configuracion
276.       //Mostrar Temperatura
277.       lcd.setCursor(0,1);
278.       lcd.print("T");
279.       //lcd.print(num_steps+1);
280.       lcd.print(": ");
281.       lcd.setCursor(3,1);
282.       if (temp<10) lcd.print("0");     // Simplemente mantiene consistencia
    en el display, se quieren dos digitos
283.       double potencia=pow(10,1);       // Se redondea el valor de la
    temperatura a 1 decima
284.       lcd.print(roundf(temp*potencia)/potencia);
285.       lcd.setCursor(7,1);
286.       lcd.print("\337C ");
287.       lcd.setCursor(10,1);
288.       lcd.print("(");
289.       lcd.print(set_temp[num_steps]); // Se muestra a la vez la
    temperatura objetivo, para facilitar la comprobacion de su correcto
    funcionamiento
290.       lcd.setCursor(15,1);
291.       lcd.print(")");
292.     }
293.
294.     if (modo_pantalla==show_OUTPUT)
295.     {
296.       lcd.setCursor(0, 0);
297.       lcd.print("Output: ");
298.       lcd.print(Output);
299.       lcd.print("       ");
300.     }
```

```
301.
302.
303.   }
304.
305.   /*
306.    * Subroutine that controls the display while the program is being
   configured.
307.    * Controls the blinking of the selected digit
308.    */
309.   void pantallaConfig()
310.   {
311.     current_millis=millis();
312.
313.     // Mostrar tiempo
314.     lcd.setCursor(0,0);
315.     lcd.print("Time");
316.     lcd.print(num_steps+1);
317.     lcd.print(": ");
318.     //lcd.setCursor(6,0);
319.     for (int i=0;i<=2;i++)
320.     {
321.       if (i!=pos_time || time_or_temp==s_TEMPERATURE || status_program==st
   atus_WORK)
322.       {
323.         lcd.setCursor(7+i,0);
324.         lcd.print(v_time[i][num_steps]);
325.       }
326.     }
327.     lcd.setCursor(10,0);
328.     lcd.print(":");
329.     for (int i=3;i<=4;i++)
330.     {
331.       if (i!=pos_time || time_or_temp==s_TEMPERATURE || status_program==st
   atus_WORK)
332.       {
333.         lcd.setCursor(8+i,0);
334.         lcd.print(v_time[i][num_steps]);
335.       }
336.     }
337.     lcd.setCursor(13,0);
338.     lcd.print(":");
339.     for (int i=5;i<=6;i++)
340.     {
341.       if (i!=pos_time || time_or_temp==1 || status_program==status_WORK)
342.       {
343.         lcd.setCursor(9+i,0);
344.         lcd.print(v_time[i][num_steps]);
345.       }
346.     }
347.
348.     // Parpadeo del tiempo
349.     if (time_or_temp==s_TIME)
350.     {
351.       if (pos_time<=2) lcd.setCursor(pos_time+7,0);
352.       else if (pos_time<=4) lcd.setCursor(pos_time+8,0);
353.       else if (pos_time<=6) lcd.setCursor(pos_time+9,0);
354.     }
355.     if (current_millis-last_millis_blink>=BLINK_INTERVAL)
356.     {
357.       last_millis_blink+=BLINK_INTERVAL;
358.       if (parpadeo==0)
359.       {
360.         if (time_or_temp==s_TIME)
361.         {
362.           parpadeo=1;
363.           lcd.print(" ");
364.         }
365.       }
```

```
366.       else
367.       {
368.         if (time_or_temp==s_TIME)
369.         {
370.           parpadeo=0;
371.           lcd.print(v_time[pos_time][num_steps]);
372.         }
373.       }
374.     }
375.
376.     //Mostrar Temperatura
377.     lcd.setCursor(0,1);
378.     lcd.print("T");
379.     //lcd.print(num_steps+1);
380.     lcd.print(": ");
381.     //lcd.setCursor(2,1);
382.     for (int i=0; i<=1;i++)
383.     {
384.       if (i!=pos_temp || time_or_temp==s_TIME)
385.       {
386.         lcd.setCursor(3+i,1);
387.         lcd.print(v_temp[i][num_steps]);
388.       }
389.     }
390.     lcd.setCursor(5,1);
391.     lcd.print(".");
392.     if (2!=pos_temp || time_or_temp==s_TIME)
393.     {
394.       lcd.setCursor(6,1);
395.       lcd.print(v_temp[2][num_steps]);
396.       lcd.print("\337C");
397.     }
398.
399.     // Parpadeo de la temperatura
400.     if (time_or_temp==s_TEMPERATURE)
401.     {
402.       if (pos_temp<=1) lcd.setCursor(pos_temp+3,1);
403.       else if (pos_temp==2) lcd.setCursor(pos_temp+4,1);
404.     }
405.     if (current_millis-last_millis_temp>=BLINK_INTERVAL)
406.     {
407.       last_millis_temp+=BLINK_INTERVAL;
408.       if (parpadeo==0)
409.       {
410.         if (time_or_temp==s_TEMPERATURE && status_program==status_CONFIG)
411.         {
412.           parpadeo=1;
413.           lcd.print(" ");
414.         }
415.       }
416.       else
417.       {
418.         if (time_or_temp==s_TEMPERATURE && status_program==status_CONFIG)
419.         {
420.           parpadeo=0;
421.           lcd.print(v_temp[pos_temp][num_steps]);
422.         }
423.       }
424.     }
425.   }
426.
427.   /*
428.    *      Esta subrutina se encarga de la configuración del programa,
   seleccionando su duración y temperatura objetivo.
429.    *      Para ello, se cambian los valores de las cifras del tiempo con
   los botones. SELECT cambia a la temperatura.
430.    *      Tras cambiar a las cifras deseadas, SEELCT termina la
   configuración y comienza el funcionamiento del sistema
```

```
431.    *
432.    *  Used for setting up the program, selecting the desired duration and
   temperature with the buttons included
433.    *  on the LCD shield.
434.    */
435.   void setupProgram()
436.   {
437.     int lcd_key = read_LCD_buttons();  // read the buttons
438.     // Configurar tiempo del programa
439.     if (time_or_temp==s_TIME)
440.     {
441.       switch (lcd_key)               // Se realizan acciones diferentes en
   funcion del boton pulsado
442.       {
443.         case btnRIGHT:
444.         {
445.           pos_time++;      // Se cambia a la posicion de la dcha
446.           if (pos_time>=7) pos_time=0; // Se vuelve a la posicion inicial
447.           delay(DEBOUNCE_DELAY);
448.           break;
449.         }
450.         case btnLEFT:
451.         {
452.           pos_time--;      // Se cambia a la posicion de la izda
453.           if (pos_time<0) pos_time=6; // Se vuelve a la posicion final
454.           delay(DEBOUNCE_DELAY);
455.           break;
456.         }
457.         case btnUP:
458.         {
459.           v_time[pos_time][num_steps]++; // Se aumenta el valor de esa
   posicion del vector
460.           if ((pos_time==3 || pos_time==5) && v_time[pos_time][num_steps]>
   5) // Si las decenas de minuto/segundo superan 5, volver a 0
461.           {
462.             v_time[pos_time][num_steps]=0;
463.           }
464.           else if (v_time[pos_time][num_steps]>9)    // Si se trata de
   cualquier otra unidad, 9 es el valor max
465.           {
466.             v_time[pos_time][num_steps]=0;
467.           }
468.           delay(DEBOUNCE_DELAY);
469.           break;
470.         }
471.         case btnDOWN:
472.         {
473.           v_time[pos_time][num_steps]--; // Se disminuye el valor de esa
   posicion del vector
474.           if ((pos_time==3 || pos_time==5) && v_time[pos_time][num_steps]<
   0) // Si las decenas de minuto/segundo bajan de 0, volver a 5
475.           {
476.             v_time[pos_time][num_steps]=5;
477.           }
478.           else if (v_time[pos_time][num_steps]<0)    // Si se trata de
   cualquier otra unidad, 9 es el valor max
479.           {
480.             v_time[pos_time][num_steps]=9;
481.           }
482.           delay(DEBOUNCE_DELAY);
483.           break;
484.         }
485.         case btnSELECT:
486.         {
487.           time_or_temp=s_TEMPERATURE;
488.           delay(DEBOUNCE_DELAY);
489.           break;
490.         }
```

```
491.          case btnNONE:
492.          {
493.            break;
494.          }
495.        }
496.      }
497.            // Configurar temp objetivo
498.      else if (time_or_temp==s_TEMPERATURE)
499.      {
500.        switch (lcd_key)               // Se realizan acciones diferentes en
    funcion del boton pulsado
501.        {
502.          case btnRIGHT:
503.          {
504.            pos_temp++;       // Se cambia a la posicion de la dcha
505.            if (pos_temp>=3) pos_temp=0; // Se vuelve a la posicion inicial
506.            delay(DEBOUNCE_DELAY);      // Se espera medio segundo para
    evitar que se detecten como multiples pulsaciones
507.            break;
508.          }
509.          case btnLEFT:
510.          {
511.            pos_temp--;       // Se cambia a la posicion de la izda
512.            if (pos_temp<0) pos_temp=2; // Se vuelve a la posicion final
513.            delay(DEBOUNCE_DELAY);      // Se espera medio segundo para
    evitar que se detecten como multiples pulsaciones
514.            break;
515.          }
516.          case btnUP:
517.          {
518.            v_temp[pos_temp][num_steps]++; // Se aumenta el valor de esa
    posicion del vector
519.            if (v_temp[pos_temp][num_steps]>9)    // Si llega al max, volver
    a 0
520.            {
521.              v_temp[pos_temp][num_steps]=0;
522.            }
523.            delay(DEBOUNCE_DELAY);      // Se espera medio segundo para
    evitar que se detecten como multiples pulsaciones
524.            break;
525.          }
526.          case btnDOWN:
527.          {
528.            v_temp[pos_temp][num_steps]--; // Se aumenta el valor de esa
    posicion del vector
529.            if (v_temp[pos_temp][num_steps]<0)    // Si llega al max, volver
    a 0
530.            {
531.              v_temp[pos_temp][num_steps]=9;
532.            }
533.            delay(DEBOUNCE_DELAY);      // Se espera medio segundo para
    evitar que se detecten como multiples pulsaciones
534.            break;
535.          }
536.          case btnSELECT:
537.          {
538.            set_temp[num_steps]=v_temp[0][num_steps]*10+v_temp[1][num_steps]
    +v_temp[2][num_steps]*0.1;  // Transforma el array en un numero
539.            num_steps++;
540.            time_or_temp=s_TIME;
541.                  pos_time=0;
542.            pos_temp=0;
543.            if (num_steps>=total_steps)
544.            {
545.              num_steps=0;
546.              status_program=status_WORK;       // Config finalizada
547.            }
```

```
548.           delay(DEBOUNCE_DELAY);     // Se espera medio segundo para
   evitar que se detecten como multiples pulsaciones
549.           break;
550.         }
551.         case btnNONE:
552.         {
553.           break;
554.         }
555.       }
556.     }
557.   }
558.
559.   /*
560.    *      Subrutina que se encarga de la cuenta atras del tiempo de
   funcionamiento. Para ello, se realiza la resta del
561.    *  tiempo actual con la de la ultima vez que se detecto el paso de 1
   segundo. Cuando se vuelve a detectar, se
562.    *  guarda el tiempo actual  y se resta 1 segundo, realizandose tambien
   los cambios necesarios en las cantidades
563.    *      de minutos y horas. Una vez la cuenta atras llega a 0, el
   programa se da por finalizado y se vuelve al estado
564.    *  de inicio status_STOP.
565.    *
566.    *  Subroutine in change of the countdown. Once it reaches 0, the
   program goes back to its intial state.
567.    */
568.   void cuenta_atras()
569.   {
570.     if (millis()-last_sec>=1000)
571.     {
572.       last_sec=millis();     // Se guarda el tiempo actual
573.       v_time[6][num_steps]--;  // Se resta una unidad de segundo
574.       if (v_time[6][num_steps]<0 && (v_time[5][num_steps]!=0 || v_time[4][
   num_steps]!=0 || v_time[3][num_steps]!=0 || v_time[2][num_steps]!=0 || v_ti
   me[1][num_steps]!=0 || v_time[0][num_steps]!=0))
575.       {
576.         v_time[6][num_steps]=9;
577.         v_time[5][num_steps]--;  //  Se resta una decena de segundo
578.         if (v_time[5][num_steps]<0 && (v_time[4][num_steps]!=0 || v_time[3
   ][num_steps]!=0 || v_time[2][num_steps]!=0 || v_time[1][num_steps]!=0 || v_
   time[0][num_steps]!=0))
579.         {
580.           v_time[5][num_steps]=5;       //  Se reinician decenas de
   segundo
581.           v_time[4][num_steps]--;      // Se resta un minuto
582.           if (v_time[4][num_steps]<0 && (v_time[3][num_steps]!=0 || v_time
   [2][num_steps]!=0 || v_time[1][num_steps]!=0 || v_time[0][num_steps]!=0))
583.           {
584.             v_time[4][num_steps]=9;    // Se reinicia unidad minuto
585.             v_time[3][num_steps]--;    // Se resta decena minuto
586.             if (v_time[3][num_steps]<0 && (v_time[2][num_steps]!=0 || v_ti
   me[1][num_steps]!=0 || v_time[0][num_steps]!=0))
587.             {
588.               v_time[3][num_steps]=5; // Se reinicia decena minutos
589.               v_time[2][num_steps]--; // Se resta una hora
590.               if (v_time[2][num_steps]<0 && (v_time[1][num_steps]!=0 || v_
   time[0][num_steps]!=0))
591.               {
592.                 v_time[2][num_steps]=9; // Reinciar unidades hora
593.                 v_time[1][num_steps]--; // Resta decena horas
594.                 if (v_time[1][num_steps]<0 && v_time[0][num_steps]!=0)
595.                 {
596.                   v_time[1][num_steps]=9; // Reinciar decena horas
597.                   v_time[0][num_steps]--; // Resta centena horas
598.                   if (v_time[0][num_steps]<0) v_time[0][num_steps]=0;
599.                 }
600.                 else if (v_time[1][num_steps]<0) v_time[1][num_steps]=0;
601.               }
```

```
602.              else if (v_time[2][num_steps]<0) v_time[2][num_steps]=0;
603.            }
604.            else if (v_time[3][num_steps]<0) v_time[3][num_steps]=0;
605.          }
606.          else if (v_time[4][num_steps]<0) v_time[4][num_steps]=0;
607.        }
608.        else if (v_time[4][num_steps]<0) v_time[4][num_steps]=0;
609.      }
610.      else if (v_time[6][num_steps]<0)
611.      {
612.        v_time[6][num_steps]=0;
613.        num_steps++;
614.        lcd.clear();
615.        if (num_steps>=total_steps) status_program=status_STOP;    // Se
    reactiva la configuracion: Programa finalizado
616.      }
617.    }
618.  }
619.
620.  /*
621.   *      Subrutina que emplea la librería del Adafruit MAX31865 para
622.   *      obtener el valor de la termorresistencia y convertirla en su
623.   *      valor equivalente en grados celsius cada segundo.
624.   *
625.   *  Uses the Adafruit MAX331865 library to obtain the value of the
626.   *  temperature and convert it to °C
627.   */
628.  void lectura_sensor()
629.  {
630.    if (millis()-last_sec_temp>=1000)
631.    {
632.     last_sec_temp = millis();    // Se reinicia el tiempo
633.     temp=thermo.temperature(RNOMINAL, RREF);     // Se guarda el valor de
    la temperatura
634.     //DPRINT("Temperature = ");
635.     //DPRINTLN(temp);
636.
637.     // Comprobación de errores en el sensor
638.     uint8_t fault = thermo.readFault();
639.     if (fault)
640.      {
641.       DPRINT("Fault 0x "); DPRINT(fault, HEX);DPRINT(" ");
642.       if (fault & MAX31865_FAULT_HIGHTHRESH)
643.       {
644.         DPRINT("RTD High Threshold ");
645.       }
646.       if (fault & MAX31865_FAULT_LOWTHRESH)
647.       {
648.         DPRINT("RTD Low Threshold ");
649.       }
650.       if (fault & MAX31865_FAULT_REFINLOW)
651.       {
652.         DPRINT("REFIN- > 0.85 x Bias ");
653.       }
654.       if (fault & MAX31865_FAULT_REFINHIGH)
655.       {
656.         DPRINT("REFIN- < 0.85 x Bias - FORCE- open ");
657.       }
658.       if (fault & MAX31865_FAULT_RTDINLOW)
659.       {
660.         DPRINT("RTDIN- < 0.85 x Bias - FORCE- open ");
661.       }
662.       if (fault & MAX31865_FAULT_OVUV)
663.       {
664.         DPRINT("Under/Over voltage ");
665.       }
666.            //DPRINTLN(); // Salto de linea al final -> No necesario para
    el excel
```

```
667.         thermo.clearFault();
668.       }
669.     }
670.   }
671.
672.   /*
673.    *      Función que contiene el lazo de control. El algoritmo PID genera
   una señal PWM
674.    *  en función del error que se envia al sistema de actuación
675.    *
676.    *  Contains the PID control. The controller generates a PWM signal used
   to tell
677.    *  the AC dimmer how much power the heater should be receiving.
678.    */
679.   void PID()
680.   {
681.     Input=temp;
682.     Setpoint=set_temp[num_steps];
683.     myPID.Compute();
684.     analogWrite(PIN_OUTPUT, Output);
685.     current_millis=millis();
686.     if (current_millis-last_millis_PID>=4000)    // Periodo en el que se
   muestran los valores por el serial
687.     {
688.       last_millis_PID=current_millis;
689.       DPRINT(":  ");
690.       DPRINT(Setpoint);
691.       DPRINT("  :  ");
692.       DPRINT(Input);
693.       DPRINT("  :  ");
694.       DPRINTLN(Output);
695.     }
696.   }
697.   /*
698.    *  Esta subrutina lee los botones durante el funcionamiento del
   programa, seleccionando una dirección (Arriba), la pantalla
699.    *  muestra información diferente: El output del PID
700.    *
701.    *  Reads the buttons while the program is running, allowing the user to
   see the PID Output by pressing UP
702.    */
703.   void opcion_Pantalla()
704.   {
705.     int lcd_key = read_LCD_buttons();  // read the buttons
706.
707.     switch (lcd_key)               // Se realizan acciones diferentes en
   funcion del boton pulsado
708.     {
709.       case btnRIGHT:
710.       {
711.         break;
712.       }
713.       case btnLEFT:
714.       {
715.         break;
716.       }
717.       case btnUP:
718.       {
719.         modo_pantalla=show_OUTPUT;
720.         break;
721.       }
722.       case btnDOWN:
723.       {
724.         break;
725.       }
726.       case btnSELECT:
727.       {
728.         break;
```

```
729.       }
730.       case btnNONE:
731.       {
732.         modo_pantalla=show_TIME;
733.         break;
734.       }
735.     }
736.   }
737.
738.   /*
739.    *  Función de lectura de los botones del shield LCD
740.    *
741.    *  Reads the buttons on the LCD shield
742.    */
743.   int read_LCD_buttons()
744.   {
745.    int adc_key_in = analogRead(0);      // Se lee el valor de entrada
   analogica.
746.    // Los botones estan centrados en los siguientes valores: 0, 144, 329,
   504, 741
747.    // Se añade aproximadamente 50 a cada valor y se comprueba
748.    if (adc_key_in > 1500) return btnNONE; // Dado que es la opcion mas
   probable, se comprueba la primera
749.    if (adc_key_in < 50)   return btnRIGHT;
750.    if (adc_key_in < 195)  return btnUP;
751.    if (adc_key_in < 380)  return btnDOWN;
752.    if (adc_key_in < 500)  return btnLEFT;
753.    if (adc_key_in < 700)  return btnSELECT;
754.    return btnNONE;  // Si todos los demas if fallan, se devuelve nada
755.   }
```

Anexo C: User's Guide

1. Setting up the program

The device has a series of different menus to help configure the process' desired duration and temperature.

1.1. Start menu

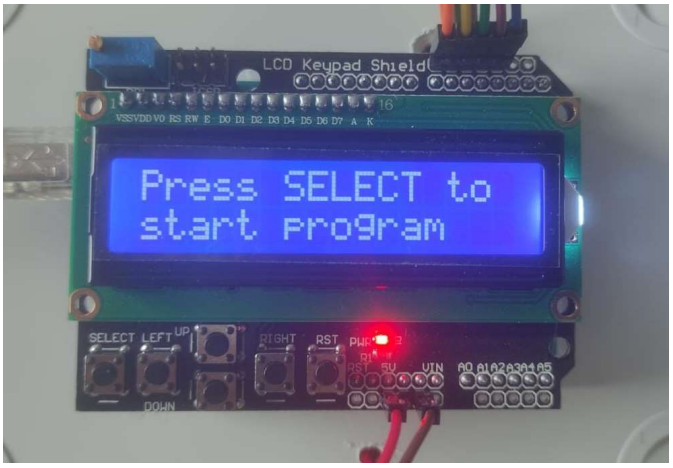

Figure 1: Start menu

1.2. Selection of the number of steps

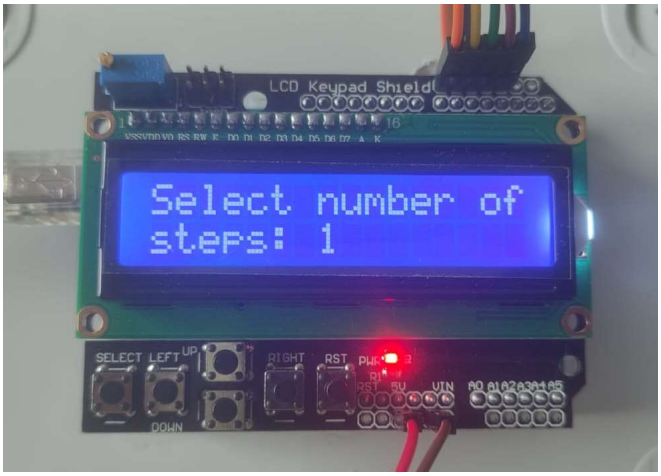

Figure 2: Steps selection menu

Allows the user to configure up to 5 different temperature steps, each with its own duration.

1. Press the UP/DOWN buttons to change the value.
2. Press SELECT to continue to the next menu.

1.3. Configuration menu

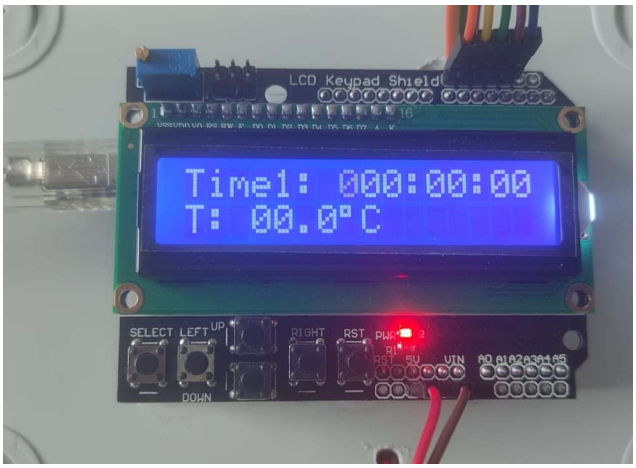

Figure 3: Configuration menu

1. Pressing LEFT/RIGHT changes to a different digit of time. Pressing UP/DOWN changes the value of the selected digit.
2. Press SELECT to configure the temperature setpoint.
3. Choose the temperature with the same method as the time and press SELECT to continue.

1.4. Display while the program is running

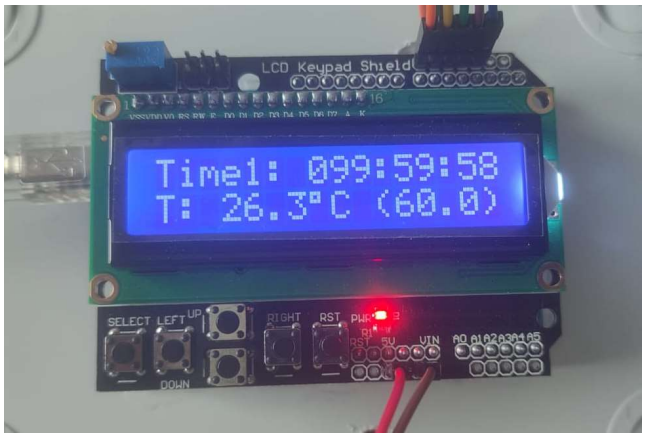

Figure 4: Process menu

Displays real-time information about the remaining duration, the current temperature on the inside of the bioreactor and the setpoint value.

Holding UP while in this menu will display the value of the PID output, that is, the power the heater is currently receiving, on a scale with a maximum value of 255. The heating may not be noticeable until the output reaches a value of around 70.

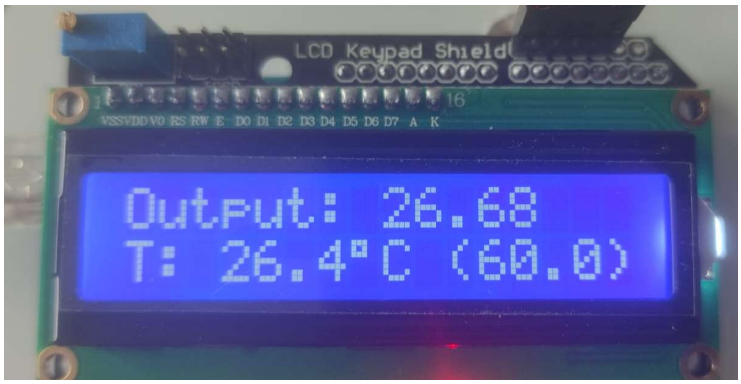

Figure 5: Output menu

2. Connections

2.1. SPI Logic Pins

Figure 6: SPI Logic Pins

The picture above shows the Arduino pins being used by the temperature sensor and the AC dimmer. The connections inside the box are as follows:

Adafruit Max31865

- **#13:** CLK
- **#12:** SDO
- **#11:** SDI
- **#2:** CS

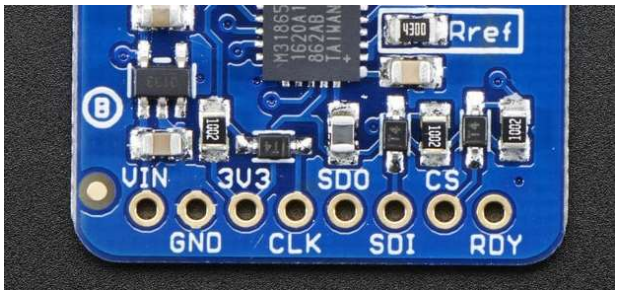

Figure 7: Adafruit MAX31865 Pins

The pins 3V3 and RDY are not used.

AC dimmer

- **#3:** PWM

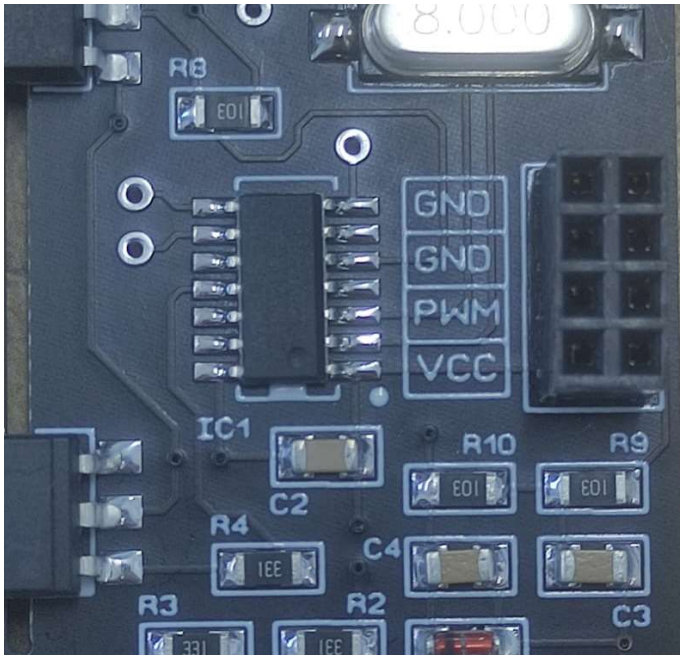

Figure 8: AC dimmer pin connections

2.2. Power Pins

Figure 9: Power pins

They are connected to the pins on the left of the PCB seen below. These pins are used to power the Adafruit MAX31865 and the AC dimmer.

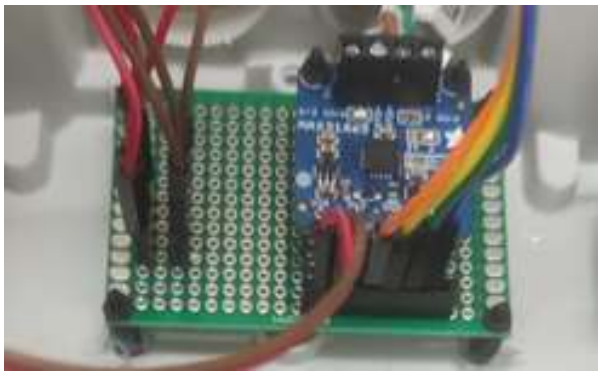

Figure 10: Connections inside the box

3. Hardware

Name	Link	Price (€)
Adafruit MAX31865	https://www.adafruit.com/product/3328	15
Krida Electronics PWM AC dimmer	https://www.ebay.es/itm/124296310513?hash=item1cf0a32ef1:g:MLIAAOSwnjlhyxvy	17

Table 1: Hardware

The picture below shows the proper way to connect the AC dimmer to the AC line input.

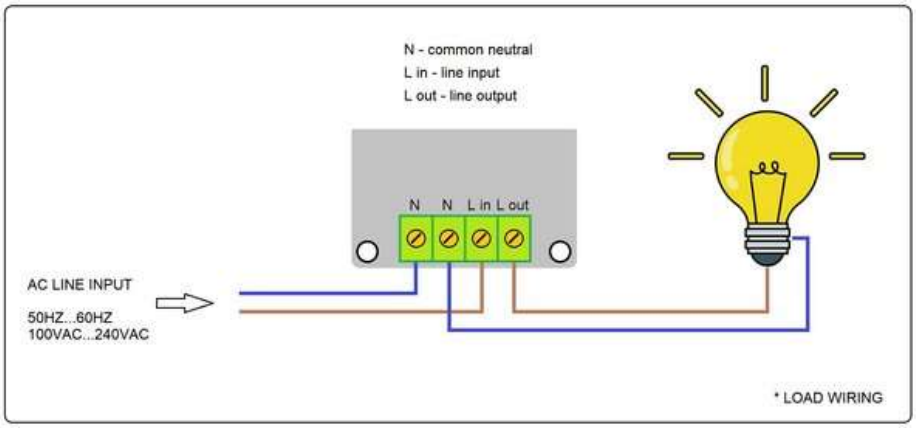

Figure 11: AC line connection

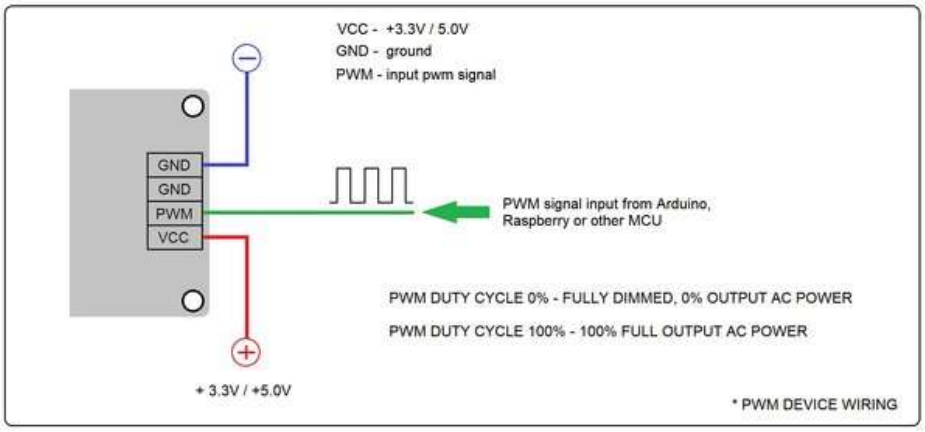

Figure 12: Arduino connection