industriales
etsii

Escuela Técnica Superior de Ingeniería Industrial

UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Escuela Técnica Superior de Ingeniería Industrial

Desarrollo de un entorno para la monitorización remota de prácticas de laboratorio para sensores de temperatura

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

Autor: **Pablo Saura Ródenas**
Director: Joaquín Roca González

Cartagena, 21 de marzo de 2020

Universidad Politécnica de Cartagena

TRABAJO DE FIN DE GRADO

DESARROLLO DE UN ENTORNO PARA LA MONITORIZACIÓN REMOTA DE PRÁCTICAS DE LABORATORIO PARA SENSORES DE TEMPERATURA

Tutor: Joaquín Roca González

Alumno: Pablo Saura Ródenas

A mi familia por el apoyo mostrado y por haberme permitido estudiar una carrera.

A mi tutor Joaquín por haber lanzado esta beca y haberme seleccionado.

A mi novia Conchi, por el apoyo en los peores momentos.

Y, por último, a Dios, por darme la vida.

ÍNDICE DE CONTENIDOS:

ÍNDICE DE FIGURAS:

ÍNDICE DE TABLAS:

ÍNDICE DE ECUACIONES:

1. INTRODUCCIÓN:

En los últimos años, la vida cotidiana y el trabajo del hombre se están estrechando cada vez más con el rápido crecimiento de las tecnologías de la comunicación e información (TICs), cambiando así también nuestro modo de vida, siendo actualmente la vida segura, cómoda y rentable el ideal de toda persona.

Debido al rápido desarrollo de las tecnologías electrónicas se ha favorecido su integración en la industria. La idea básica es emplear sensores y sistemas de control para monitorear y, por lo tanto, ajustar los diversos mecanismos. En la actualidad es más normal la obtención de datos especialmente en tiempo real, su tratamiento y envío, o lo que comúnmente conocemos como el Internet de las Cosas (IoT). En esta tecnología siempre se pretende ir un paso más allá en cuanto al lenguaje entre máquinas (M2M), de manera que se busca el bienestar de las personas realizando o simplificando tareas tan comunes como son poner el navegador en el móvil, abrir a una persona que llama al telefonillo, e incluso realizar una videollamada con alguien que se encuentra en la otra esquina del planeta [1].

No solo ha mejorado la calidad y se ha favorecido la aparición de sensores inteligentes, sino que también ha mejorado el software y cualquier interfaz de usuario, pudiendo ver en la actualidad multitud de plantas industriales y viviendas que gracias a los grandes avances en las telecomunicaciones (especialmente utilizando Wi-Fi y Bluetooth) que ya permiten el control mediante dispositivos móviles, e incluso estando fuera de la propia vivienda, de los diversos electrodomésticos.

Si a la transmisión de dichos datos le incluimos el pretratamiento y el postratamiento, podremos encontrarnos con un sistema inteligente que además analiza los datos en búsqueda de conclusiones, como podría ser un atasco, o la densidad de gente que acude a un comercio en función de la hora del día, estimando incluso el tipo de producto que hemos comprado [2].

El control de sistemas es una rama muy importante dentro del terreno de la ingeniería. Nos permite establecer unas consignas y obtener una salida que se identifica como función de la entrada y que nos ayuda a conocer cómo es nuestro sistema.

Hoy en día, la educación de control generalmente depende en gran medida de los paquetes de simulación disponibles y de laboratorios virtuales, los cuales tienen un puesto irremplazable en el proceso educativo. Pero desafortunadamente, los experimentos de circuitos cerrados se limitan con frecuencia a estos dominios virtuales y los estudiantes carecen de la retroalimentación física del impacto de los algoritmos de control y sus parámetros.

Ante la concesión de una beca de colaboración con un profesor experto en ingeniería biomédica, consideré la importancia de lanzarme a un proyecto que nos permitiera el control y la monitorización de temperatura remotamente, trabajando con tecnologías como Arduino y Raspberry, que ya despertaron mi curiosidad para realizar proyectos en casa antes de empezar con el proyecto, aunque nunca tuvieron para mí una utilidad real. Probablemente en el futuro, se valore la creación de un reloj que mida ciertos parámetros como la temperatura de un paciente con una determinada enfermedad y un doctor pueda analizarla y tener constancia de varianzas graves desde su misma consulta.

Además de ello, es gratificante saber que un proyecto servirá de utilidad en el futuro para su uso en docencia, como algunos que yo he utilizado, de manera que, el alumnado pueda obtener una visión real haciendo diferentes pruebas de medición de temperatura, incluso ver el funcionamiento de cada uno de los sensores y su respuesta, pudiendo así salir del entorno de simuladores que tanto se utiliza actualmente en la enseñanza pero que, en multitud de casos, distan de la realidad o tienen un precio bastante alto.

1.1. OBJETIVOS Y MOTIVACIÓN:

El presente proyecto tiene por objetivo el concepto de diseño de un sistema de control de temperatura en tiempo real que utiliza un microcontrolador Arduino como plataforma de tratamiento y adquisición de datos, Raspberry para el envío de los mismos y LabVIEW como plataforma de recepción y control, lo que nos permitirá trabajar en red, y nos ayudará a verificar la fiabilidad y limitaciones de los sistemas con los que vamos a trabajar aplicado para su uso en educación [3].

Se llevarán a cabo las siguientes actividades:

- Revisión bibliográfica de los proyectos similares, e información de los elementos que se van a realizar, así como una estimación presupuestaria del mismo.
- Revisión de las distintas tecnologías para la medición de temperatura existentes a bajo precio, y que nos permitan siempre que sea posible controlarlas desde Arduino.
- Explotación de cada uno de los sensores.
- Programación de las diferentes consignas en Arduino y creación de un sistema de control PID.
- Encendido, configuración programación y puesta en marcha de Raspberry Pi para la transmisión de datos desde el puerto serie hasta su puesta en red, para la transmisión de datos hasta el cliente LabView.
- Análisis de protocolo de comunicación utilizado para el transporte de datos de temperatura.
- Creación de panel de control y monitorización de datos de temperatura en LabView.
- Establecer unas conclusiones del proyecto de sensores realizado.

1.2. ORGANIZACIÓN DE LA MEMORIA:

El proyecto se divide en cuatro grandes partes que en su mayoría es una división temporal, o lo que podríamos llamar fases, resolviendo los problemas que van surgiendo en las diferentes fases.

En la primera fase (apartado 2) vamos a ver un análisis de las tecnologías a utilizar, tanto de hardware de manipulación de datos (Arduino, Raspberry, etc.), hasta software (LabView, lenguaje de Arduino, lenguaje Python, etc.), y una visión general de algunos sistemas parecidos que veremos en el apartado.

En la segunda fase (apartado 3) vamos a ver el desarrollo del proyecto en sí, la caracterización de cada uno de los sensores, la programación de cada uno de dispositivos y su unión. También veremos la programación de la Raspberry de manera que nos acepte los datos que le aporta la Arduino y se los envíe al cliente LabView, y los diferentes problemas que se han solventado a lo largo de ella.

En la tercera fase (apartado 4) se realizará el montaje y puesta a punto del sistema, realizando el conexionado, la puesta a punto de la Raspberry y LabView y su configuración hasta conseguir el correcto funcionamiento del sistema y la realización de 3 pruebas diferentes, que nos permitirá realizar además un análisis estadístico para poder diferenciar las distintas tecnologías.

En la cuarta y última fase (apartado 7) se realizará un análisis de los datos que hemos obtenido al realizar dicho proyecto, además de incluir una conclusión y posibles trabajos futuros.

2. ANTECEDENTES Y ESTADO DEL ARTE:

2.1. AUTOMATIZACIÓN E IoT:

Hoy en día, las tareas de control y vigilancia de variables cada vez se destinan en mayor parte a aparatos electrónicos, que nos notifican cuando algo no va bien en nuestra planta. Esto es, en mayor parte, debido a que son tareas monótonas y aburridas, además de que se requiere un recurso humano, que es incapaz de mantener la atención más de un cierto número de horas, o de realizar ciertas tareas que requieran de una exigente precisión, mejorando así las condiciones laborables del humano.

Además, podemos tener un control más exacto y adecuado de la producción, y podemos llegar a hacer estimaciones futuras, además de reducir costes.

Gracias a esto, se creó la corriente *IoT (Internet of Things)* que consiste en el control remoto vía Internet, utilizando tecnologías de conexión inalámbrica como lo son Wi-Fi, GSM, Bluetooth, Zigbee, que son diferentes en cuanto a potencia y alcance, de manera que los dispositivos puedan interaccionar sin la necesidad de influencia humana, de aparatos que anteriormente no lo disponían e incluso ampliar sus posibilidades [2]. Tras esto, comenzaron a aparecer las *Smart Homes* y *Smart Cities* y otros, e incluso las *SmartBands* que llevamos muchísimas personas en nuestra muñeca. El único gran problema radica en el protocolo de comunicación, ya que actualmente no existe ningún estándar que los unifique, aunque se está intentando introducir *MQTT (Message Queuing Telemetry Transport)*, que es un protocolo abierto propuesto por *IBM* [5].

A continuación, veremos algunos de los elementos que se utilizan en muchos de los proyectos de *IoT*, sobre todo en los proyectos *DIY (Do It Yourself)* o como comúnmente podríamos decir, hechos por ti mismo, haciendo hincapié en aquellos que vamos a utilizar.

Dentro de este hardware tan conocido entran la placa Arduino y sus alternativas, y también Raspberry. Al juntarlas, ambas se compenetran bien ya que el procesamiento de datos y la creación de proyectos bajo C en el Arduino es de los más sencillos que existen actualmente. El gran problema que tiene Arduino es que no dispone de una conexión a internet, y aunque actualmente se hayan creado modelos que dispongan de ello, o se le ponga una tarjeta de Wi-Fi o Ethernet nunca se va a conseguir la experiencia de un sistema operativo basado en Linux que nos ofrece esta segunda. Además, entre ella

se comunican por el puerto serie e incluso tenemos el IDE de Arduino disponible para Raspberry.

Existen multitud de utilidades con las que podemos utilizar esta combinación, principalmente para la monitorización de datos, desde la utilización de la plataforma Arduino para la enseñanza con sensores [6], también la utilización de una Raspberry como servidor a modo de implementar una arquitectura basada en IoT [7], utilizando la Raspberry con nodos Zigbee y comunicándola utilizando PHP y HTML para crear una aplicación web donde emitir los datos [8].

Por último, también conocemos multitud de propuestas para trabajar con una Arduino y LabView utilizando el USB [9], e incluso la utilización de los mismos integrantes utilizados midiendo temperatura y humedad para fines médicos [10].

2.2. MICROCONTROLADOR ARDUINO:

Arduino es una plataforma de creación de electrónica de código abierto, flexible y fácil de utilizar que, junto con un lenguaje de programación *Processing/Wiring* y el bootloader, podemos realizar multitud de proyectos. [11]

Es una placa que trabaja bajo un microcontrolador ATMEL, en el que grabamos instrucciones. Para ello, utilizamos el lenguaje C adaptado al entorno Arduino. Las instrucciones permiten la creación de pequeños programas que interactúan con los periféricos de la placa.

Arduino posee una interfaz para las entradas que nos permite la conexión de la placa con periféricos de entrada y una interfaz de salida que se encarga de llevar la información a las salidas de un microcontrolador, y que puede ser programada tanto en Windows como macOS y GNU/Linux. Un proyecto que promueve la filosofía 'learning by doing', que viene a querer decir que la mejor manera de aprender es “cacharreando”. [12]

Existen multitud de placas Arduino, según las especificaciones y tamaño que necesitemos. Algunas de ellas, son las que podemos ver en la siguiente imagen, aunque para nosotros la selección precio/características se sitúa entre el modelo UNO o NANO.

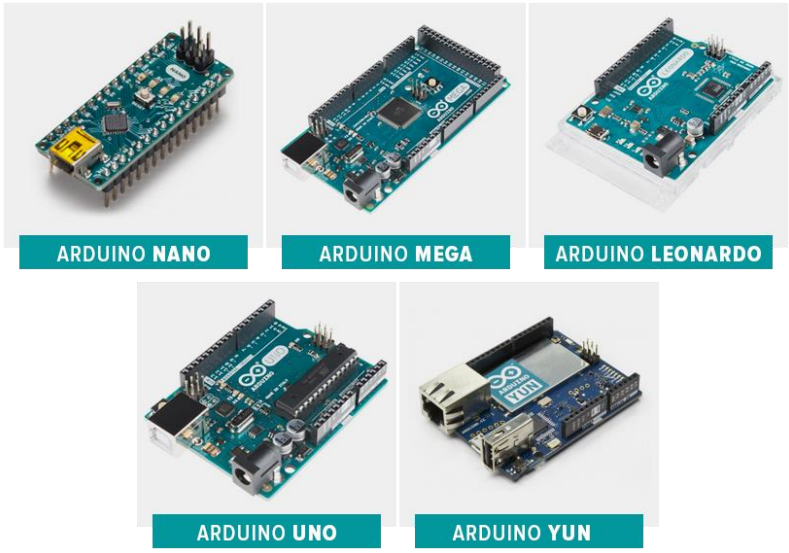


Figura 1: Tipos principales de Arduino

2.2.2. El Arduino NANO:

El Arduino NANO[1] es uno de los integrantes de la familia Arduino. Es muy similar al modelo UNO, que es el más utilizado en la actualidad, aunque presenta una pequeña diferencia en cuanto a tamaño y la alimentación de la placa, ya que se alimenta únicamente mediante mini-B USB. Podemos ver una primera impresión de la placa y a continuación una tabla con las características:

Figura 2: Arduino NANO

[1] Consulte más información en la página original en https://store.arduino.cc/arduino-nano

Microcontrolador	ATmega328
Memoria FLASH	32 KB
Velocidad de Reloj	16 MHz
Pines Entrada Analógicos	8
Corriente DC I/O	40 mA
Voltaje Entrada	7-12 V
Pines Digitales I/O	14 (6 PWM)
Tamaño	18 x 45 mm

Tabla 1: Características Arduino Nano

Podríamos aprovechar y utilizar una Arduino nano para nuestro proyecto, pero debido al gran número de sensores que tenemos conectados, al conectar la Arduino a la Raspberry ha ocurrido que algunas veces, ésta, se apaga por consumo. Esto va a indicar que no nos vale con solo la entrada de USB que proviene de la Raspberry, si no que tendremos que alimentarla por independiente. Esto nos lleva a utilizar la Arduino UNO, aunque es perfectamente intercambiable en cuanto a funciones.

2.2.3. El Arduino UNO:

El Arduino UNO[2], tal y como se ha mencionado, existen muy pocas diferencias entre ambas placas, entre las que destacan las entradas/salidas, el tamaño y la alimentación. Se adjunta una tabla con las características de esta placa:

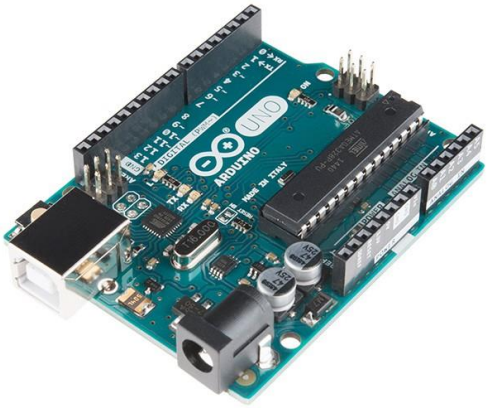


Figura 3: Arduino UNO

[2] Consulte más información de su última versión en https://store.arduino.cc/arduino-uno-rev3-smd

Microcontrolador	ATmega328P
Memoria FLASH	32 KB
Velocidad de Reloj	16 MHz
Pines Entrada Analógicos	6
Corriente DC I/O	20 mA
Voltaje Entrada	7-12 V
Pines Digitales I/O	14 (6 PWM)
Tamaño	68.6 x 53.4 mm

Tabla 2: Características Arduino UNO

2.3. RASPBERRY PI:

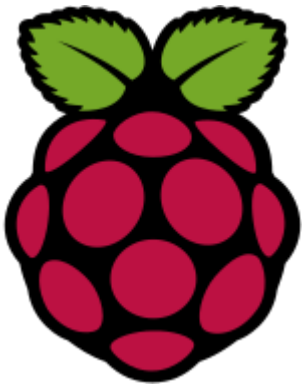


Figura 4: Logo Raspberry Pi Foundation

Raspberry Pi es un ordenador muy pequeño y de bajo coste desarrollado en Reino Unido por la Raspberry Pi Foundation, para estimular la enseñanza de informática en las escuelas, pero que permite su uso libre tanto a nivel educativo como particular. El modelo original se convirtió en más popular de lo que se esperaba, y se vendió para usos muy distintos lo puede ser la robótica. No dispone de periféricos (como teclado y ratón) ni carcasa [13].

Trabaja bajo un software open *source*, adaptado de Debian, llamado Raspbian, aunque existen multitud de sistemas operativos disponibles, incluido Windows 10, y trabaja principalmente bajo Python. Incluyen un procesador Broadcom, RAM, GPU, Ethernet (el primer modelo no lo tenía), HDMI, 40 pines GPIO (desde la Raspberry Pi 2), puertos USB, y posibilidad de incorporar una cámara. No incluye memoria ROM, pero si un slot para tarjetas MicroSD.

La organización dispone de una fundación caritativa con el objetivo de promocionar la informática en colegios en países en desarrollo.

2.3.1. LA RASPBERRY PI 2:

Figura 5: Vista frontal de la Raspberry Pi 2

Este modelo fue lanzado en 2014. Aumentaron considerablemente las características técnicas. Sustituye el procesador BCM2835 usado anteriormente por el BCM2836, pasando de un procesador de un núcleo a cuatro, y aumentando la velocidad de 700 a 900MHz [14].

Además, cuenta con 1 GB de RAM compartida con la gráfica, 40 pines GPIO, y cuatro puertos USB, como añadido a los puertos ya mencionados anteriormente. Ésta nos va a permitir trabajar sin problemas con UDP, que es el uso final de dicho trabajo, aunque más tarde veremos que si queremos utilizar Firebase como método predeterminado, o añadir algunas funciones más, quizás tendríamos que cambiar de modelo. En la actualidad se ha presentado la versión 4 de la placa que cuenta con el procesador Broadcom BCM2711 que dispone de cuatro núcleos que funcionan a 1.5 GHz y versiones desde los 1 a 4 GB de RAM y USB-C. En la siguiente figura se realiza una comparación de los modelos más vendidos con sus principales diferencias, aunque finalmente se utilice el modelo 2 por su disponibilidad y que tiene la capacidad suficiente para trabajar con UDP.

	Raspberry Pi 4 B	Raspberry Pi 3 B+	Raspberry Pi 2
Imagen			
Release date	24/6/2019	14/3/2018	1/2/2015
Product details			
Precio	35,00 $	35,00 $	35,00 $
SOC			
SOC Type	Broadcom BCM2711	Broadcom BCM2837B0	Broadcom BCM2836
Core Type	Cortex-A72 (ARM v8) 64-bit	Cortex-A53 64-bit	Cortex-A7
No. Of Cores	4	4	4
GPU		VideoCore IV	VideoCore IV
CPU Clock	1.5 GHz	1.4 GHz	900 MHz
RAM	1 GB , 2 GB, 4 GB	1 GB DDR2	1 GB
Wired Connectivity			
USB			
Ethernet			
Wireless Connectivity (On-Board)			
Wi-Fi			✖
Bluetooth®			✖
Dimensiones			
Altura	3.37 in (85.6 mm)	3.37 in (85.6 mm)	3.37 in (85.6 mm)
Ancho	2.22 in (56.5 mm)	2.22 in (56.5 mm)	2.22 in (56.5 mm)
Profundidad	0.43307 in (11 mm)	0.66929 in (17 mm)	0.66929 in (17 mm)
Peso		1.58 oz (45 g)	1.58 oz (45 g)
Power			
Power ratings		1.13 A @5V	800 mA
Power sources	USB-C	microUSB, GPIO	microUSB or GPIO
Power Over Ethernet	⊖ with PoE Hat	⊖ with PoE Hat	✖

Tabla 3: Comparación características modelos Raspberry por socialcompare.com

2.4. TCP/IP e UDP:

Hoy día, la comunicación más extendida es vía Wi-Fi, por el bajo precio de su hardware y su correcto alcance y expansión. Además, podemos tener o no tener acceso a internet, algo que nos brinda un mundo de posibilidades. Siguiendo el modelo ISO de interconexión de sistemas abiertos, más conocido como OSI[3] (Open System Interconnection), vamos a analizar ahora la capa de transporte de Internet, donde nos encontramos con dos protocolos principales, que son UDP y TCP, que utilizan redes IP, que es un protocolo encargado de unir diferentes redes, pero que solo no es muy fiable. Por ello vamos a comentar algunas de las tecnologías con las que trabaja y sus diferencias [15]:

- TCP: (Transmission Control Protocol) se orienta a la conexión. Al enviar datos el destinatario es informado de la llegada de estos, y confirma su buena recepción. Se utiliza principalmente en HTTP. Si los datos recibidos están corrompidos, el protocolo TCP permite que se solicite al emisor que vuelva a enviarlos, algo que también se puede en UDP pero no nos aseguramos la correcta entrega de estos. Dispone de un tamaño máximo de bytes y se debe de finalizar la conexión. [16]

- UDP: (User Datagram Protocol) es un protocolo no está orientado a una conexión y se utiliza principalmente en el protocolo DHCP. El flujo solo se da en un sentido. La transferencia de datos se realiza hacia el destinatario sin enviar la confirmación de recibido al emisor, simplemente abriendo un socket. No transmiten la información relacionada al emisor, excepto su IP, por lo que su encabezado ocupa menos y es más rápido que TCP [17].

[3] Más información acerca de los niveles del modelo OSI https://es.wikipedia.org/wiki/Modelo_OSI

2.5. FIREBASE:

Figura 6: Logo Firebase

Firebase es una agrupación de herramientas que permite al usuario facilidades a la hora de crear una aplicación de gran calidad, y que están disponibles para muchos entornos [18], dentro de las cuales destacan:

- Base de datos Realtime: Ésta es la que más nos interesa. Almacena datos en formato JSON y se pueden agregar reglas de lectura y escritura.
- Autenticación: Es un servicio que nos simplifica el inicio de sesión, ya que nos permite utilizar nuestras cuentas de servicios conocidos como Facebook.
- Almacenamiento: Especialmente para guardar archivos del usuario.
- Y muchas más que no son necesarias para dicho trabajo. Podemos consultar más información en la web oficial de Firebase[4].

Lo mejor y más importante de esta tecnología es que nos ofrece una base de datos en tiempo real en la web que nos permite conectarnos desde donde queramos, siempre y cuando cumplamos las reglas, extremadamente rápida y gestionada por Google, por lo que también tiene una alta fidelidad, aunque no es la única.

Otra cosa a mencionar de esta base de datos es que sigue una estructura estandarizada que usa el formato JSON. Un paquete JSON tiene la siguiente estructura:

[4] Consulte las aplicaciones que ofrece en https://firebase.google.com/products

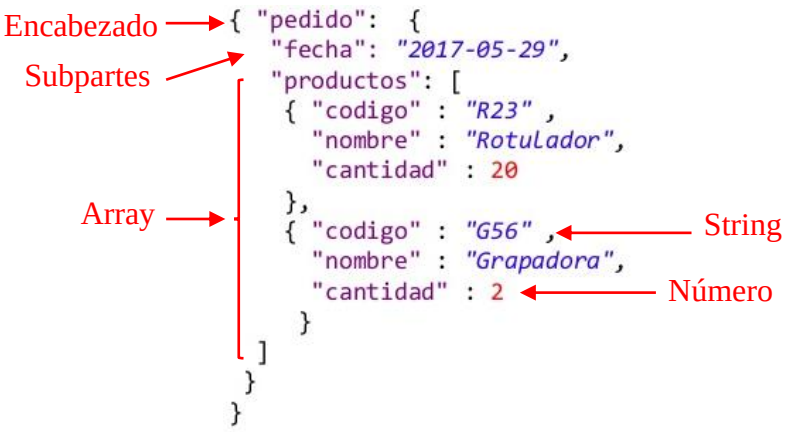


Figura 7: Descripción de JSON

2.5.1. REALTIME DATABASE:

Firebase Realtime Database es un servicio de almacenaje y sincronización de datos con bases de datos que no están basadas en SQL (NoSQL) ubicada en la nube y que permite la sincronización con todos los clientes en tiempo real y sigue estando disponible, pese a que no estemos conectados a Internet [19].

Trabaja con paquetes de datos JSON. Cuando se crea una app multiplataforma, todos tus clientes presentan una instancia de Realtime Database y se actualizan automáticamente [20].

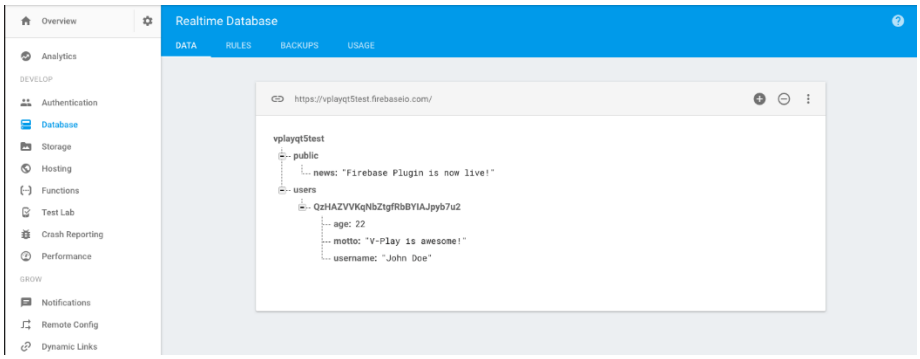


Figura 8: Ejemplo Realtime Database

Sus funciones principales son:

- *Tiempo real:* en vez de crear una solicitud de HTTP normal, sincroniza los datos. Cuando los datos varían, los dispositivos conectados reciben el cambio en milisegundos, lo que permite trabajar colaborativamente.

- *Trabajo sin conexión:* El SDK Firebase Realtime Database aloja tus datos en tu dispositivo hasta que vuelvas a tener conexión. En ese momento, las modificaciones que se hayan realizado se sincronizaran con el servidor.
- *Accesibilidad desde los dispositivos del cliente:* Es multiplataforma y nos permite acceder desde apps o directamente desde el navegador. La seguridad y la validación de datos se gestionan mediante las reglas de Firebase Realtime Database, que limitan la lectura y escritura de datos, lo que puede permitir que se deniegue la entrada de ciertos datos si no cumplen con las reglas, que podremos ver en la imagen siguiente:

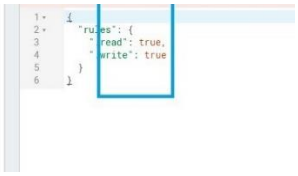


Figura 9: Ejemplo reglas Realtime Database

Esta aplicación también te permite crear apps con acceso seguro a la base de datos de forma directa desde el cliente, si lo integramos con Firebase Authentication [21]. Los desarrolladores deciden quién puede acceder a los datos, además de permitir la autenticación utilizando plataformas conocidas como GitHub, Facebook, Twitter, etc, y herramientas como la verificación de correo electrónico, o incluso guardar esos usuarios o colecciones en la base de datos como podemos ver en la siguiente imagen:

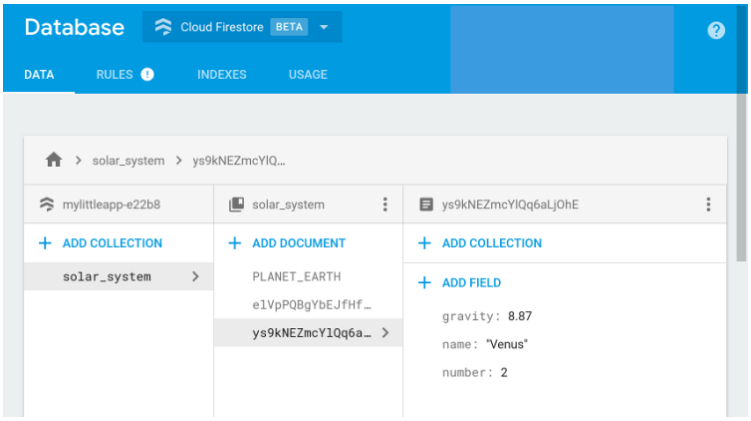


Figura 10: Ejemplo colección Realtime Database

En este trabajo no se implementará dicho método de autenticación ya que LabView no nos facilita este método, que se tiene que hacer desde el cliente mediante el uso de un token, además de no desaprovechar el tiempo en introducir cuentas y poder hacer lo más parecido posible para el cliente la comunicación por UDP y por Firebase.

2.6.LABVIEW:

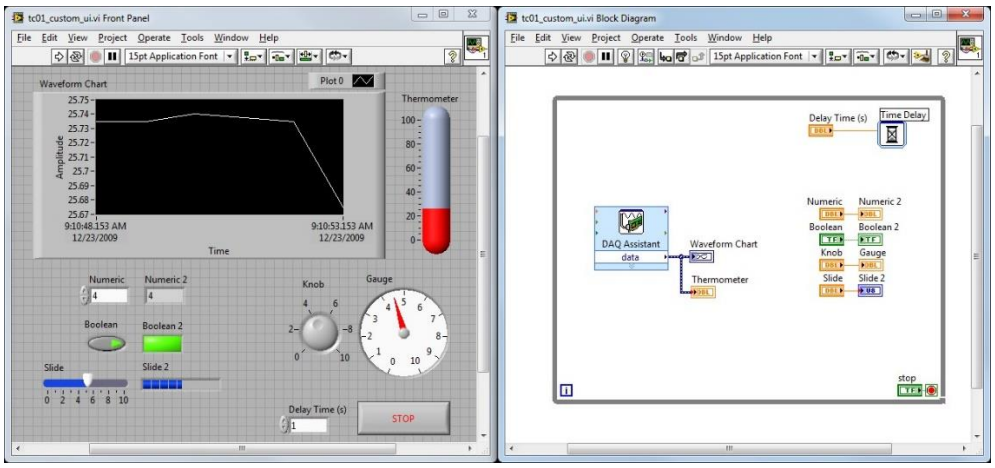


Figura 11: Paneles frontal y diagrama de bloques de LabView

LabView es una plataforma que permite desarrollar y diseñar sistemas, utilizando un lenguaje de programación gráfico pensado para sistemas hardware y software de pruebas, control y diseño, creado por National Instruments (1976) para Mac, pero fue ampliado para las demás plataformas, utilizando código *open source*.

Los programas de LabView se denominan Instrumentos Virtuales, o VIs, y su origen proviene del control de instrumentos, aunque actualmente no se utiliza únicamente en el mundo de electrónica. Como vemos en la imagen anterior, presenta dos ventanas, una de visualización y la otra de programación gráfica. Entre sus objetivos están el reducir el tiempo de desarrollo de aplicaciones de todo tipo y acercar la informática a todo tipo de profesionales. Además, LabView es fácilmente combinable con elementos tanto del mismo fabricante, como de otros.

2.6.1. LAS LIBRERÍAS PARA TCP/IP Y UDP:

Como ya hemos mencionado anteriormente, LabView tiene una amplía librería para todo tipo de utilidades que podemos encontrar en el mismo programa, además de una amplia ayuda. Entre ellas, existe un apartado para comunicación con diferentes protocolos.

Podríamos utilizar tanto las librerías de TCP o UDP según el protocolo utilizado. En este caso, como la perdida de información no es importante vamos a utilizar UDP

como protocolo por su rapidez y su menor tamaño, aunque queda extrapolable a utilizar otros protocolos.

Tal y como hablábamos, LabView presenta una librería para trabajar con los protocolos mencionados que tiene los siguientes bloques:

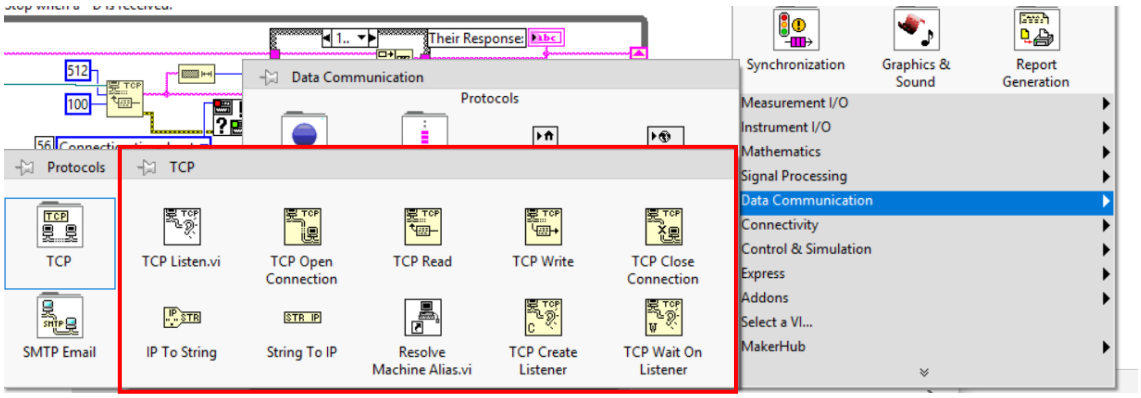


Figura 12: Librería TCP/IP LabView

Existen 10 bloques para diferentes tareas, pero nos centramos en los que nos permiten abrir y cerrar conexiones IP, además de la posibilidad de transmitir y recibir datos, que corresponde a la primera fila.

En el caso de UDP los bloques disponibles son:

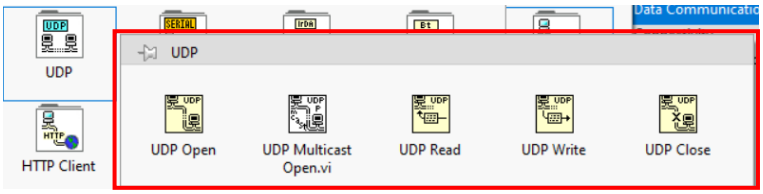


Figura 13: Librería UDP Labview

Si queremos información acerca del funcionamiento de cada uno de los bloques mencionados, podemos acudir a la ayuda en LabView o en el apartado 8.3 de la bibliografía [22].

2.6.2. LAS LIBRERÍAS HTTP Y JSON PARA LABVIEW:

Labview no dispone actualmente de una librería dedicada a Firebase exclusivamente, si no que incluye una librería general para trabajar con archivos JSON, que ya hemos hablado de ellos en el apartado de Firebase, que son los paquetes que vamos a enviar y que podemos trabajar con la siguiente librería:

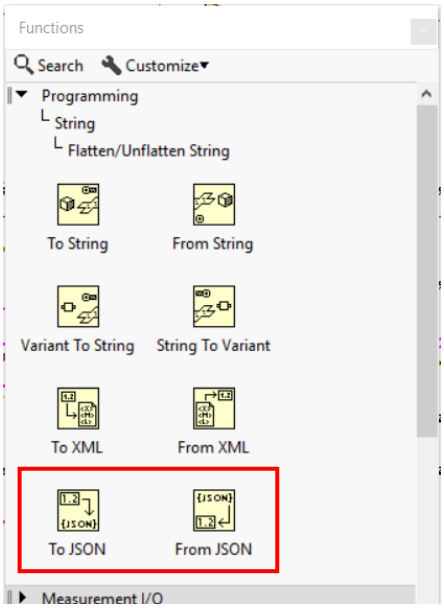


Figura 14: Librería JSON en LabView

Además, como se trata de un cliente web, para poder enviar y recibir estos paquetes de datos hemos usado la librería de protocolos entre los que está HTTP, que lo utilizaremos para enviar y recibir datos de la base de datos utilizando el conjunto de bloques de a continuación:

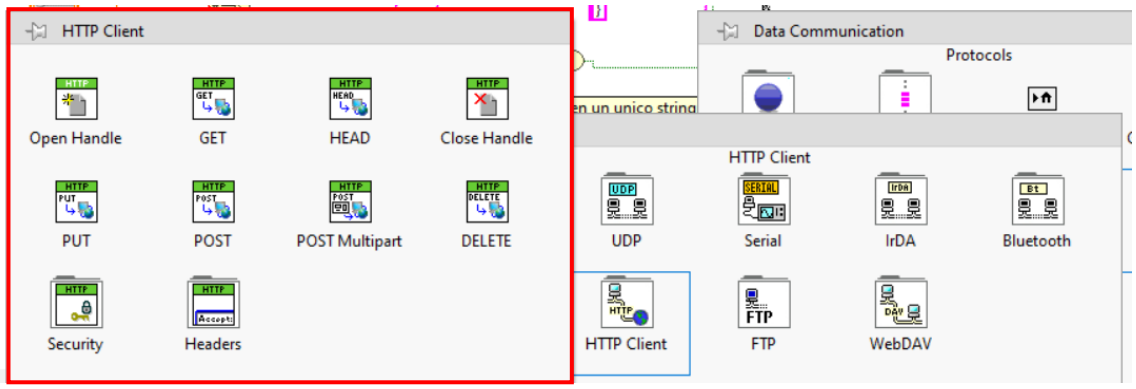


Figura 15: Librería HTTP en LabView

Podremos ver más información acerca de las librerías de LabView en la ayuda de LabView o en el *apartado 8.4. de la bibliografía* [22].

2.7. TECNOLOGÍAS SIMILARES:

Actualmente, y cada vez más, existen multitud de alternativas para Arduino y Raspberry Pi, incluso estas mismas se van desarrollando día a día para ser más completas atendiendo sobre todo a las necesidades que reclama la comunidad y a los avances.

En el caso de Arduino, más que competencia tiene multitud de clones, en tiendas chinas podemos encontrar multitud algunas muy similares y otras que directamente le han quitado funcionalidades. Lo bueno es que la mayoría te permiten trabajar con el IDE de Arduino [23].

También existe otra gama de microcontroladores, pero con menos funcionalidad que Arduino, aunque tengan más potencia e incorporen bluetooth y Wi-Fi como es la gama de *ESP32*[5].

En el caso de Raspberry, sí que existe competencia además de clones. Entre las más destacadas podemos encontrar *BeagleBone*[6], que la tenemos en la figura siguiente, que tiene características similares y que es posible trabajar con multitud de lenguajes diferentes. También destaca la alternativa de Microsoft *Sharks Cove*[7], que, aunque sus características sean notablemente mejores, su precio se dispara [24].

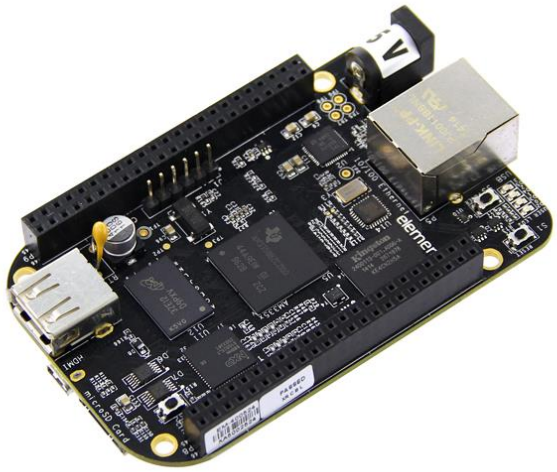


Figura 16: BeagleBone Black, alternativa a Raspberry

Sin embargo, las pioneras siguen siendo Arduino y Raspberry, tanto por su gran relación calidad-precio, como por la comunidad que tienen detrás desarrollando día a día

[5] Consulte información en https://www.espressif.com/en/products/hardware/esp32/overview
[6] Consulte información en https://beagleboard.org/bone
[7] Consulte información en https://www.prometec.net/sharks-cove/

multitud de proyectos al alcance de casi cualquier persona que tenga unas nociones de programación y electrónica.

3. DESCRIPCIÓN DE LA PROPUESTA:

3.1. ARQUITECTURA DE LA PROPUESTA:

Realizaré en este trabajo la implementación de diferentes tipos de medición de temperatura y su manipulación desde un microcontrolador, aprovechando la amplia posibilidad que nos aportan los módulos ya existentes.

Se propone que para conseguir dicho sistema se siga la siguiente estructura basada en placa Arduino[8], a la que conectaremos los diferentes sensores de medida de temperatura y que se desglosarán en la *sección 3.6. Caracterización de sensores*:

- Sensor integrado (DS18B20)
- Sensor RTD PT100
- Sensor basado en termopar tipo K
- Sensor de temperatura infrarrojo
- Sensor de tipo NTC

Esta placa Arduino estará conectada a una Raspberry Pi 2[9], que utilizaremos como servidor y que nos permitirá entrar en el sector de las IoT (*Internet of Things*), pudiendo acceder a los datos de forma remota, desde LabView[10] en nuestro caso, por ser una de las más utilizadas y que se enseñan y utilizan en la universidad, aunque existen muchas alternativas como pueden ser MyOpenLab[11], que está basado en Java.

La estructura general del sistema podemos verla en la siguiente figura, en la que la zona naranja nos muestra el sistema de control que estará semicerrada con la intención de que la medida de temperatura sea más estable, pero sin eludir el enfriamiento, además de intentar evitar las fluctuaciones debidas a la posición de los diferentes sensores utilizados.

Esta característica mencionada últimamente nos permitirá jugar con la posición de los sensores, que permitirá discutir si la posición en la que situamos los sensores de temperatura con respecto al foco de calor es indiferente o no en función de la forma de nuestra fuente de calor.

[8] Consulte más información en https://www.arduino.cc/
[9] Consulte más información en https://www.raspberrypi.org/
[10] Consulte más información en https://www.ni.com/es-es/shop/labview.html
[11] Consulte más información en https://myopenlab.org/inicio/

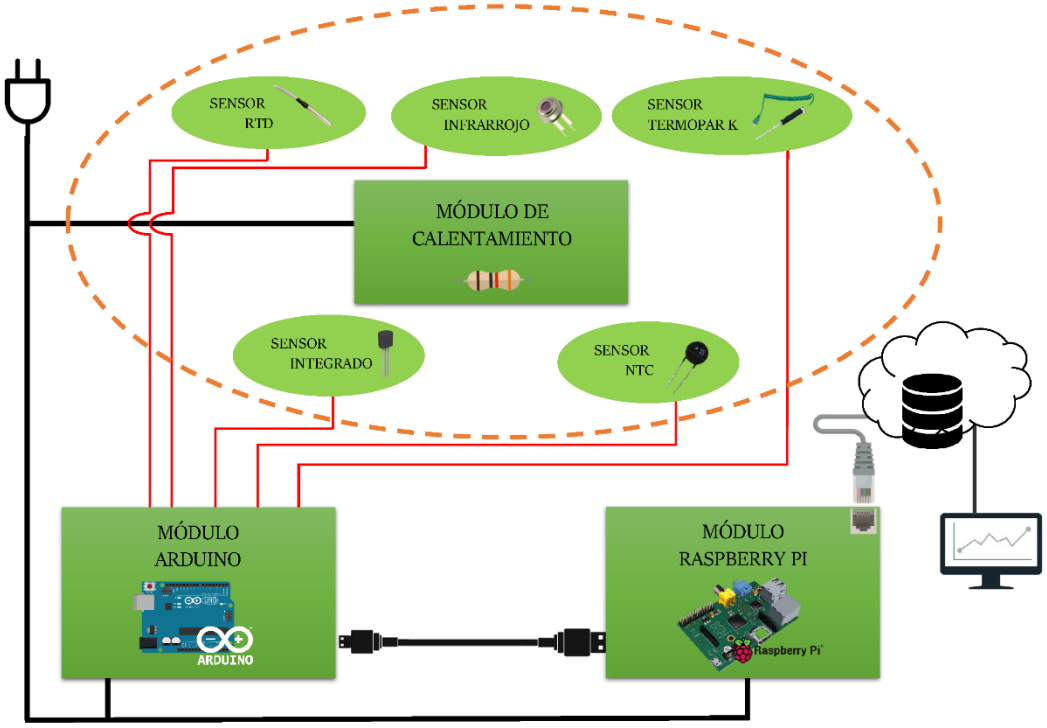


Figura 17: Arquitectura global del sistema

Dicho sistema nos permitirá trabajar a temperaturas cercanas a la de ambiente, entre [0-50] ºC, pero no se realizará enfriamiento, por lo que trabajaremos entre la temperatura ambiente y 50 ºC, aunque en la práctica trabajaremos hasta unos 40 ºC.

Cada uno de los sistemas de medida dispondrá de su propio acondicionamiento digital o analógico en su caso, preferiblemente digital por su facilidad de tratamiento y su mayor posibilidad en la tarjeta Arduino, a la que se le acoplará una *Shell* o placa de montaje, para unificar todos ellos.

Se realizará la programación en el lenguaje C aplicado de Arduino, para ello, utilizaremos su propia IDE. Para el trabajo de integración entre ésta, y la Raspberry trabajaremos en lenguaje Python bajo el software Linux adaptado a ella, *Raspbian* [25].

Una vez adquiridas todas las piezas, se realizará su montaje y conexionado, y se comprobará uno a uno su correcto funcionamiento, y se realizará una prueba general de funcionamiento para validar la consecución de los objetivos.

Esta propuesta se realiza en un marco temporal establecido, que podemos ver en el apartado 6 un desglose de las tareas y su planificación temporal, que fue ampliado debido a que el número de horas diarias se limitó drásticamente debido a la realización de las prácticas de empresa. También tenemos un apartado dedicado al marco

presupuestario ocasionado por la realización de dicho proyecto, donde no se ha introducido gastos de carácter material que han sido prestados, ni de la supuesta creación de un PCB o de la impresión en plástico de la caja contenedora de dicho proyecto, que, para llevarlas a cabo, precisan de la aprobación del personal y de dicho gasto.

3.2. COMUNICACIONES:

Una de las partes importantes de dicho trabajo consiste en las comunicaciones entre los diferentes componentes. Para dicha aplicación se hará uso de dos tipos de conexiones principalmente:

- *Conexión serie:* utilizaremos el puerto USB de Arduino y Raspberry para la comunicación entre ambas placas. No hace falta hacer ninguna configuración especial, ya que Raspberry se dedicará únicamente a leer el contenido que Arduino le comparte. A cambio Raspberry le compartirá información que incluye el modo de trabajo, temperaturas iniciales y final, tiempo y el sensor con el que vamos a realizar el PID [26].
- *Conexión IP:* utilizaremos dicho estándar para la conexión en red entre el cliente LabView y el servidor Raspberry. Para ello, tendremos que hacer uso de un código que hará dicho intercambio gracias al uso de *sockets* en Python, o de actualización de una base de datos, que podemos ver en el *3.2.2. Raspberry como bridge* de este mismo trabajo. Además, veremos otras posibilidades de trabajar con datos en la nube. También se hará una comparación entre dos protocolos importantes dentro de las conexiones para la transmisión de datos, como son TCP y UDP en el *apartado 2.4.* en donde comparamos ambas tecnologías.

3.3.ARDUINO COMO CONTROLADOR:

La función de Arduino en este caso va a ser la de simplificar el sistema, debido a que existen multitud de módulos que nos acercan todas estas tecnologías de medición a un tamaño mínimo y muy bajo coste, además de aislar un sistema de otro.

Visto de otra manera, el Arduino es el componente principal que nos va a permitir conocer el estado de los sensores y utilizamos la Raspberry para poder acceder a internet.

El Arduino va a tener dos funciones principales, que son la recogida de datos de temperatura por cada uno de los sensores y la puesta en acceso para Raspberry mediante el puerto serie, además de para el caso del cliente Master, permitir la configuración de la referencia que queremos aplicar para el control de nuestro sistema utilizando o Firebase o UDP, además de incluir un sistema de seguridad de cara a no poder calentar más de una cierta temperatura y otras características como la de enviar dos controles a la vez para aumentar la seguridad del sistema. Por último, se adjunta un diagrama de flujo asociado al funcionamiento del programa:

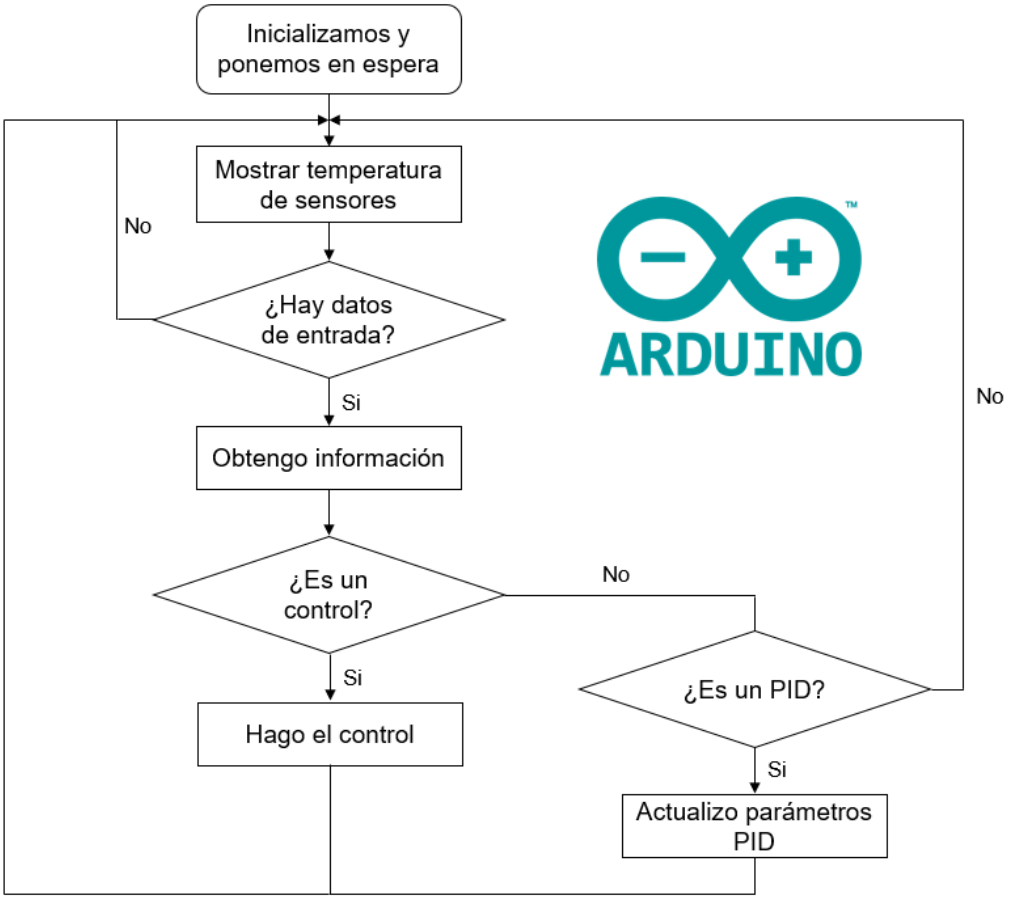


Figura 18: Diagrama de flujo programa Arduino

Para el desarrollo del programa se han utilizado librerías de componentes que veremos en el apartado 3.6 que podemos ver en el ANEXO 1: LIBRERÍAS DE SENSORES DE ARDUINO.

3.4.RASPBERRY PI EN MODO BRIDGE:

La función de la Raspberry Pi es un mero bridge entre la Arduino que hace de Master con los sensores (que serán los esclavos), y los dispositivos externos conectados remotamente que dispongan del panel de LabView creado a tal fin, tal y como vemos en la figura [27]:

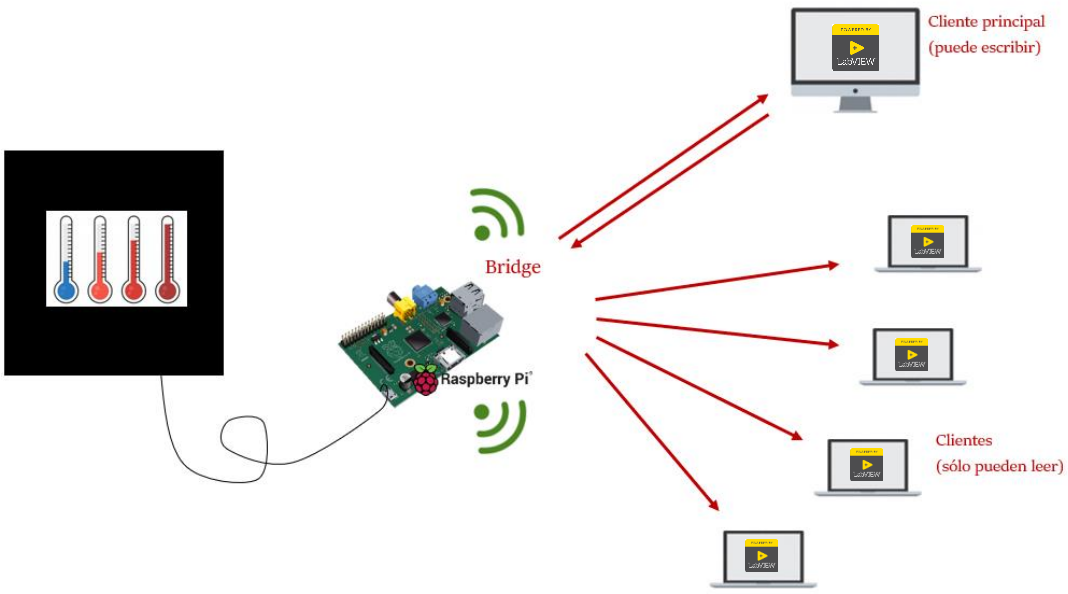


Figura 19: Raspberry actuando como bridge

Para evitar los inconvenientes que se puedan ocasionar al enviar desde varios clientes a la vez un comando para que realice una determinada señal de entrada, he desarrollado un programa similar pero que no autorice a ellos el envío de dichas señales, de manera que, únicamente podrá enviar estas señales el dispositivo con una ID estipulada anteriormente, que mandará una trama ligeramente distinta para limitar el tráfico externo, y que, además, su envío estará restringido con contraseña.

Para poder hacer que la Raspberry actúe como puente entre los datos que nos ofrece la Arduino y los datos que recibe y trasmite Labview, desarrollé un programa que realice dicho trabajo, además de la característica mencionada anteriormente de configuración.

Como ya he mencionado anteriormente, voy a permitir dos métodos de tratamiento de datos en función de si trabajamos con Firebase o si trabajamos enviando datos por UDP.

El programa basado en Firebase tendrá un diagrama de flujo como el que podemos ver a continuación que se ejecutará mientras el programa de LabView tenga abierta la sesión:

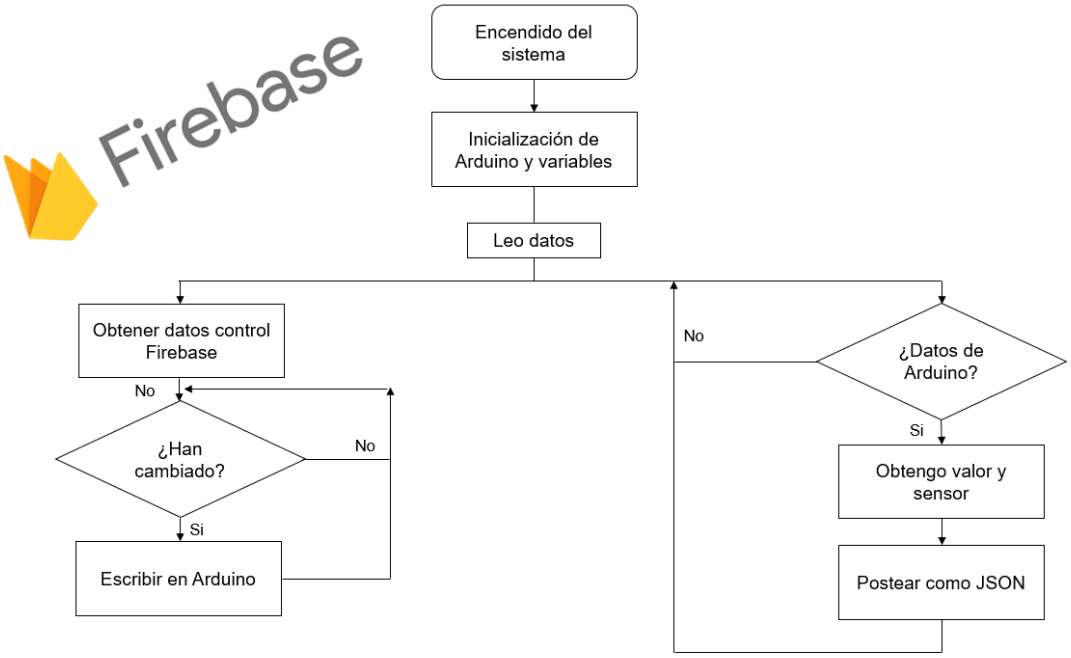


Figura 20: Diagrama flujo programa Firebase

En el siguiente diagrama podremos ver de manera más general, la comunicación mediante Firebase:

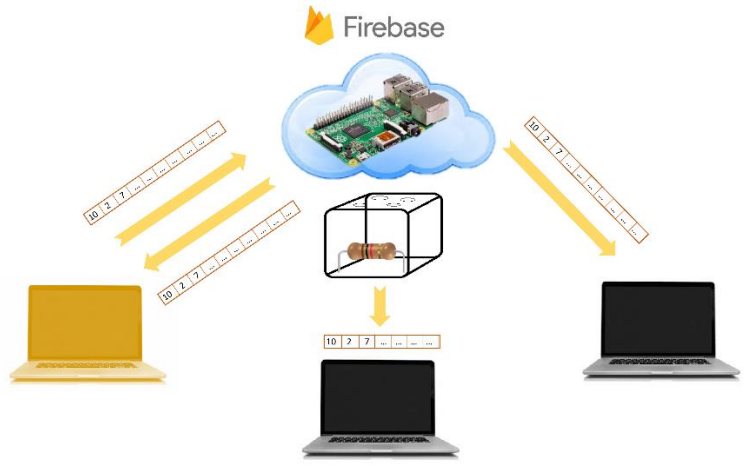


Figura 21: Visión general comunicación Firebase

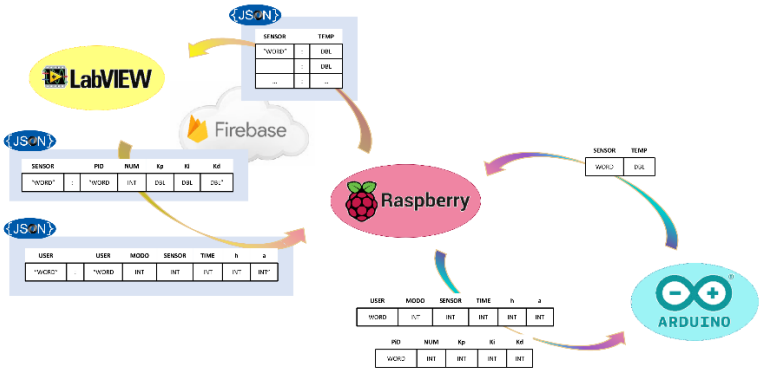


Figura 22: Análisis de datos en la comunicación por Firebase

El diagrama de flujo que veremos a continuación será para el caso en el que trabajemos con envío de datos a través de UDP. Aquí se realizará el envío de las temperaturas hacia los clientes de Labview mediante multicast de datos, mientras que la orden de control hacia el Arduino se hará utilizando una conexión normal en un puerto diferente.

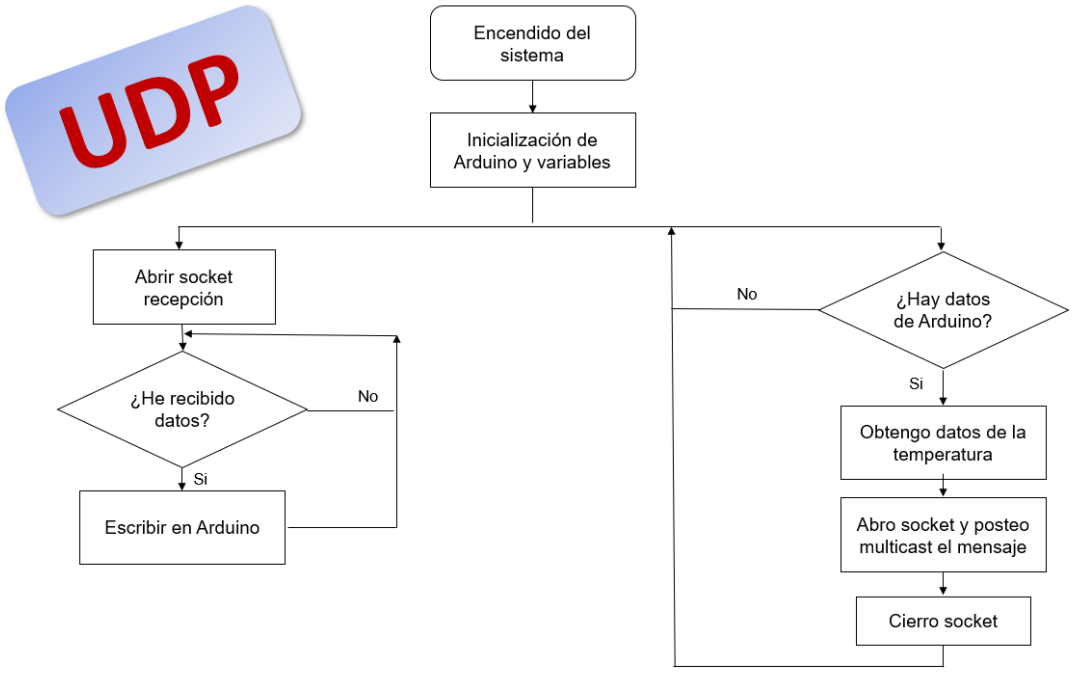


Figura 23: Diagrama de flujo programa UDP

Ambas metodologías harán uso de hilos, teniendo un hilo ejecutando el envío de datos y otro realizando la recepción de los datos para así aumentar la tasa de transferencia de datos y evitar la pérdida de datos en el camino además de porque en el caso de UDP,

la función de espera no permite continuar con la ejecución del código si no recibe información de entrada, a menos que le pongamos un *timeout*.

En los siguientes diagramas podemos ver la comunicación por UDP de una forma más general:

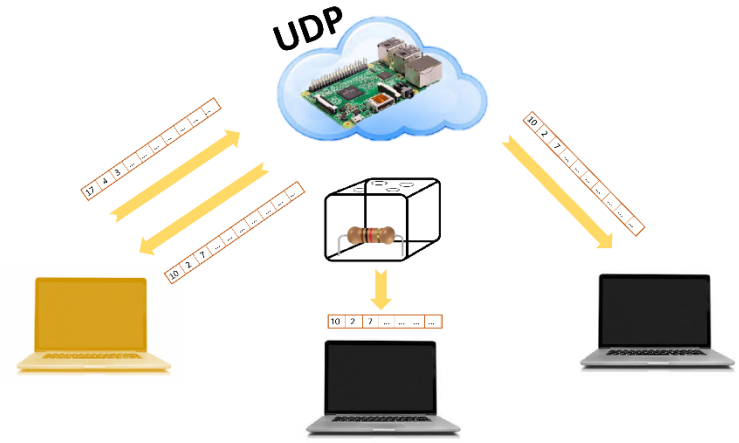


Figura 24: Visión general comunicación UDP

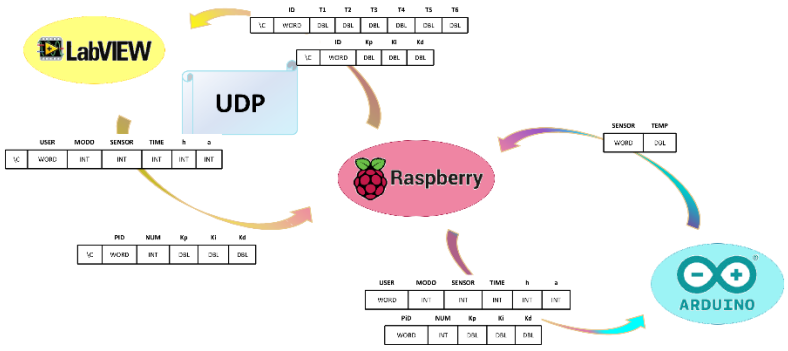


Figura 25: Análisis de datos en la comunicación por UDP

3.5. LABVIEW COMO CLIENTE:

De manera similar a lo que ocurre con el código que ejecutamos en la Raspberry, nuestro programa de Labview es ligeramente diferente. El programa desarrollado en Labview actúa como cliente de la Raspberry, que actúa como servidor. Este cliente puede ser de dos tipos, según la distinción ya realizada en el apartado (Raspberry pi):

- *Cliente Master:* tendrá la posibilidad de escribir las diferentes entradas que se le aplicarán a la resistencia calefactora, además de la capacidad de leer los valores de los transductores, y poder modificar las variables asociadas al PID escribiendo el código correcto donde se precisa. Se ha suprimido esta restricción por IP ya que implicaría tener que cambiarlo cada vez que se utiliza debido a que los router de la universidad disponen de servidor DHCP.
- *Cliente habitual:* únicamente podrá leer los valores de los transductores, aunque intente escribir datos de control, nunca llegarán a enviarse.

3.5.1. LABVIEW CON UDP/FIREBASE:

El panel de UDP y Firebase es muy similar tanto en la estética como en el uso para el cliente, aunque cambia el panel de programación.

Una vez realizada esta distinción del apartado anterior, que a ojos del participante no tendrán efecto ya que ambos tendrán el panel de mando, simplemente que el cliente master sabrá la contraseña para el posible envío de datos, y que se encontrará en desuso para el caso de los clientes habituales, pasamos a desarrollar ligeramente el funcionamiento de nuestra plataforma creada en Labview. Para más información se puede acudir al ANEXO 3: MANUAL PANEL LABVIEW.

Nuestra plataforma dispondrá de varias partes que desglosaremos a continuación:

- *Configuración IP*:

  Aquí podremos configurar el puerto de conexión establecido en LabView y nuestro programa en Python de manera que puedan realizar dicha conexión, que tendremos que introducir antes de iniciar la simulación. Para evitar problemas, se suprimirá dicho elemento y solo se podrá cambiar con la contraseña de modificación.

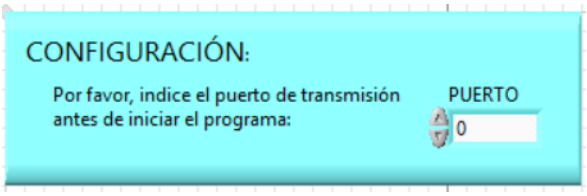


Figura 26: Configuración conexión LabView

- *Referencias de entrada*:

  Dispondremos de un panel de referencias para el cliente Master, que permitirá escoger entre diferentes posibilidades de entrada (escalón, sierra, escalones y rampa) establecer sus parámetros y el tiempo de simulación. Además, tendrá un botón de encendido para decidir cuándo enviarlo. Esto nos permite, ver los valores de los sensores cuando trabajan en reposo.

  Además de esto, en la parte de selección de PID podemos decir que sensor queremos que haga la acción de realimentación incluso cambiar sus parámetros de control antes de realizar la simulación.

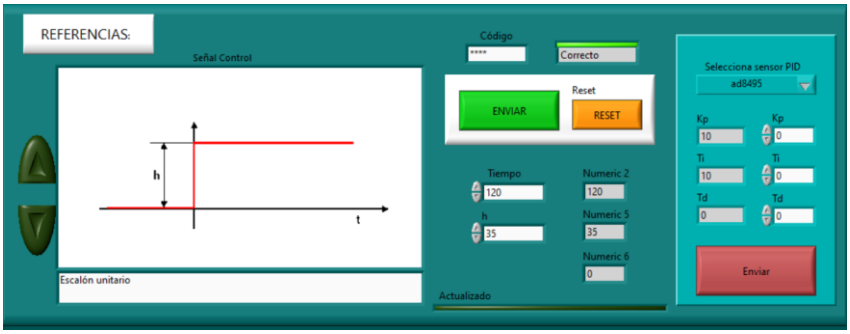


Figura 27: Configuración referencias LabView

En el caso de trabajar con UDP, no tenemos constancia de los números que hay puestos actualmente por lo que no tendremos la fila de datos, al igual que para los valores de kp, ki y kd. Para conocer el estado de simulación en el que nos encontramos podemos consultar el led de la placa que nos indicará si estamos en simulación, o la pantalla LCD.

- *Panel de lectura de datos*:

  En este panel vamos a poder visualizar la temperatura de los diferentes transductores de temperatura que tenemos, tanto en formato gráfico, texto y una gráfica temperatura-tiempo para que podamos ver la evolución que realiza a lo largo del tiempo:

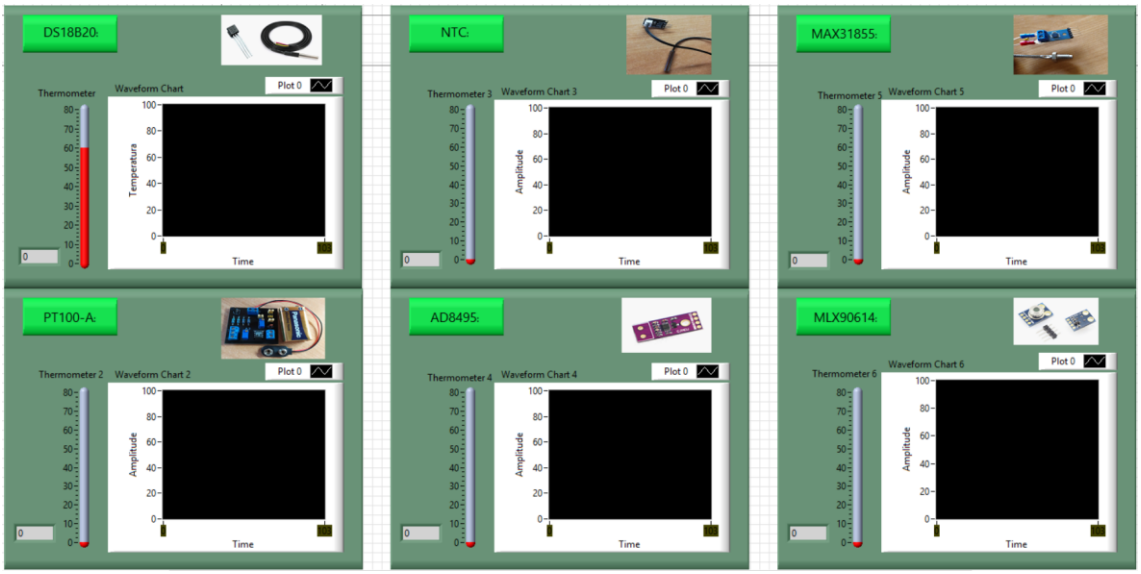


Figura 28: Panel de lectura de datos LabView

- *Panel de comparación de datos*:

  En este panel podremos visualizar la historia de cambios de temperatura que tienen cada uno de los sensores, pudiendo así además caracterizar el ruido que presenta cada uno de ellos en reposo y la variación ante diferentes cambios de temperatura, todo ello en una misma gráfica:

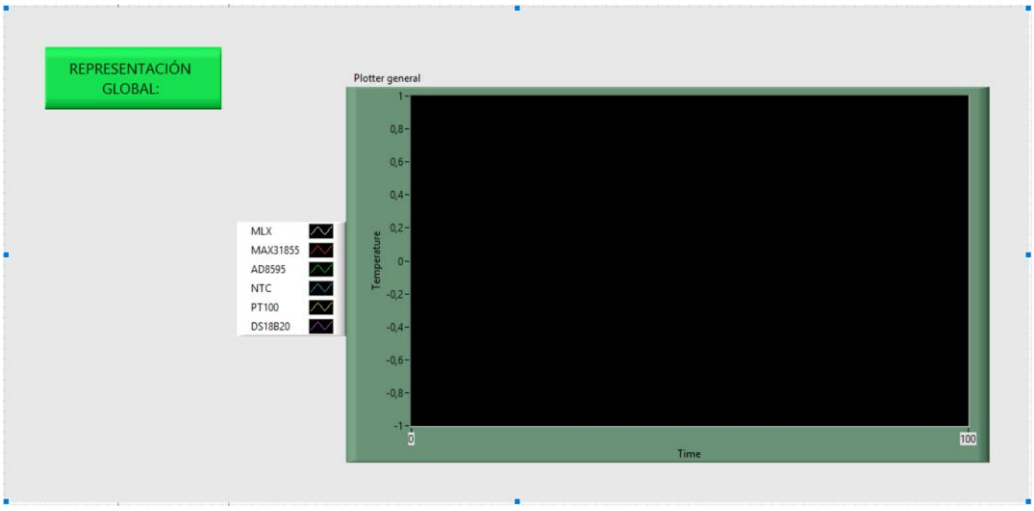


Figura 29: Panel de comparación LabView

- *Análisis estadístico de datos*:

  Las gráficas a veces no lo son todo, y más en este caso donde el espacio en donde se muestra no es lo grande que nos gustaría, y la amplitud temporal no nos permite tener un absoluto conocimiento. A pesar de que el waveform de Labview nos permita cambiar tanto el *span*, como el tiempo de visualización, no nos da una visión general del comportamiento.

  En este caso, utilizaremos la estadística para obtener algunos datos como la media, la desviación típica, la varianza, el rango y la mediana. Además, el waveform nos permite poder exportar los datos de la simulación a un Excel, y utilizando macros gratuitas de internet podemos hacer un análisis más exhaustivo de los datos.

Para una explicación más detallada diríjase al 8.3ANEXO 3: MANUAL PANEL LABVIEW acerca de la descripción del panel realizado en LabView.

3.6.CARACTERIZACIÓN DE LOS SENSORES:

En el mercado existen multitud de sensores para medir distintas magnitudes, como lo son la temperatura, presión, nivel, caudal, etc. Dentro de cada una, existen varias tecnologías, por ejemplo, para un sensor de caudal podemos tener tecnologías como Vortex, Electromagnético, Coriolis, ultrasonidos, etcétera según caudal, líquido o gas que queramos medir y otras limitaciones. En este caso, queremos caracterizar una serie de sensores de temperatura analizando las distintas posibilidades que podemos encontrar en el mercado, desde sensores infrarrojos, resistencias variables con la temperatura, termopares y demás, haciendo así un análisis general de las tecnologías de medición de temperatura y su comparación.

3.6.1. SENSOR DE INFRARROJOS (MLX90614):

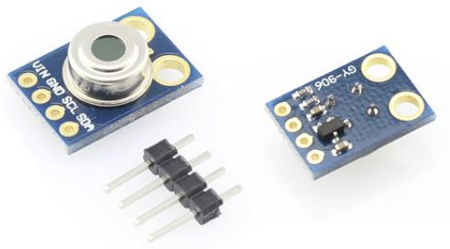


Figura 30: Sensor Infrarrojo MLX90614

Descripción:

El MLX90614[12] es un sensor infrarrojo sin contacto que recoge la radiación infrarroja emitida por cualquier objeto y transforma en un impulso eléctrico proporcional a la temperatura del objeto [28].

El integrado que vemos incluye un amplificador, un conversor analógico-digital de 17 bits, un procesador de la señal digital y una compensación de temperatura ambiente.

Le influye la suciedad en la zona de medida, y sólo es sensible a objetos que se sitúen en su rango de visión, que varía entre 5º y 80º. Además, en función de las temperaturas del objeto y ambiente tendremos una precisión diferente, aunque a nosotros no nos afecta ya que trabajaremos en el margen de 0.5ºC.

[12] Descargue el *datasheet* en https://www.melexis.com/-/media/files/documents/datasheets/mlx90615-datasheet-melexis.pdf

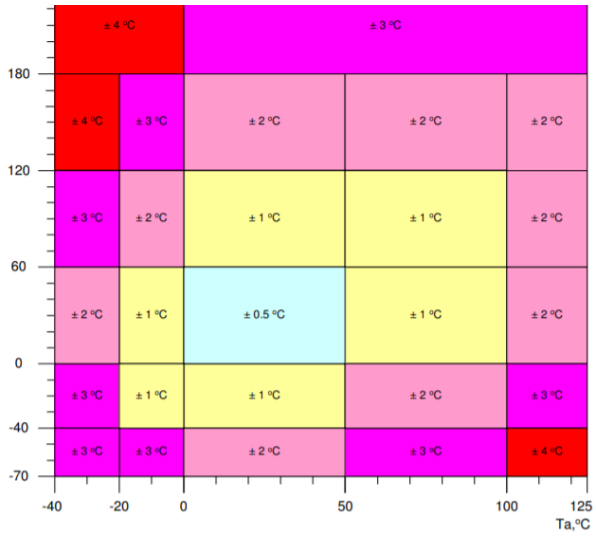


Figura 31. Precisión de medida en MLX90614

Características:

- Tamaño pequeño, bajo coste.
- Precisión de 0,5 ºC
- Permite trabajar con I2C con resolución de 0.02ºC, o con PWM con salida de 0.14ºC [29].
- Rango temperatura ambiente: -40 - 85 °C
- Rango temperatura de medida: -70 - 382 °C

Diagrama de conexiones:

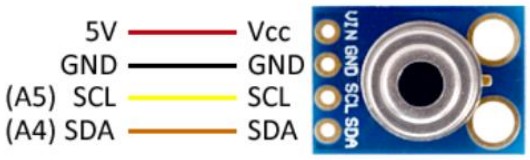


Figura 32: Diagrama conexión MLX90614

Código de prueba:

Previa introducción de las librerías Wire.h y la del componente.

```
//www.luisllamas.es
#include <Wire.h>
#include <Adafruit_MLX90614.h>

Adafruit_MLX90614 mlx90614 = Adafruit_MLX90614();

void setup() {
  Serial.begin(9600);

  mlx.begin();
}

void loop() {
  Serial.print("Tamb = ");
  Serial.print(mlx.readAmbientTempC());
  Serial.print("ºC\n TObj = ");
  Serial.print(mlx.readObjectTempC());
  Serial.println("ºC");
  delay(1000);
}
```

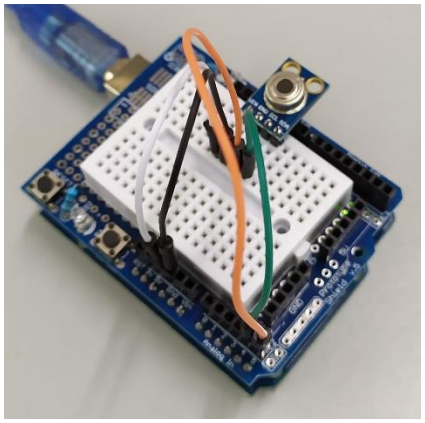


Figura 33: Montaje real MLX90614

Resultados:

```
Ambiente = 29.41°C      Objeto = 32.35°C
Ambiente = 29.41°C      Objeto = 33.63°C
Ambiente = 29.45°C      Objeto = 33.47°C
Ambiente = 29.45°C      Objeto = 32.33°C
Ambiente = 29.45°C      Objeto = 30.09°C
Ambiente = 29.47°C      Objeto = 28.05°C
Ambiente = 29.43°C      Objeto = 27.91°C
Ambiente = 29.43°C      Objeto = 27.59°C
Ambiente = 29.43°C      Objeto = 27.75°C
Ambiente = 29.43°C      Objeto = 27.87°C
Ambiente = 29.41°C      Objeto = 27.75°C
Ambiente = 29.43°C      Objeto = 27.85°C
Ambiente = 29.41°C      Objeto = 27.75°C
Ambiente = 29.43°C      Objeto = 27.81°C
Ambiente = 29.43°C      Objeto = 27.75°C
Ambiente = 29.43°C      Objeto = 27.79°C
Ambiente = 29.45°C      Objeto = 34.07°C
Ambiente = 29.43°C      Objeto = 34.83°C
Ambiente = 29.47°C      Objeto = 35.13°C
Ambiente = 29.49°C      Objeto = 35.39°C
Ambiente = 29.47°C      Objeto = 35.61°C
Ambiente = 29.55°C      Objeto = 28.19°C
```

Figura 34: Resultados prueba MLX90614

Conclusiones:

Una buena utilidad cuando necesitemos medir temperaturas de un cierto objeto con bastante rapidez, y con un gran campo de amplitud. La temperatura ambiente que nos marca no es la correcta pero no le prestaremos mayor importancia ya que no la vamos a utilizar y la del objeto mide correctamente. El gran problema puede ser su fragilidad, el problema de las fuentes de calor externas o la suciedad en la zona de medición, aunque tiene grandes ventajas como lo es su precisión y que al no presentar una vaina no tarda un rato en medir el enfriamiento, sino que es bastante rápido.

3.6.2. SENSOR DE TERMOPAR MAX31855:

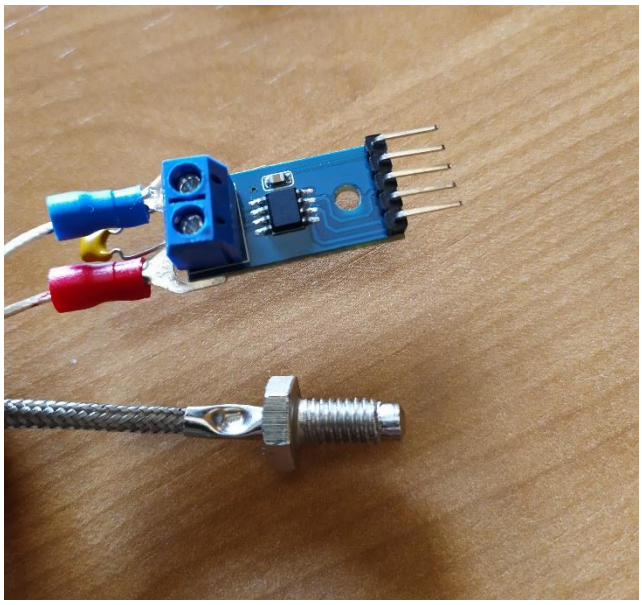


Figura 35: Sensor termopar digital MAX31855

Descripción:

El MAX31855[13] es un compensador de unión fría para termopar tipo K. Se trata de una actualización del MAX6675 que veremos a continuación. La única diferencia está en el rango en el que trabaja [30, p. 31855].

Este módulo se encarga de compensar la unión fría, además de amplificar y realizar una corrección en la medida de la sonda, aproximándola a un valor lineal.

La característica más relevante de los termopares es su amplio rango de medición, y su velocidad de medición, aunque no nos proporcionará una medida tan exacta como otros sistemas de medida.

Características:

- Alimentación a 3.3V o 5V
- Rango de operación -40ºC a 125ºC
- Rango de medición -200 a 700ºC con ±2ºC de precisión para termopares tipo K.
- Resolución de medida 0.25ºC.
- Trabaja con la interfaz SPI digital [31].

[13] Descargue el *datasheet* en https://datasheets.maximintegrated.com/en/ds/MAX31855.pdf

Diagrama de conexiones:

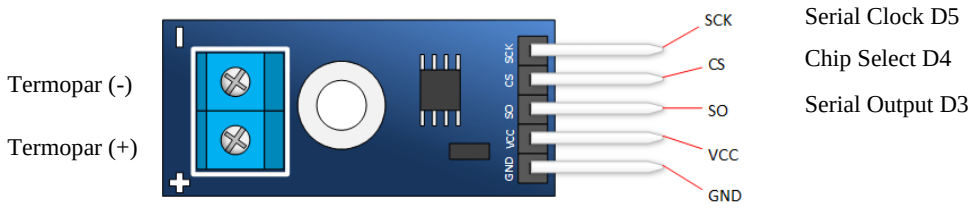


Figura 36: Diagrama de conexiones MAX31855

Código de prueba:

Previa introducción de las librerias SPI.h y la del componente.

```
#include <SPI.h>
#include "Adafruit_MAX31855.h"
#define MAXDO   3
#define MAXCS   4
#define MAXCLK  5

Adafruit_MAX31855 thermocouple(MAXCLK, MAXCS, MAXDO);

void setup() {
  Serial.begin(9600);
  delay(500);
}

void loop() {
   Serial.print("Internal Temp = ");
   Serial.println(thermocouple.readInternal());
   double c = thermocouple.readCelsius();
   Serial.print(c);
   Serial.println(" ºC");
   }
   delay(500);
}
```

Resultados:

```
Internal Temp = 31.75
42.75 °C
Internal Temp = 31.75
42.75 °C
Internal Temp = 31.75
43.00 °C
Internal Temp = 31.75
43.00 °C
Internal Temp = 31.75
43.00 °C
Internal Temp = 31.81
42.75 °C
Internal Temp = 31.81
43.25 °C
Internal Temp = 31.81
42.75 °C
Internal Temp = 31.81
43.25 °C
Internal Temp = 31.81
44.75 °C
Internal Temp = 31.81
46.75 °C
Internal Temp = 31.81
47.75 °C
Internal Temp = 31.81
48.25 °C
```

Figura 37: Resultados prueba MAX31855

Conclusiones:

Es uno de los sistemas más integrados debido a su gran rango de amplitud y rapidez, cuando la precisión no es lo más importante, ya que necesitará de un pequeño procesamiento para la compensación con la unión fría, que por cierto está un poco desviada. Aunque cabe recalcar que, debido al encapsulado, las variaciones de temperatura no son tan instantáneas, pudiendo compararlas con un encapsulado similar en una pt100. En teoría la velocidad de medida sin desviarse mucho es su fuerte, aunque en las pruebas no funciona demasiado bien. He tenido varios problemas con este sensor y con el MAX6675, tuve que reinstalar varias veces las librerías, las sondas me aportaban mucho ruido, y pese a trabajar sin librerías no obtuve el buen comportamiento que esperaba. Cabe mencionar que no era el oficial de Adafruit.

3.6.3. SENSOR DE TERMOPAR MAX6675 (ANTIGUO MAX31855):

Descripción:

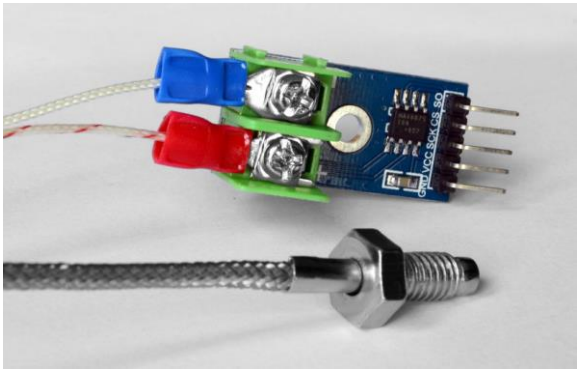


Figura 38: Sensor termopar digital MAX6675

El MAX6675[14] es un compensador de unión fría para termopar tipo K. Es muy parecido al caso anterior. La única diferencia está en el rango de trabajo [32].

Como ya hemos dicho, los termopares son muy eficaces para trabajar con amplios rangos de temperaturas, especialmente en los tramos altos.

Características:

- Alimentación de 3.3 a 5.5V
- Comunicación mediante interfaz SPI [31].
- Rango de medición 0-700ºC para mejor resolución.
- Precisión de ±3ºC.
- Tiempo de cálculo de unos 250ms.
- Rango de medida de la unión fría: -20 a 85ºC , aunque se recomienda menor al máximo.
- Resolución de medida 0,25ºC.
- Trabaja con la interfaz SPI digital.

[14] Descargue el *datasheet* en https://datasheets.maximintegrated.com/en/ds/MAX6675.pdf

Diagrama de conexiones:

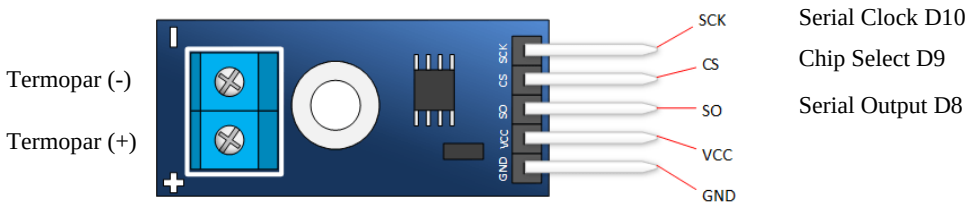


Figura 39: Diagrama de conexiones MAX6675

Código de prueba:

Previa introducción de las librerías del componente.

```
#include "max6675.h"

int SO = 8;
int CS = 9;
int CLK = 10;

MAX6675 sensor(CLK, CS, SO);

void setup()
{
  Serial.begin(9600);
  delay(500);
}

void loop()
{
   Serial.print("Graus C = ");
   Serial.println(sensor.readCelsius());
   delay(500);
}
```

Resultados:

```
Graus C = 33.50
Graus C = 33.25
Graus C = 32.50
Graus C = 33.25
Graus C = 33.00
Graus C = 33.00
Graus C = 33.00
Graus C = 33.25
Graus C = 33.25
Graus C = 33.25
Graus C = 33.25
Graus C = 33.25
Graus C = 33.00
Graus C = 34.00
Graus C = 34.50
Graus C = 34.50
Graus C = 35.25
Graus C = 36.25
Graus C = 37.75
Graus C = 38.25
Graus C = 38.25
Graus C = 38.75
```

Figura 40: Resultados prueba MAX6675

Conclusiones:

Buen comportamiento, aunque la temperatura que nos ofrece puede mejorarse utilizando un condensador en paralelo al termopar que nos permitirá tener una menor desviación. No se implementará en el circuito final debido a que tenemos la actualización. Personalmente, lo veo más fiable que la actualización. Como ya he mencionado hasta obtener un funcionamiento correcto, tuve que cambiar sondas, y probar sin librería.

3.6.4. SENSOR TERMOPAR ANALÓGICO (AD8495):

Descripción:

Figura 41: Sensor termopar digital AD8495

El sensor AD8495[15] al igual que los anteriores, se trata de un módulo de compensación de unión fría para termopares, en nuestro caso el tipo K, pero esta vez de salida analógica, en vez de mediante interfaz SPI digital [33] [34].

Para calcular la temperatura utiliza la siguiente ecuación:

$$\text{Temperatura} = (\text{Voltaje} - 1.25) / 0.005\ \text{V}.$$

Ecuación 1: Cálculo de la temperatura en AD8495

Características:

- Precisión 5mV/ºC
- Precisión de 1ºC.
- Sensibilidad de 5 mV/ºC.
- Alimentación 2.7 a 18 V, típicamente a 5 V.
- Rango de medición de la unión fría: 0 a 50ºC.
- Rango de temperatura: -250°C a +750 °C. (rango termopar K)

[15] Descargue el *datasheet* en https://cdn-shop.adafruit.com/datasheets/AD8494_8495_8496_8497.pdf

Diagrama de conexiones:

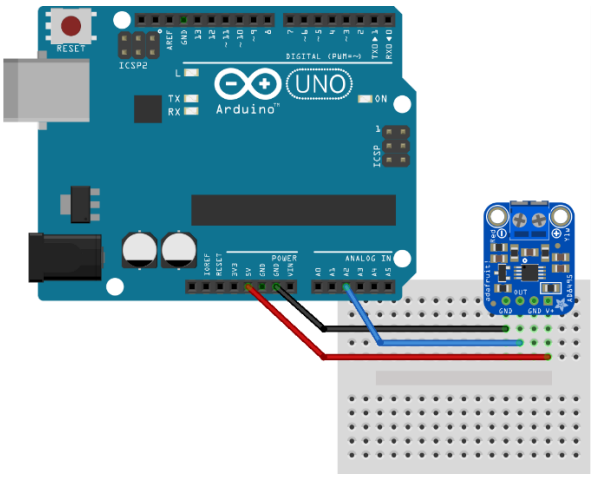


Figura 42. Diagrama de conexiones AD8495

Código de prueba:

```
//Analog read pin
#define TC_PIN A2
#define AREF 5
#define ADC_RESOLUTION 10  // set to ADC bit resolution, 10 is default

float reading, voltage, temperature;

float get_voltage(int raw_adc) {
  return raw_adc * (AREF / (pow(2, ADC_RESOLUTION)-1));
}

float get_temperature(float voltage) {
  return (voltage - 1.25) / 0.005;
}

void setup() {
  Serial.begin(9600);
}

void loop() {
  reading = analogRead(TC_PIN);
  voltage = get_voltage(reading);
  temperature = get_temperature(voltage);
  Serial.print("Temperature = ");
  Serial.print(temperature);
  Serial.println(" C");
  delay(50);
}
```

Resultados:

```
Temperature = 26.64 C
Temperature = 26.64 C
Temperature = 26.64 C
Temperature = 26.64 C
Temperature = 26.64 C
Temperature = 27.61 C
Temperature = 26.64 C
Temperature = 26.64 C
Temperature = 27.61 C
Temperature = 26.64 C
Temperature = 27.61 C
Temperature = 27.61 C
Temperature = 28.59 C
Temperature = 30.55 C
Temperature = 32.50 C
Temperature = 34.46 C
Temperature = 34.46 C
Temperature = 36.41 C
Temperature = 36.41 C
Temperature = 37.39 C
Temperature = 37.39 C
Temperature = 38.37 C
```

Figura 43: Resultados prueba AD8495

Conclusiones:

Dentro de la buena alternativa que presenta el termopar, encontramos otra metodología de medición esta vez de forma analógica, tendiendo un poco más de precisión en la medida. Todo dependerá de la cantidad de dispositivos que queramos utilizar y como vayamos a trabajar con la señal. Este caso funciona bastante mejor que en el caso digital, aunque las sondas no permiten apreciarlo con totalidad. Otro factor a tener en cuenta es que este sensor posee demasiada sensibilidad para tan corta variación. Para que lo entendamos, una pequeña variación o ruido en la señal son unos 8 ºC de diferencia, debido a la suma de errores del termopar y del conversor analógico-digital que posee Arduino.

3.6.5. SENSOR RTD PT100 MAX31865:

Descripción:

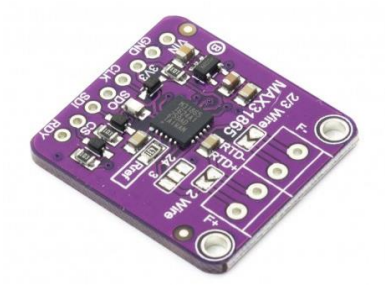


Figura 44: Sensor PT100 digital MAX31865

El sensor MAX31865[16] es muy útil cuando necesitamos trabajar con precisión en la temperatura, ya que utilizaremos una RTD, debido a que es mucho más sensible y estable que otras tecnologías, aunque más cara. La pt100 presenta 100ohms a 0ºC. Pero para obtener esta precisión, necesitaremos de un buen compensador y comparador que detecte pequeños cambios de resistencia [35].

El MAX31865 nos permite trabajar con RTD de 2, 3 y 4 hilos, aunque en nuestro caso trabajaremos a 3 hilos por disponibilidad de manera digital, obteniendo los datos por SPI. Para ello, tendremos que hacer algunas soldaduras:

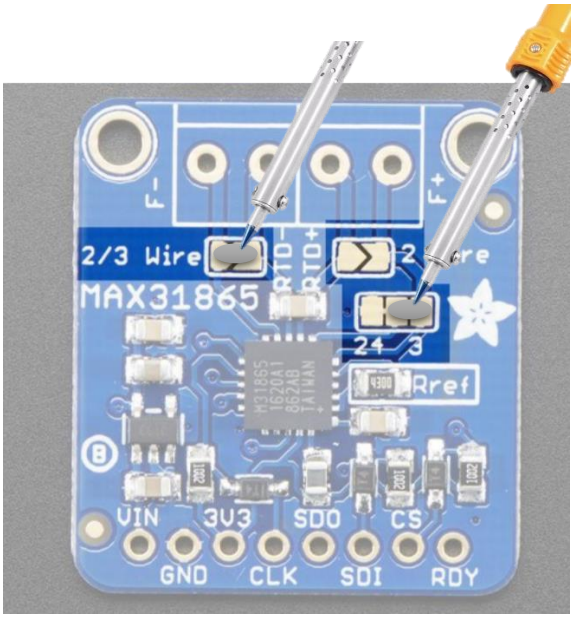


Figura 45: Preajuste PCB para 3 hilos

[16] Descargue el *datasheet* en https://datasheets.maximintegrated.com/en/ds/MAX31865.pdf

Características:

- Alimentación a 5V, aunque lleva un regulador a 3.3V.
- Compatible con interfaz SPI [31].
- 21 ms de tiempo de conversión.
- Resolución de 15-bit (0.03125ºC)
- Precisión de 0.5ºC.

Diagrama de conexiones:

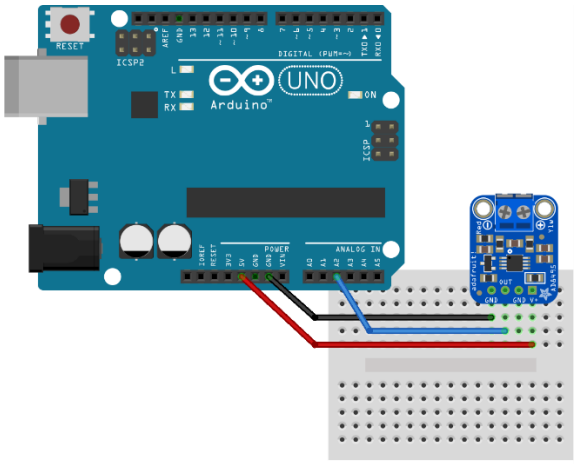


Figura 46: Diagrama de conexiones MAX31865

Código de prueba:

```
//www.scada123.com
#include <Adafruit_MAX31865.h>

// Use software SPI: CS, DI, DO, CLK
Adafruit_MAX31865 max = Adafruit_MAX31865(10, 11, 12, 13);
// use hardware SPI, just pass in the CS pin
//Adafruit_MAX31865 max = Adafruit_MAX31865(10);

// The value of the Rref resistor. Use 430.0 for PT100 and 4300.0 for PT1000
#define RREF      430.0
// The 'nominal' 0-degrees-C resistance of the sensor
// 100.0 for PT100, 1000.0 for PT1000
#define RNOMINAL  100.0

void setup() {
  Serial.begin(9600);
  Serial.println("Adafruit MAX31865 PT100 Sensor Test!");

  max.begin(MAX31865_3WIRE);  // set to 2WIRE or 4WIRE as necessary
}


void loop() {
  uint16_t rtd = max.readRTD();

  Serial.print("RTD value: "); Serial.println(rtd);
  float ratio = rtd;
  ratio /= 32768;
  Serial.print("Ratio = "); Serial.println(ratio,8);
  Serial.print("Resistance = "); Serial.println(RREF*ratio,8);
  Serial.print("Temperature = "); Serial.println(max.temperature(RNOMINAL, RREF));
```

```
  // Check and print any faults
  uint8_t fault = max.readFault();
  if (fault) {
    Serial.print("Fault 0x"); Serial.println(fault, HEX);
    if (fault & MAX31865_FAULT_HIGHTHRESH) {
      Serial.println("RTD High Threshold");
    }
    if (fault & MAX31865_FAULT_LOWTHRESH) {
      Serial.println("RTD Low Threshold");
    }
    if (fault & MAX31865_FAULT_REFINLOW) {
      Serial.println("REFIN- > 0.85 x Bias");
    }
    if (fault & MAX31865_FAULT_REFINHIGH) {
      Serial.println("REFIN- < 0.85 x Bias - FORCE- open");
    }
    if (fault & MAX31865_FAULT_RTDINLOW) {
      Serial.println("RTDIN- < 0.85 x Bias - FORCE- open");
    }
    if (fault & MAX31865_FAULT_OVUV) {
      Serial.println("Under/Over voltage");
    }
    max.clearFault();
  }
  Serial.println();
  delay(1000);
}
```

Resultados:

```
Ratio = 0.00000000
Resistance = 0.00000000
Temperature = 313.13
Fault 0x3

RTD value: 0
Ratio = 0.00000000
Resistance = 0.00000000
Temperature = -242.02

RTD value: 32639
Ratio = 0.99606323
Resistance = 428.30718994
Temperature = -242.02

RTD value: 0
Ratio = 0.00000000
Resistance = 0.00000000
Temperature = 630.16
Fault 0x80
RTD High Threshold
```

Figura 47: Resultados prueba MAX31865

Conclusiones:

Como se puede ver en la simulación, no se tiene información del componente, nos aparecen todos los errores posibles. Tras analizar dicho componente, probar la medición de temperatura sin librería, hacer cambios en la sonda, comprobar soldaduras y medir con el polímetro, se descubre que el integrado esta en cortocircuito y no va a funcionar correctamente. Por tanto, prescindiré de dicho componente.

3.6.6. SENSOR RTD PT100 ANALÓGICO:

Descripción:

Utilizaremos el ajuste analógico utilizando un amplificador con alto CMRR AD625AN[17] para una PT100 de 3 hilos realizada en la asignatura de Instrumentación Electrónica. No se tienen demasiados parámetros acerca de este componente ya que se elaboró desde cero.

Figura 48: Sensor PT100 analógico

Características:

- Rango de medición de 0ºC a 80ºC.
- Rango de salida de 0-4 V.
- Sensibilidad de 1/20.
- Precisión de 0.5ºC.
- Alimentación a 5V o 9V configurable con jumpers.

[17] Descargue el *datasheet* en https://www.analog.com/media/en/technical-documentation/data-sheets/AD625.pdf

Diagrama de conexiones:

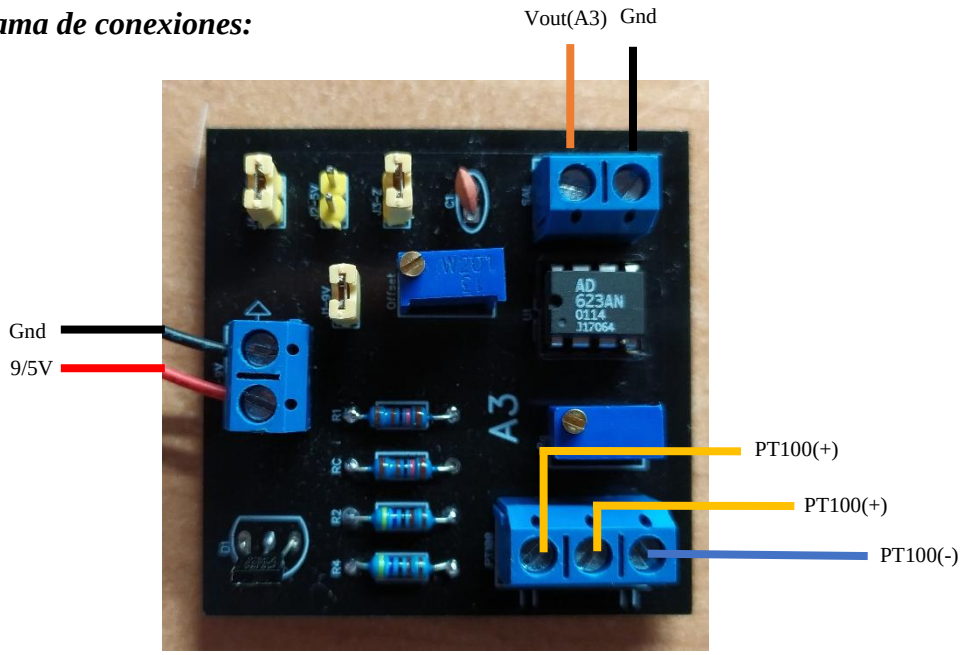


Figura 49: Diagrama de conexiones PT100 analógica

Código de prueba:

```
int pin = 4;
int value;
float Temperatura;
void setup() {
Serial.begin(9600);
Serial.println("Activación PT100");
}

void loop() {
value = analogRead(pin);
Temperatura = 0.097*value;
Serial.print("Temperatura= ");
Serial.print(Temperatura);
Serial.println(" C");
delay(500);
}
```

Resultados:

```
Temperatura= 27.55 C
Temperatura= 27.55 C
Temperatura= 27.45 C
Temperatura= 27.45 C
Temperatura= 27.55 C
Temperatura= 27.45 C
Temperatura= 27.35 C
Temperatura= 27.35 C
Temperatura= 27.45 C
Temperatura= 27.45 C
Temperatura= 27.35 C
Temperatura= 27.35 C
Temperatura= 27.26 C
Temperatura= 27.35 C
Temperatura= 27.45 C
Temperatura= 29.20 C
Temperatura= 30.94 C
Temperatura= 31.62 C
Temperatura= 32.59 C
Temperatura= 34.92 C
Temperatura= 37.83 C
Temperatura= 39.96 C
```

Figura 50: Resultados prueba PT100-A

Conclusiones:

Implementé el circuito ya realizado y calibrado, aunque presentará algún pequeño desvío debido a la no utilización del mismo, aunque es sin duda uno de los más fiables.

Probando con otra PT100 de peor calidad como la de la imagen de abajo, no se obtienen los mismos resultados, bastante más desplazados.

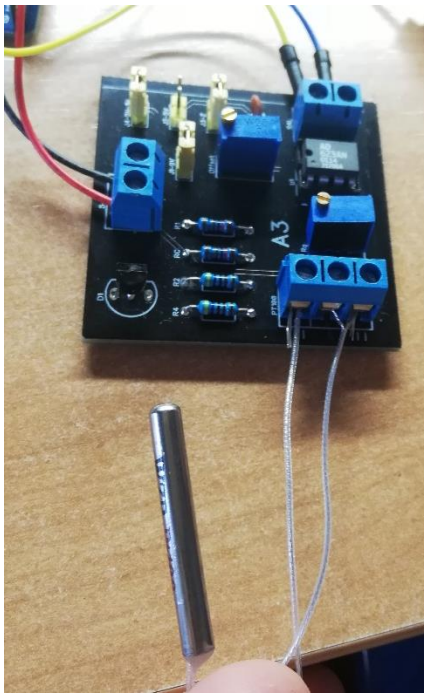


Figura 51: Montaje real PT100 analógica

3.6.7. SENSOR NTC:

Descripción:

Figura 52: Sensor NTC

Un termistor es un dispositivo cuya resistencia varia al variar la temperatura. Lo bueno es que para un pequeño rango podemos asumir casi su linealidad. Esta variación nos permite medir la temperatura ambiente. Existen dos tipos de termistores, PTC, que aumenta con la temperatura, y NTC, que disminuye. Vamos a trabajar con el más usado, NTC, que tienen una resistencia inferior al aumentar su temperatura [36].

El problema de los termistores es que no presentan un comportamiento lineal. Esto se soluciona con el uso de modelos matemáticos que permiten calcular con precisión la temperatura a pesar de que nos obliga a trabajar con números en coma flotante.

Características:

- MF52[18] Rnominal de 10k en nuestro caso.
- Tolerancia 5%.
- Tamaño pequeño, bajo coste y muy duraderas.
- Poco susceptible al ruido.
- Rango de temperatura: -30 a 110 ºC.
- Problema de no linealidad.
- Errores introducidos por procesador, tolerancia de resistencia, etc.

[18] Descargue el *datasheet* en https://www.eaa.net.au/PDF/Hitech/MF52type.pdf

Diagrama de conexiones:

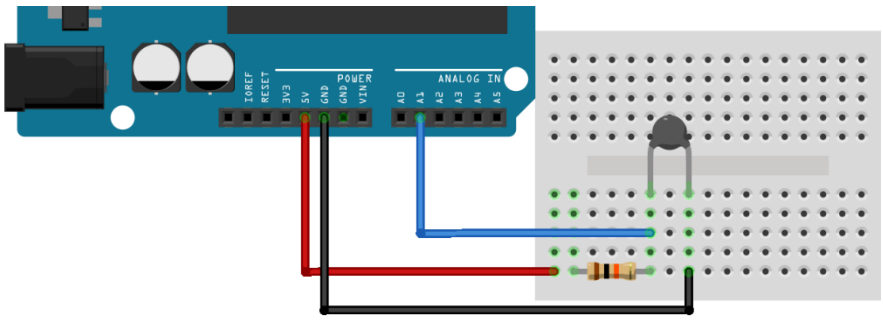


Figura 53: Diagrama de conexiones NTC

Código de prueba:

```
#include <math.h>

const int Rc = 10000; //valor de la resistencia
const int Vcc = 5;
const int SensorPIN = A0;

float A = 1.11492089e-3;
float B = 2.372075385e-4;
float C = 6.954079529e-8;

float K = 2.5; //factor de disipacion en mW/C

void setup()
{
  Serial.begin(9600);
}

void loop()
{
  float raw = analogRead(SensorPIN);
  float V =  raw / 1024 * Vcc;

  float R = (Rc * V ) / (Vcc - V);


  float logR  = log(R);
  float R_th = 1.0 / (A + B * logR + C * logR * logR * logR );

  float kelvin = R_th - V*V/(K * R)*1000;
  float celsius = kelvin - 273.15;

  Serial.print("T = ");
  Serial.print(celsius);
  Serial.print("C\n");
  delay(2500);}
```

Resultados:

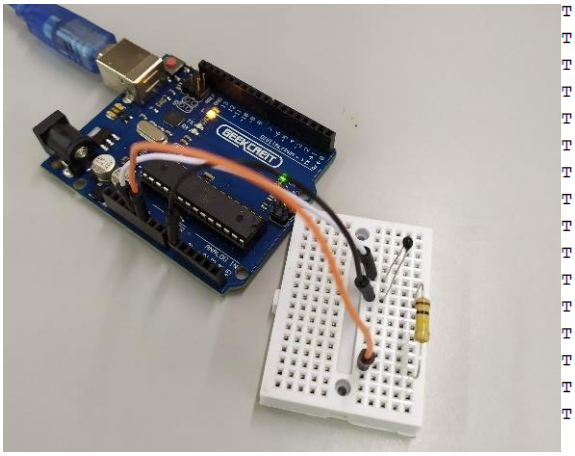


Figura 54: Montaje real NTC

Conclusiones:

Se trata de un buen método de medición de temperatura. Es bastante rápido y fiable, además de muy barato y pequeño, aunque tendremos que asumir el procesamiento debido a la escala no lineal de temperatura, además de las limitaciones de linealidad dentro de la escala conforme vayamos ampliando el rango.

3.6.8. MÓDULO DE ACONDICIONAMIENTO NTC CON VAINA:

Descripción:

Consiste en un PCB previamente creado por la empresa DM. La diferencia con el montado en protoboard es que esta NTC lleva encapsulado para la resistencia al agua, pero el conexionado es el mismo. Además, lleva el divisor de tensión necesario para el acondicionamiento de una NTC.

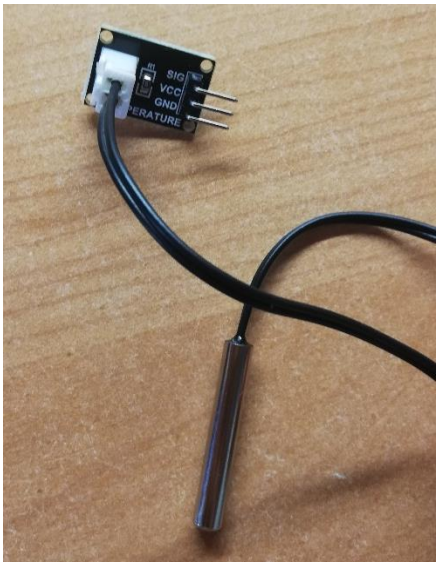


Figura 55: Sensor NTC compacto

Características:

- Voltaje de funcionamiento: 2.2 - 12 VDC
- Corriente máx: 0.5 mA
- Rango de medida: 30 – 120 ºC
- Precisión de medida: ± 2% (4 °C - 50 °C), y ± 3% (-15 °C ~ 80 °C)
- Valor β: 3950 K
- Resistencia NTC adjunta de R_{25}: 1 KΩ, 0.1%

Como no tenemos información exacta acerca de ella, ya que se encuentra descatalogada, lo que haremos será un par de experimentos con el fin de mediante uno de los dos métodos, o Steinhart, que utiliza un polinomio de aproximación lineal, o mediante el cálculo de la beta obtener la temperatura. En mi caso, voy a utilizar el método de la beta:

$$\beta = \frac{\ln(\frac{R_{T1}}{R_{T2}})}{(\frac{1}{T_1} - \frac{1}{T_2})}$$

Ecuación 2: Cálculo de beta

Utilizando el calculador de beta[19] y haciendo un par de experimentos en torno a unos 25ºC, y otra metiendo la sonda en hielo en torno a unos 0ºC, calculamos el valor de beta para poder implementarlo:

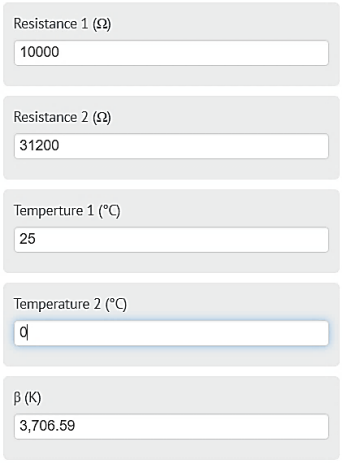


Figura 56: Cálculo de beta a partir de dos puntos

Código de prueba:

```
#include <math.h>
int analogPin1 = 0;
//Datos para las ecuaciones

float Vin = 5.0;
float Rfija = 1000;
float R25 = 10000;
float Beta = 3707.0;
float T0 = 293.15;

float Vout = 0.0;
float Rntc = 0.0;

float TempK = 0.0;
float TempC = 0.0;

void setup() {
  Serial.begin(9600);
}

void loop()
{
  Vout=(Vin/1024)*(analogRead(analogPin1));
  Rntc=(Vout*Rfija)/(Vin-Vout);

  //temperatura en Kelvin
  TempK = Beta/(log(Rntc/R25)+(Beta/T0));
```

[19] Cálculo de la beta https://www.ametherm.com/thermistor/ntc-thermistor-beta

```
  //Y ahora la pasamos a celsius
  TempC = TempK-273.15;

  //Y lo mostramos por puerto serie
  Serial.print("T (ntc): ");
  Serial.print(TempC);
  Serial.println(" ºC");

  delay(500);
}
```

```
T (ntc): 26.72 °C
T (ntc): 26.72 °C
T (ntc): 26.49 °C
T (ntc): 26.26 °C
T (ntc): 27.40 °C
T (ntc): 27.62 °C
T (ntc): 27.40 °C
T (ntc): 27.40 °C
T (ntc): 27.62 °C
T (ntc): 27.85 °C
T (ntc): 28.07 °C
T (ntc): 28.07 °C
T (ntc): 28.07 °C
T (ntc): 28.07 °C
T (ntc): 28.07 °C
T (ntc): 28.29 °C
T (ntc): 28.07 °C
```

Figura 57: Resultados montaje NTC con vaina

Conclusiones:

Como vemos es un buen sensor a utilizar, aunque tenemos que tener en cuenta que la calibración hecha a mano en casa no siempre es la correcta y quizás nos da unas temperaturas un par de grados por debajo de las que realmente hay en la habitación al comenzar el experimento, ya que la hoja de características nos dice que la beta está en torno a los 3950 K.

3.6.9. MÓDULO DE ACONDICIONAMIENTO NTC 2:

Este componente es un termostato propiamente dicho, cuando llega a una cierta temperatura que establecemos con el potenciómetro, salta. En las figuras siguientes se puede ver como al calentar el termistor, al final conseguimos que se encienda.

Figura 58: Sensor NTC on-off

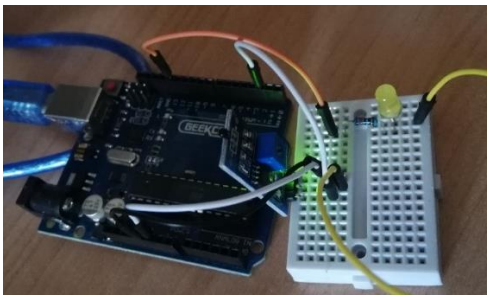


Figura 59: Sensor NTC off

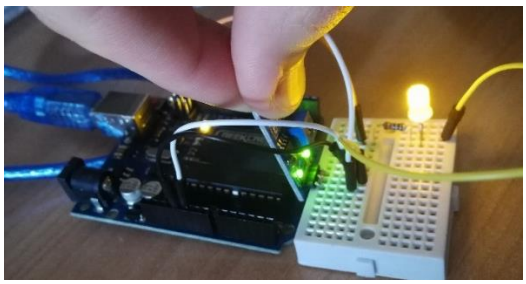


Figura 60: Sensor NTC on

Conclusiones:

Se trata de un sensor todo-nada que actúa a modo de relé. Por su alejada utilidad con respecto al proyecto, no se implementará al final, pero es muy útil para una alarma por temperatura.

3.6.10. SENSOR INTEGRADO DIGITAL DS18B20 ONEWIRE:

Descripción:

Figura 61: Sensor integrado DS18B20

El DS18B20[20] es un sensor de temperaturas que soporta bus de comunicación digital (1-Wire) que podemos leer con las entradas digitales de Arduino [37].

Podemos encontrar al DS18B20 tanto con encapsulado TO-92 (al estilo LM35) como en forma de sonda impermeable, como la que disponemos, preparada para líquidos y gases.

El bus 1-Wire solo necesita un cable para la comunicación (sin contar GND). Se pueden alimentar directamente por la línea de datos, al estilo PoE (Power Over Ethernet), o mediante una línea de tensión. Se puede medir hasta 20 sensores en el mismo bus, e incluso se puede montar hardware para tenerlo a larga distancia [38].

Para ello, OneWire funciona con control de tiempos en la señal, y el tiempo de adquisición total de una medición de 750ms. Requiere de resistencia de pull-up de 4,7K entre Vcc y la señal, para que comunique [39].

Características:

- Tamaño pequeño, bajo coste.
- Tecnología 1-Wire.
- Muy útil en redes con gran número de sensores.
- Rango medida: -55ºC a +125ºC.
- Precisión ±0.5°C.
- Alimentación entre 3.0V y 5.5V.
- Resolución configurable de 9 a 12 bits.

[20] Descargue el *datasheet* en https://datasheets.maximintegrated.com/en/ds/DS18B20.pdf

Diagrama de conexiones:

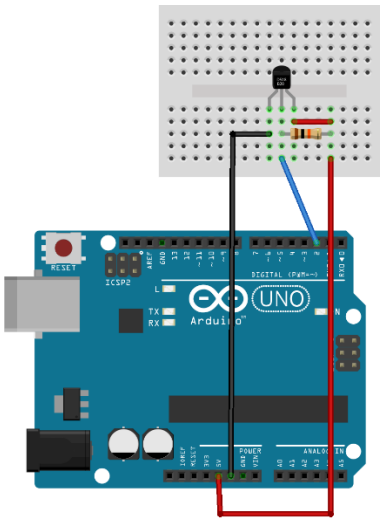


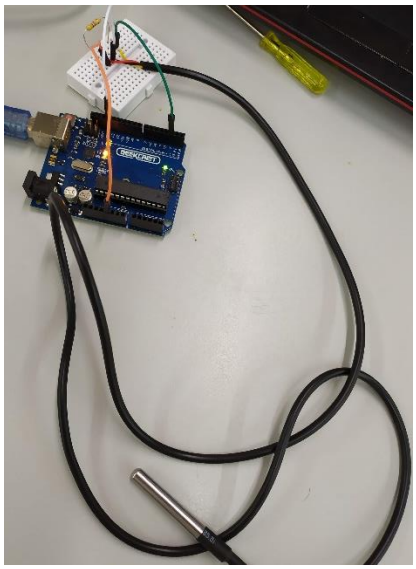
Figura 62: Diagrama de conexiones DS18B20

Código de prueba:

Previa introducción de las librerias OneWire.h y Dallas.h (la del componente, ya que antes pertenecía a Dallas).

```
#include <OneWire.h>
#include <DallasTemperature.h>

OneWire ourWire(2);                //Se establece el pin 2  como bus OneWire

DallasTemperature sensors(&ourWire); //Se declara una variable u objeto para nuestro sensor

void setup() {
delay(1000);
Serial.begin(9600);
sensors.begin();   //Se inicia el sensor
}

void loop() {
sensors.requestTemperatures();   //Se envía el comando para leer la temperatura
float temp= sensors.getTempCByIndex(0); //Se obtiene la temperatura en ºC

Serial.print("Temperatura= ");
Serial.print(temp);
Serial.println(" C");
delay(500);
}
```

Resultados:



```
Temperatura= 29.56 C
Temperatura= 29.50 C
Temperatura= 29.50 C
Temperatura= 29.56 C
Temperatura= 29.81 C
Temperatura= 30.06 C
Temperatura= 30.37 C
Temperatura= 30.62 C
Temperatura= 30.87 C
Temperatura= 31.00 C
Temperatura= 31.25 C
Temperatura= 31.37 C
Temperatura= 31.56 C
Temperatura= 31.69 C
Temperatura= 31.75 C
Temperatura= 31.94 C
Temperatura= 32.00 C
Temperatura= 32.06 C
Temperatura= 32.19 C
Temperatura= 32.25 C
Temperatura= 32.31 C
Temperatura= 32.38 C
```

Figura 63: Montaje real sensor DS18B20

Conclusiones:

Presenta un buen comportamiento, aunque se nota la presencia del encapsulado, ya que tarda bastante tiempo en detectar la temperatura, que nos viene a indicar que está preparado para trabajar con líquidos y gases. Tiene la ventaja del bus OneWire, ya que el cableado se reduce bastante, y nos permite conectar más de uno con mucha facilidad.

3.7. MÓDULO DE CALENTAMIENTO:

Lo primero que haremos será tratar de calcular la intensidad que queremos hacer pasar a través de nuestra resistencia. Ya que tenemos una resistencia de 2.2 ohms y una potencia de 10 W, vamos a alimentarla a 5V, de manera que nos permita trabajar hasta los 2 amperios. Para estar en zona segura, trabajaremos con 1-1.5 amperios, que nos lo proporcionará una fuente de alimentación.

Una vez fijada la fuente de alimentación, veo que Arduino no es capaz de ofrecer tanta potencia ya que podemos quemarlo, por lo que tendremos que utilizar un MOSFET controlado desde el puerto PWM de Arduino, que nos aislará un circuito y otro además de poder permitirnos conmutar correctamente [40].

Como no necesitamos una excesiva rapidez y nuestra intención es poder amplificar intensidad trabajando a 5V, vamos a utilizar la gama de MOSFET preparados para trabajar con niveles lógicos, esto es, 5V. Esta gama es la IRL5…, y consta del IRL510[21], IRL520[22] e IRL540.

Seleccionaremos los dos primeros para ver cuál de ellos se ajusta mejor a nuestros requerimientos, en función de la corriente del drenador que es la que pasa por la carga, utilizando las gráficas de los *datasheet*, que se encuentran en el pie de página.

[21] Descargue el *datasheet* del IRL510 https://www.mouser.es/ProductDetail/Vishay-Siliconix/IRL510PBF?qs=cvaI6Thkwxs7Om4q9N5ahw==

[22] Descargue el *datasheet* del IRL520 https://www.mouser.es/ProductDetail/Infineon-Technologies/IRL520NPBF?qs=%2Fha2pyFaduhMTTyip%252BO%2FGloDB4DIRrGhrhyBXhvAOtz8PXq6KSu3AA%3D%3D

Gráfica para IRL510:

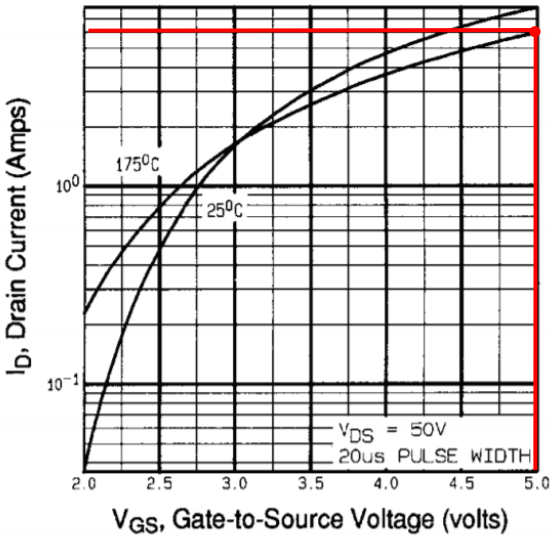


Figura 64: Gráfica Vgs-Id IRL510

Nos ofrece una intensidad en el peor caso de temperatura ambiente de unos 6A, totalmente suficiente para nuestro trabajo. Aun así, vemos la siguiente gama:

Gráfica para IRL520:

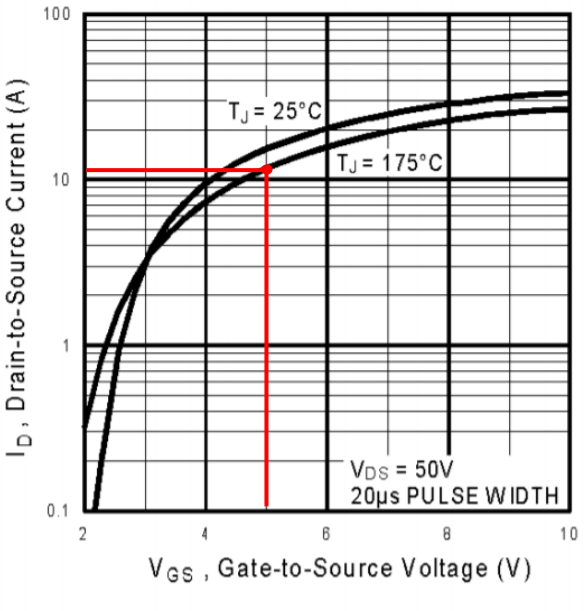


Figura 65: Gráfica Vgs-Id IRL520

Como vemos esta gama nos ofrece casi 12A, superando al anterior en casi el doble. Como ya hemos mencionado no es necesario utilizar un MOSFET más potente, aunque a nuestros efectos da igual cual poner ya que ambos andan al mismo precio en el mercado debido a que este modelo tiene una mayor demanda.

Una vez seleccionado el componente a utilizar, vemos como conectar este componente a nuestro Arduino. Para ello, mostramos el siguiente esquema para un canal-N, que es el que vamos a utilizar, realizado con el entorno *fritzing*[23], aprovechando su gran librería y la facilidad para incorporar nuevos componentes, aunque podía haber utilizado *Tinkercad Circuits*[24]:

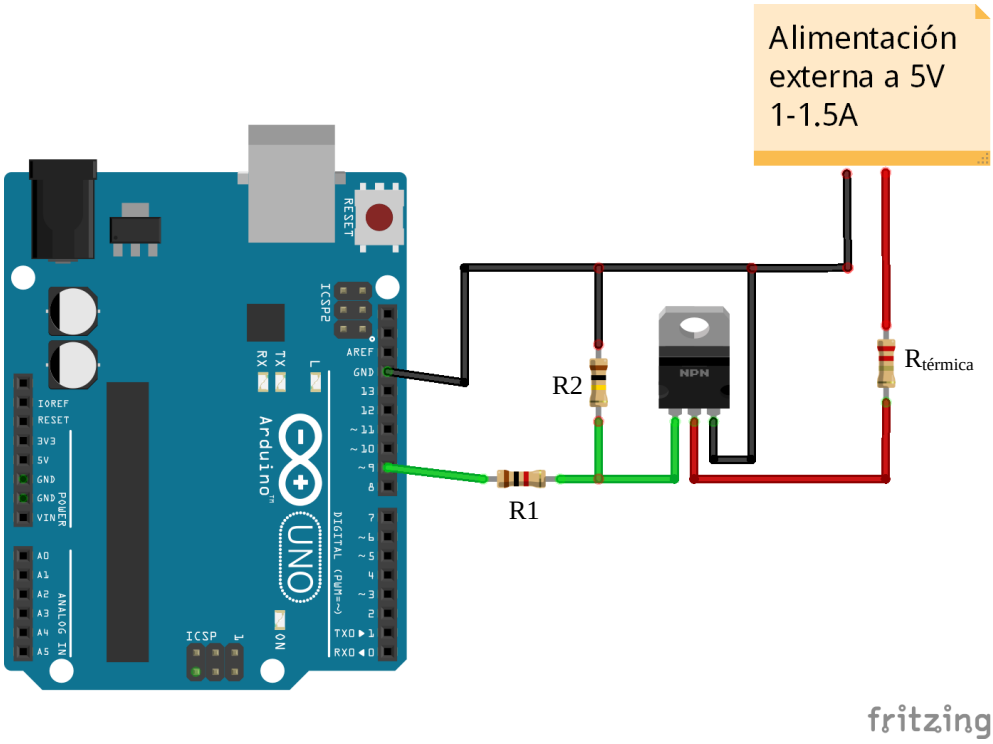


Figura 66: Diagrama de conexión módulo de calentamiento

Todas las resistencias son necesarias. Tenemos la resistencia térmica de la que ya hemos hablado a la derecha. La resistencia R1 nos limita la corriente que nos pide la Gate del transistor y nos evitará el sobreconsumo y aumentará su velocidad. Por otro lado, R2 pone el transistor a masa cuando el pin está en un estado indeterminado, que podría provocar encendidos y apagados del MOSFET.

[23] Consúltelo en https://fritzing.org/home/
[24] Consúltelo en https://www.tinkercad.com/circuits

3.7.1. RESISTENCIA DE CALENTAMIENTO:

Para no limitar nuestro trabajo y poder hacer mejores mediciones, quiero ampliar la posibilidad de variar las fuentes y las resistencias utilizadas para el calentamiento de la zona a estudiar. Debido a que trabajo con una Arduino, y por requerimientos, interesaba trabajar a 5V, aunque quizás es un poco justo, pero para eso tenemos los transistores IRL, ya que quizás otros con una Arduino no conseguimos abrirlos, o únicamente a modo de relé. A continuación, enumero las diferentes fuentes utilizadas para realizar las pruebas. Estas fuentes alimentaban únicamente la resistencia de calentamiento.

- Fuente de 5V 1A máx.
- Fuente de 5V 2 A máx.
- Fuente de 5V 2.5 A máx.

Finalmente, se utiliza la fuente de 2.5 A como fuente definitiva, aunque cabe la posibilidad de en el futuro subir a 9 o 12 V. Cabe destacar que debido a que trabajamos con muy poca resistencia, ya que a 5V si queremos la máxima intensidad necesitaremos poca resistencia por la ley de Ohm, en torno a los 2 ohmios, que es casi un cortocircuito, las fuentes caen un poco y ofrecen en torno a los 3.5 - 4.2 V de salida.

Nos centraremos ahora en la resistencia utilizada para emitir calor. Se han probado dos principales alternativas variando entre ellas la resistividad de éstas:

- Resistencia de hilo bobinado entre los valores 2.2, 4.7 y 1.5 ohmios haciendo el paralelo de ambas.

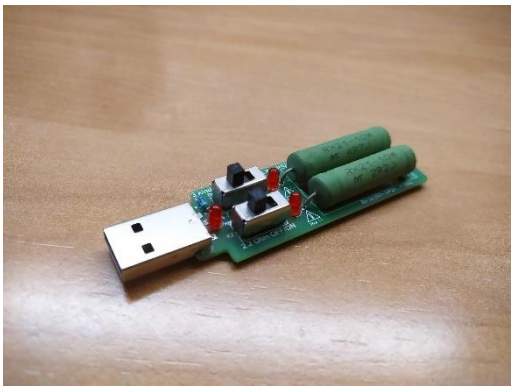


Figura 67: Módulo de resistencias de hilo bobinado

- Resistencia de cerámica entre los valores 0.39 y 1.25 ohmios.

Figura 68: Resistencias de cerámica

3.7.2. PID PARA LAZO DE CONTROL:

Una vez hecho el circuito de alimentación de la resistencia y su actuador, creé el lazo de realimentación en lazo cerrado que nos permitirá saber cómo varía la temperatura de la planta en función de la potencia que le estemos aplicando a la resistencia, y poder actuar para mantener esta temperatura o cambiarla [41].

Para ello, primero haré un experimento de identificación para cada uno de los sensores para poder ajustar los valores del PID. Aunque haya utilizado las tres componentes del PID, realmente la más importante es la componente proporcional ya que si modelizamos una resistencia vemos que se trata de una simple constante y no incluye parte derivativa ni integral, ya que su calor depende de su resistividad, que se define como:

$$R = \rho \frac{L}{A}$$

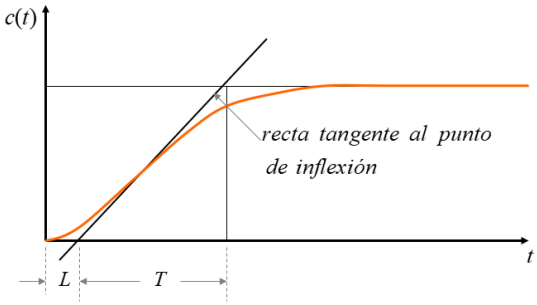
Ecuación 3: Ecuación de la resistividad

en donde en una resistencia normal la densidad ρ, la longitud L, y el área A, son constantes.

Los experimentos han sido realizados en casa, obteniendo resultados similares para ellos empleando el software *Realterm*[25] para la recopilar datos y su posterior exportación a una macro de Excel en donde se separa para cada sensor y se obtiene la

[25] Consulte información en https://sourceforge.net/projects/realterm/

respuesta, en donde a partir del segundo método de Ziegler Nichols [42], y los parámetros L y T de la siguiente figura, obtenemos los parámetros de PID para cada uno de ellos, que podemos consultar en el ANEXO 2: CÁLCULO DE PARÁMETROS DE PID



Tipo de controlador	K_p	τ_i	τ_d
P	$\frac{T}{L}$	∞	0
PI	$0.9\frac{T}{L}$	$\frac{L}{0.3}$	0
PID	$1.2\frac{T}{L}$	$2L$	$0.5L$

Figura 69: Cálculo parámetros PID con el segundo método de Ziegler-Nichols

4. MONTAJE Y PUESTA A PUNTO:

4.1. MONTAJE DE PRUEBAS DE ARDUINO:

Arduino es una plataforma con la que ligeramente había trasteado para proyectos de DIY donde mi mayor meta fue la de controlar una tira LED mediante un módulo bluetooth. Esta vez, tenía que conseguir que el tratamiento de los datos de temperatura previamente realizado se exportase de manera adecuada utilizando el puerto serie de Arduino. Tras el previo análisis de los respectivos sensores y sus librerías y de la creación del PID de control en función de las diferentes entradas, se procede a la integración de las entradas con la temperatura y el control. Descubrí que en este caso la librería de PID que nos ofrece Arduino no es válida en este caso debido a que tengo diferentes sensores y debo de poder elegir cual quiero utilizar para hacer el lazo de control, además de configurar sus parámetros que serán diferentes para cada tipo de sensor, por lo que desarrolle uno que nos permitiera lo mencionado [43]. Las primeras pruebas se realizaron utilizando una fuente de 5V 1 A máx. y una batería portátil que calentaba las resistencias, pero posteriormente se reemplazó debido a que por la baja resistencia que utilizamos para calentar, lo que significa casi un corto, la tensión de alimentación disminuye. El resultado fue el siguiente:

```
T (ds18b20) = 27.75 °C
T (PT100 - A) = 26.09 °C
T (NTC) = 24.07°C
T (ad8495) = -201263.67 °C
T (MAX31855) = 29.50 °C
T (MLX) = 29.63°C


T (ds18b20) = 27.75 °C
T (PT100 - A) = 25.70 °C
T (NTC) = 23.82°C
T (ad8495) = -201263.67 °C
T (MAX31855) = 29.00 °C
T (MLX) = 29.37°C


T (ds18b20) = 27.75 °C
T (PT100 - A) = 26.19 °C
T (NTC) = 24.31°C
T (ad8495) = -201263.67 °C
T (MAX31855) = 30.00 °C
T (MLX) = 29.93°C
```

Figura 70: Salida puerto serie Arduino

Como podemos apreciar los resultados de temperatura de los diferentes sensores son un poco dispares. En el caso del AD8495 o termopar digital, éste es debido a que no se disponía de la sonda para el momento de estas pruebas. Vemos también que la distancia al foco de calor influye, y la calidad y forma de la vaina, también. En el caso de la NTC,

aunque disponga de una beta teórica, al intentar hacer pruebas para obtener una nueva beta obtenemos que esta es ligeramente diferente. Cabe destacar también que el sensor infrarrojo, cuando está enfocado hacia el foco de calor nos arroja una temperatura de la superficie, por lo que es considerablemente mayor a las demás, por lo que tuvimos que cambiar su posición en la maqueta v2 que mostraré más adelante.

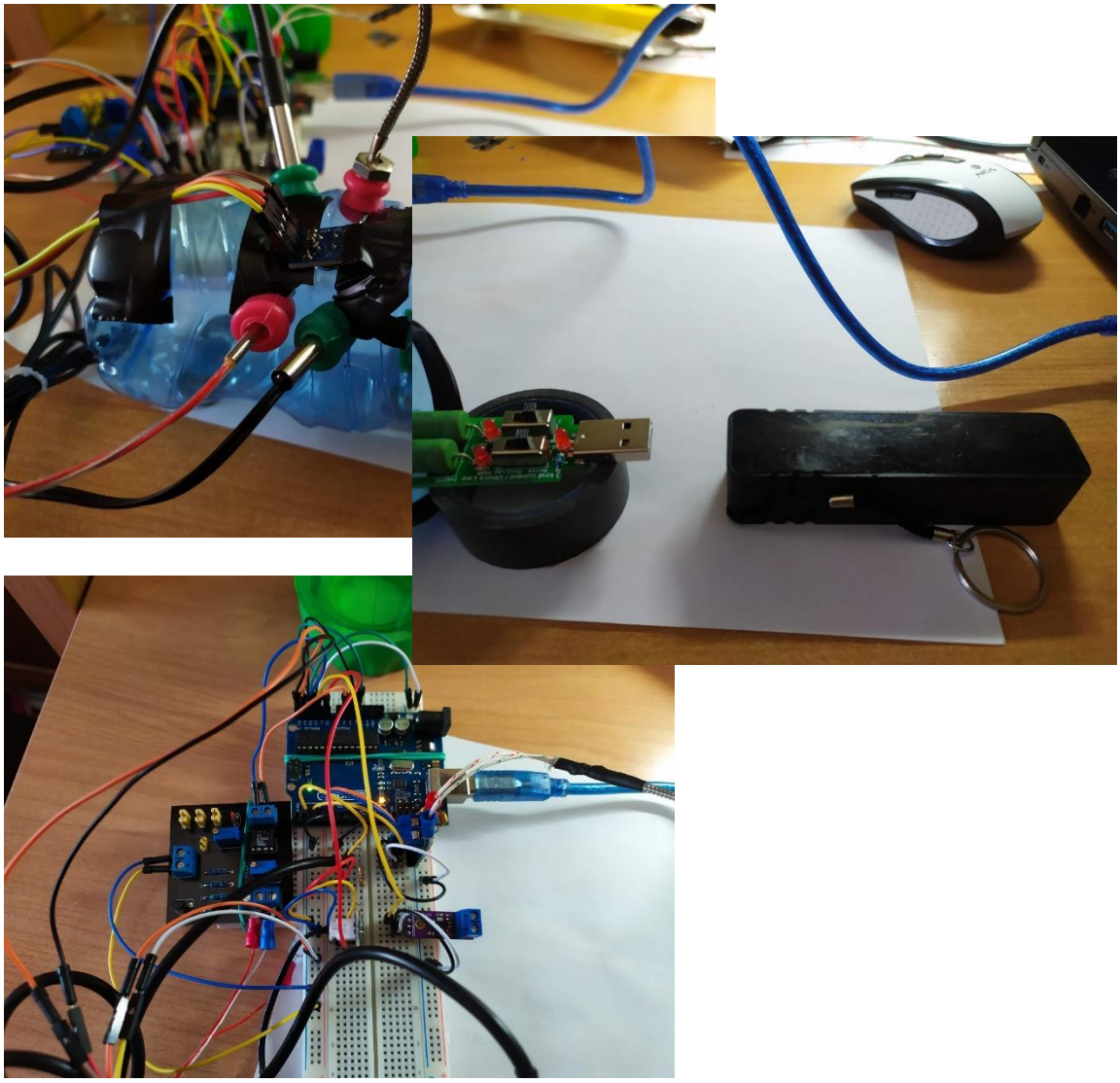


Figura 71: Versión 1 de la maqueta de pruebas

Cabe destacar en esta sección que los sensores de termopar disponían de un ruido que los hacia oscilar unos 5 ºC a temperatura ambiente, pero hemos podido eliminar este ruido en gran medida utilizando para ello condensadores de tántalo para cada uno de ellos, además de realizar un procesado en Arduino que nos va dando la media de 10 muestras de temperatura, que equivaldría a coger el valor central de la sinusoide que estábamos recibiendo. Estos errores tienen varias componentes, desde que el cable de todos los sensores pasa por el mismo lugar a pesar de que estén apantallados, también influye que

el conversor analógico digital de la Arduino no oficial que se ha utilizado no tiene una excesiva precisión ya que al medir la salida con el polímetro obtenemos una salida constante y al convertirla varía unos 0.5 V, lo que equivale a casi 10 ºC en el caso del sensor termopar analógico. No podemos olvidar que estamos ante sensores y transductores de bajo precio, ya que un sensor como los que se usan en la industria puede costar entre 5 y 15 veces más únicamente la sonda.

4.2. MONTAJE A RASPBERRY PI:

Para ello se formatea una tarjeta MicroSD de al menos 16 GB para poder albergar la instalación del IDE de Arduino y los programas de comunicación y las distintas librerías de Firebase. En primer lugar, se procedió a la instalación de Raspbian en una tarjeta SD, previo formateo de la misma y su introducción en la Raspberry.

Una vez inicializada, pasaremos a ponerle una IP fija, para evitar tener que estar buscando la IP asignada por DHCP cada vez que la queramos utilizar, modificando el archivo */etc/dhcpcd.conf y* que nos permitirá poder conectarnos utilizando VNCViewer, para compartir la pantalla [25]. En su uso en casa, se ha trabajado con la interface wlan0, pues se capta internet a través de wifi en la IP 192.168.0.100 que estaba libre.

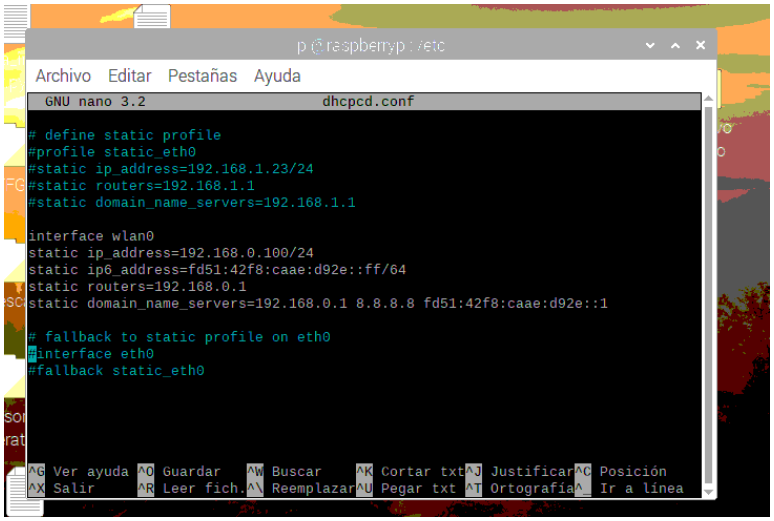


Figura 72: Configuración IP estática en Raspberry

Para la familiarización con la Raspberry se realizan varios proyectos entre los que se incluyen manejo de datos y *string*, hasta acabar en el tratamiento y envío de datos utilizando para ello el puerto serie. También se realizaron diversos programas que nos permitían compartir datos entre la placa y la plataforma Firebase.

Una vez se consiguió una base de conocimientos, se desarrollan los programas que nos permitirán compartir la información mediante Firebase y mediante UDP [17], partiendo de la base de los ejemplos que se incluyen en la librería de LabView hasta conseguir los *scripts* que se utilizarán para su desarrollo y que tendremos que pulir en busca de fallos. Los situaremos en una carpeta en el escritorio.

4.3. ENLACE CON LABVIEW:

Cuando tuve cierto conocimiento de Raspberry comencé a trabajar con LabView, que, si bien ya había trabajado en una asignatura previa, no lo suficiente como para tener cierta autonomía con el programa y su gran variedad de librerías.

Desarrollé diferentes programas de prueba con los que trabajar mediante protocolos TCP/IP y UDP entre la Raspberry y mi ordenador personal tratando de compartir datos entre uno y otro [44]. Posteriormente, trabajé con datos provenientes de Firebase mediante la instalación del SDK de Firebase para Python [45] y la creación y conocimiento de los archivos JSON [46], las reglas de escritura y lectura de la propia base de datos. Como resultado obtuve el sistema que queríamos, pero no a la velocidad que esperaba debido al hardware de la Raspberry, por lo que opté por utilizar UDP.

Aunque conseguimos los requisitos de velocidad de UDP, las funciones de UDP están siempre esperando a que venga una respuesta, y para evitar que se bloquee el código en un único sentido se comenzó a tratar con hilos, uno que permitiese el envío y otro para recibir datos [47]. Además, debido a que los datos de temperatura son los mismos para todos los usuarios se analizaron los métodos de multicast que el protocolo ofrece y del que LabView también dispone de las librerías suficientes [48].

4.4. PRUEBAS DE CONTROL DE TEMPERATURA REMOTA:

Una vez construido el sistema probé el correcto funcionamiento del mismo, y así comentar las características y limitaciones de las que dispone. Procedí al interconexionado de todos los elementos. Para ello, hay que acceder a VNC Viewer con el fin de evitar tener que poner una pantalla y poder controlarlo desde nuestro ordenador personal y posteriormente conectar la Arduino por USB a esta misma.

La maqueta en la que se van a realizar las pruebas se ha actualizado, para permitir el cambio de resistencias de calentamiento a gusto del alumno de manera más fácil, e incluso volver a la anterior, incluyendo además una mayor transparencia que nos facilita conocer el estado del sistema y poder tocar el plástico y experimentar un ligero aumento de la temperatura en el caso de que estemos calentando. Además, hasta ahora no se tenía información de lo que estaba pasando en la maqueta, en especial del tiempo que faltaba para que terminase la simulación. Para ello incorporé un display LCD de 16x2 a la Arduino, además de un módulo con una librería que nos permite poder controlarla mediante el bus I2C, utilizando previamente un *sketch* para conocer su dirección [49]. Este LCD nos informa tanto del estado actual, así como cuándo se han actualizado los PID y del tiempo restante que queda. Como complemento se han añadido también un botón de *reset* de la Arduino, y un LED que también nos informa de cuándo se está calentando, a modo de seguridad.

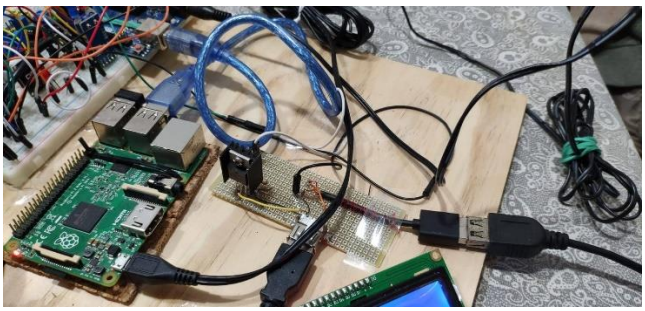


Figura 73: Circuito calentamiento y Raspberry en la maqueta v2

Figura 74: Pantalla de estado en maqueta v2

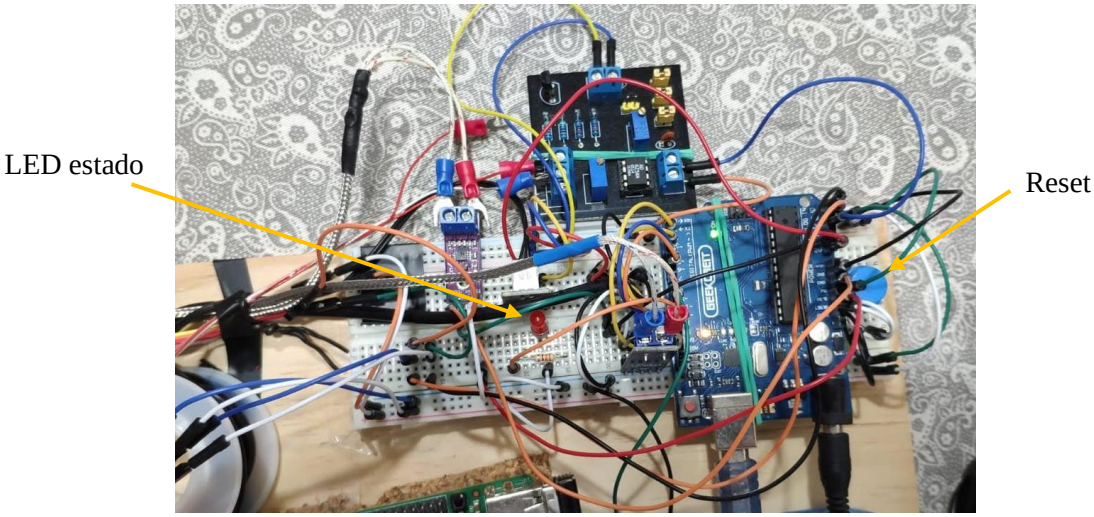


Figura 75: Circuito microcontrolador Arduino en maqueta v2

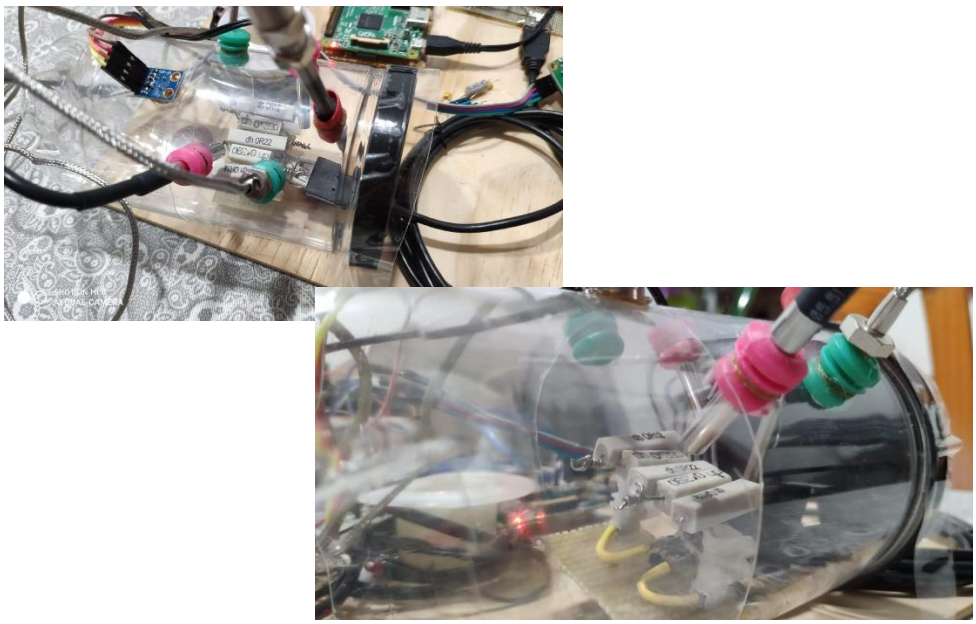


Figura 76: Planta para medición de temperatura en maqueta v2

Para dicho apartado, se van a realizar las siguientes pruebas para no saturar esta memoria:

1. Control PID utilizando Firebase con DS18B20 ante una entrada escalón.
2. Análisis a temperatura ambiente de los diferentes sensores con UDP y actualización de valores de PID.
3. Control PID con sensor MLX90614 haciendo diente de sierra.

1. Control PID utilizando Firebase con DS18B20 ante una entrada escalón:

Se realiza una simulación de unos 400 segundos con una entrada de tipo escalón que queremos que llegue a los 32 ºC. Como ya se ha mencionado anteriormente, debido a la lenta transmisión de datos debido a que la Raspberry tarda bastante tiempo en postear datos en la base de datos de Firebase, realmente no obtenemos un control en tiempo real ya que durante el tiempo no podemos ver el cambio que realmente existe debido a que se satura el puerto serie y tengo que limpiarlo de vez en cuando. Para solucionarlo habría que hacer un *delay* con el posteo de datos, además de tener que hacer unas simulaciones mucho más largas lo que provocaría que no trabajemos en tiempo real. Si realizamos esto mismo conectando la placa Arduino a nuestro pc y ejecutamos el código preparado, se obtiene una mayor transferencia de datos y podemos ver el aumento de temperatura. De hecho, este efecto se evidencia en que si cuando hemos calentado cortamos la simulación y volvemos a encender, obtendremos la experiencia que podemos ver en la figura siguiente *"Capturas de evidencia de diferencias en experimento 1"*, donde se aprecia el aumento de temperatura del que no teníamos constancia debido al *delay* existente, y las diferencias de temperatura tomadas instantes después entre lo que estaba enviando la Raspberry (a la izquierda) y lo que nos ofrecía Arduino (a la derecha). Se descarta el uso de este método si no se realiza una actualización de hardware. Podemos evidenciar que no es problema del servicio web, debido a que desde LabView se alcanza una velocidad de transferencia para el envío y recepción de datos a Firebase que ronda los 15 ms.

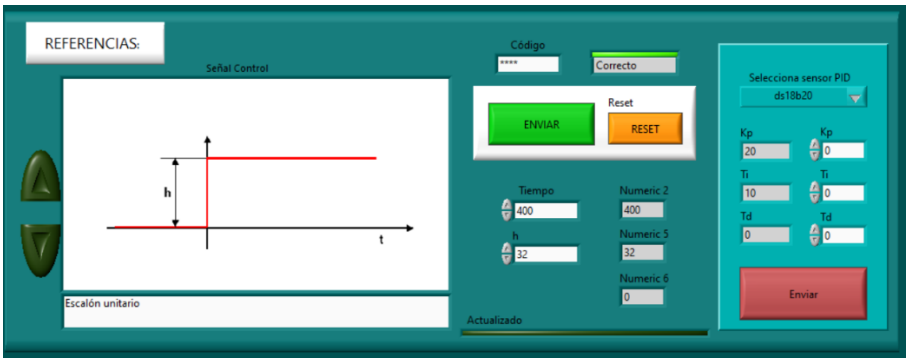


Figura 77: Configuración entrada experimento 1

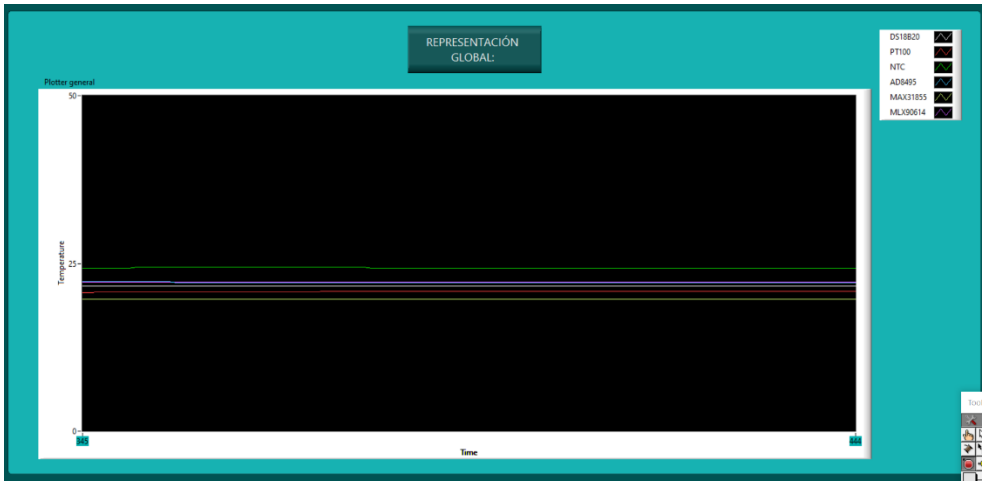


Figura 78: Captura general sensores en experimento 1

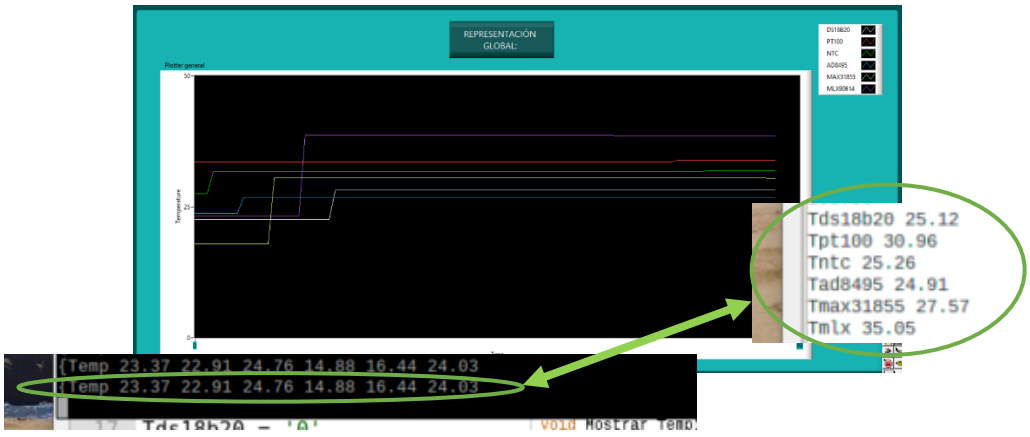


Figura 79: Capturas de evidencia de diferencias en experimento 1

2. Análisis a temperatura ambiente de los diferentes sensores con UDP y actualización de valores de PID:

Se adjuntan capturas del comportamiento de los sensores a temperatura ambiente:

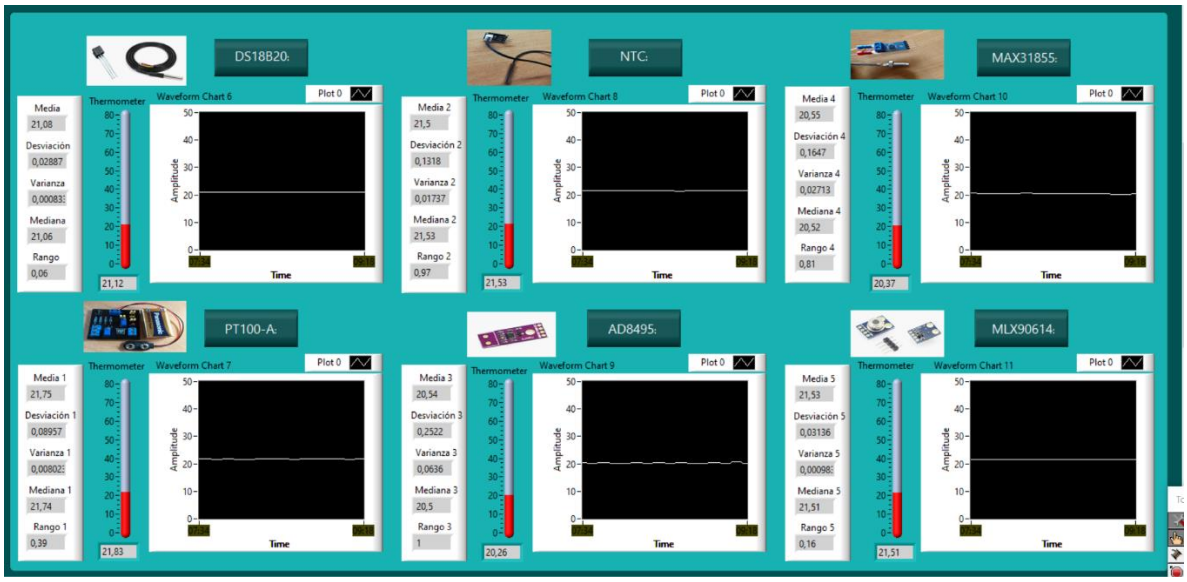


Figura 80: Captura individualizada sensores en experimento 2

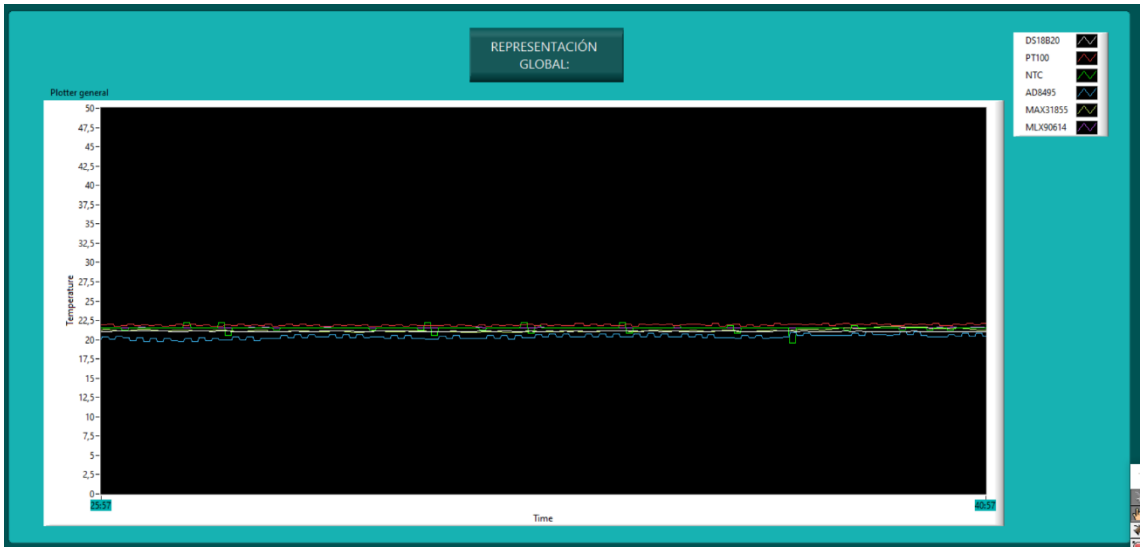


Figura 81: Captura general sensores en el experimento 2

Lo primero a destacar al trabajar con UDP es que de vez en cuando se perdía alguna trama debido a que UDP no tiene confirmación de llegada, algo poco preocupante de primeras, el problema es que a la hora de que LabView nos diera los valores estadísticos, el rango variaba, por lo que se tuvo que incorporar una sección que nos guardara el valor anterior de los sensores para que cuando exista este caso nos ofrezca el valor anterior. Este acontecimiento es poco importante estadísticamente, ya que se trata

de una muestra por minuto, y teniendo en cuenta que tenemos unas 3 muestras por segundo, sería 1/180 muestras erróneas, un 0.5%.

Como podemos apreciar en las capturas, y centrándonos en el rango podemos ver que, en cuanto al rango de la función, los sensores integrado, el infrarrojo y la PT100 son los más fiables ya que nos ofrecen una medida mucho más exacta, cerca del doble o más que los demás, no fiándonos únicamente de los valores de la captura si no en los sucesivos experimentos realizados. El termistor tenía peor exactitud y a veces daba valores distantes. También cabe añadir que los termopares poseen un ruido importante, que tuvo que ser moderado mediante un tratamiento de la media para evitar la senoidal que nos ofrecían, aunque con una de las sondas, pese a implementar un código mediante el cual prescindimos de la librería de componente y la de SPI, no se obtuvo un buen resultado [32].

Este mismo rango nos ayuda también a la hora de estudiar el enfriamiento en el que podemos ver cuál de ellos es el que tiene menos retardo, posiblemente debido a la forma del sensor y la vaina que incluyen que los protege de gases y agua. Además, tal y como podemos ver en la figura siguiente, también tenemos que destacar del sensor MLX90614 (marcado en rosa y en línea más gruesa) que, al ser infrarrojo, no presenta tanto retraso al enfriamiento debido a la vaina.

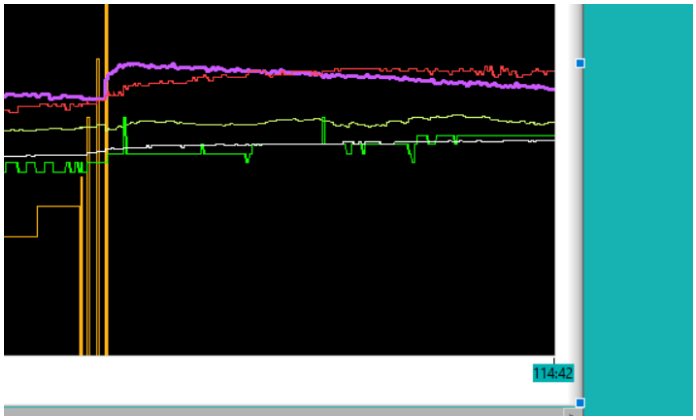


Figura 82: Captura enfriamiento MLX90614

Como experiencia personal, he de añadir que los que mejor resultado me ha ofrecido son el infrarrojo, que además trabaja con bus SPI, lo que lo hace bastante cómodo, pero como ya sabemos tiene el problema de la distancia y limpieza por lo que a veces no es tan exacto como la PT100 o el sensor OneWire. La PT100 es muy

recomendable, además se puede utilizar a dos o tres hilos, aunque quizás junto con su acondicionamiento puede resultar caro. En un lugar sin interferencias electromagnéticas recomiendo muy encarecidamente el sensor OneWire, pese a que cuando aumentamos las distancias tengamos que incorporar hardware como podemos ver en [38] para el tema del bus, y nos permite con un hardware poner más de un sensor en una subestación a una cierta distancia y controlarlo con un solo cable y una Arduino, aunque tenga una lectura mucho más lenta en comparación con el sensor PT100.

Se añade en este experimento y a modo de prueba la actualización del PID del sensor AD8495 en este caso, en el que vamos a cambiar los parámetros para cerciorando que se cambian en la pantalla de Arduino.

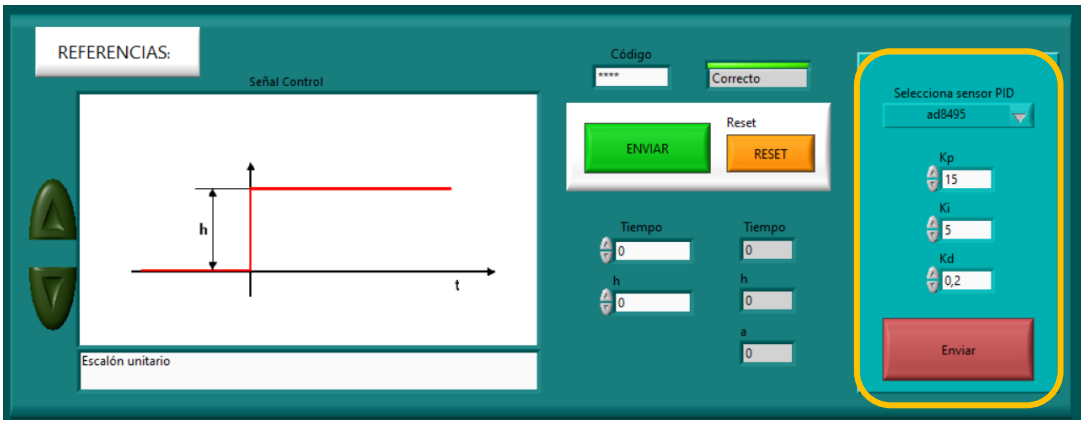


Figura 83: Captura actualización PID en el experimento 2

Pulsamos en enviar y en un segundo más o menos vemos el siguiente mensaje en la pantalla:

Figura 84: Imagen actualización PID en pantalla en experimento 2

3. Control PID con sensor MLX90614 haciendo diente de sierra:

Ahora voy a realizar una simulación mucho más larga que las de antes. Para ello configuraremos la entrada. Olvide cambiar el sensor del PID antes de tomar la captura, pero estamos realizándola con el infrarrojo. Configuraremos una entrada de tipo sierra. Tuve ciertos problemas al utilizar punteros en Arduino para el *Setpoint* y tuve que reemplazarlo, pasándolo por funciones, que, aunque queda menos estético, no me daba tantos problemas.

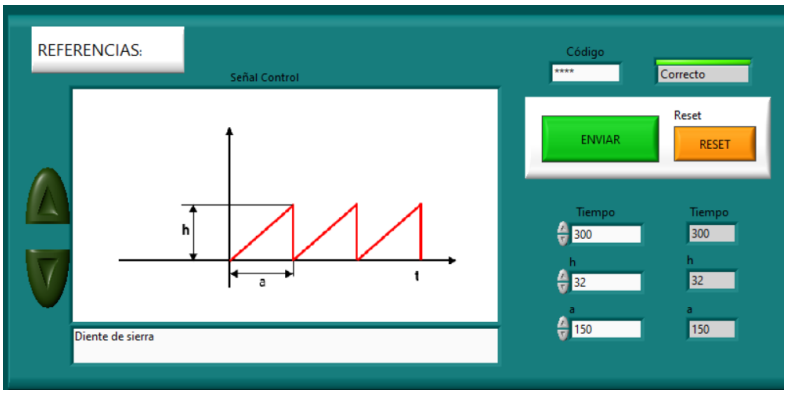


Figura 85: Captura panel de referencias experimento 3

Los resultados obtenidos son los siguientes:

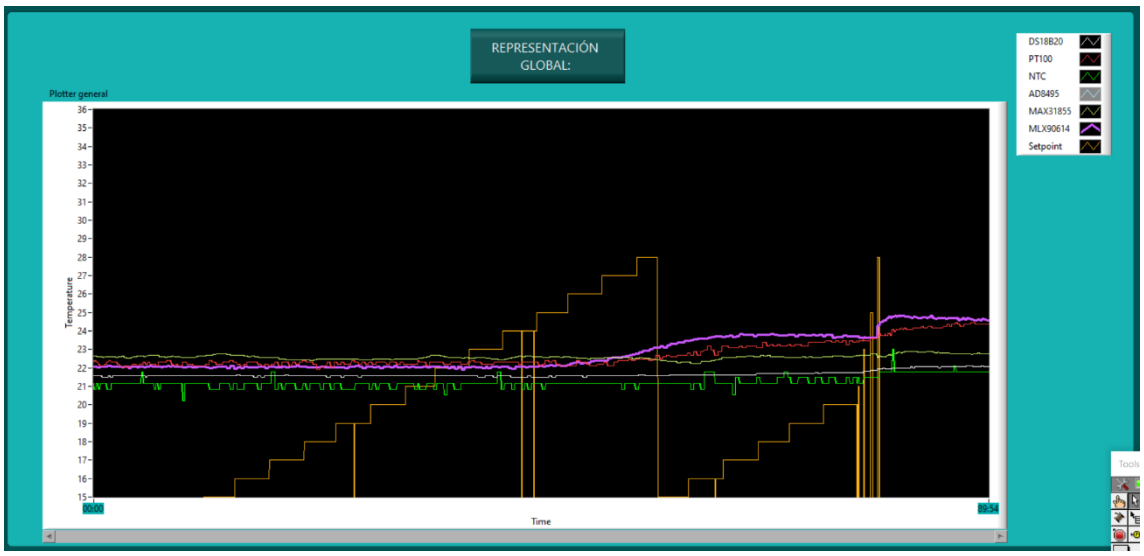


Figura 86: Captura experimento 3

Como se puede apreciar, tuve que eliminar uno de los sensores, no porque no funcionase, sino porque uno de los termopares me fallaba mucho, y decidí quitar ese en vez del MAX31855. Como se puede ver, el MLX es el sensor que más cerca de la referencia está, aunque tiene bastante retardo al igual que todos los sensores, pero

podemos definirlo como el que mejor respuesta tiene ante este tipo de entrada en comparación con los demás, habiendo incluso alguno como el MAX31855 que casi que no denota la variación, aunque también es el que más lejos está de la fuente de calor. Aquí podemos visualizar que, si queremos un impacto en temperatura mucho mayor, tendremos que utilizar para ello una mayor tensión y resistencias mucho más potentes que nos permitan un calentamiento mucho más rápido pese a tener un consumo bastante más notable.

Quise en este momento aumentar la memoria de LabView para poder tener una visión más general del sistema, y fue en este momento en el que aprecié que, al superar un cierto tiempo, se ralentiza el ciclo de LabView de manera importante, provocando en algunos casos perdidas de información bastante notables como las que vemos en el segundo diente de sierra. De hecho, LabView es incapaz de conseguir la tasa de adquisición de datos que nos ofrece la Raspberry, provocando que veamos lo que es una rampa como la suma de muchos escalones.

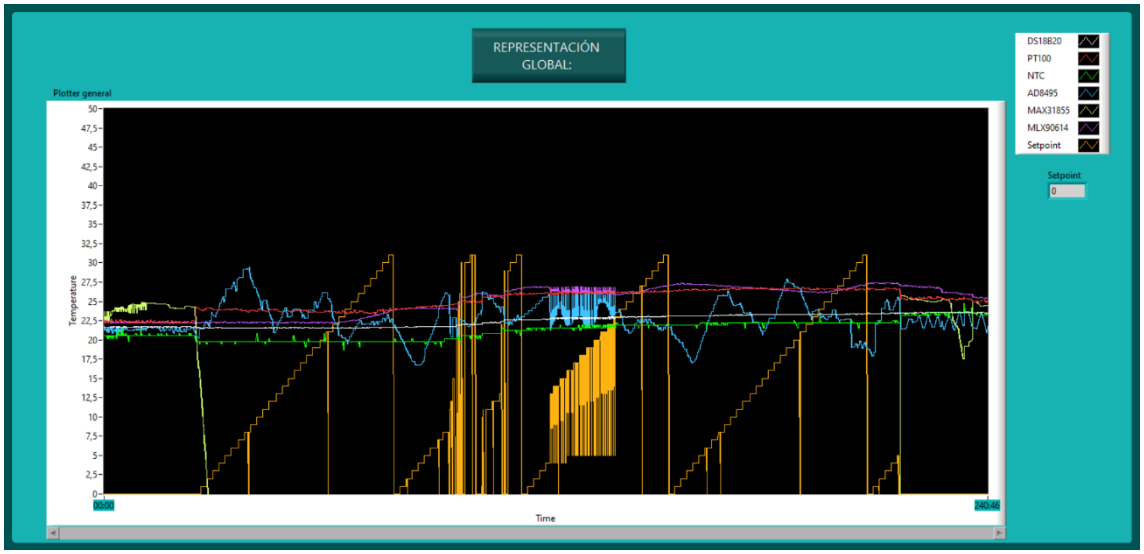


Figura 87: Captura análisis de tiempo LabView

Cabe mencionar también, que tal y como vemos en la figura, los tiempos de adquisición de datos varían a lo largo de la simulación como vemos, por eso que vemos esos triángulos deformados.

En azul podemos ver la sonda que como indico no funciona correctamente y nos deja gráficas como la que vemos en la figura anterior, por lo que se suprimió para la simulación.

5. PLANIFICACIÓN:

Se realiza un desglose de las tareas necesarias para la consecución de los objetivos, estableciendo los periodos de trabajo en semanas estipulando 5h/diarias:

TAREA 1	
Nombre	Creación de la introducción
Inicio	0
Fin	0,5
Entrada	
Salida	Bases propuesta TFG
Descripción	Se establecerán las bases para nuestro proyecto
Objetivo	Asentar las bases del proyecto a realizar

TAREA 2	
Nombre	Análisis de arquitectura global
Inicio	0,5
Fin	1
Entrada	Introducción
Salida	Esquema arquitectura global del sistema
Descripción	Se analizará y se creará una estructura gráfica del proyecto
Objetivo	Disponer de una descripción gráfica general del proyecto

TAREA 3	
Nombre	Análisis de tareas y cronograma
Inicio	1
Fin	2
Entrada	Introducción y arquitectura global
Salida	Análisis de tiempos
Descripción	Se realizará una organización temporal de las tareas y su descripción
Objetivo	Disponer de una estructura temporal de proyecto

TAREA 4	
Nombre	Selección, compra y llegada de componentes
Inicio	2
Fin	5
Entrada	Introducción y arquitectura global
Salida	Orden compra
Descripción	Búsqueda y orden de componentes principales necesarios
Objetivo	Disponer de los materiales necesarios con la mayor brevedad posible

TAREA 5	
Nombre	Sistema de calefacción del sistema
Inicio	3
Fin	4
Entrada	Arduino y MOSFET
Salida	Control por modulación temperatura
Descripción	Identificación de nuestra resistencia calefactora utilizando un MOSFET controlado por PWM
Objetivo	Control de energía en nuestra resistencia calefactora

TAREA 6	
Nombre	Creación y ajuste PID Arduino
Inicio	4
Fin	5
Entrada	Arduino, MOSFET y termómetro
Salida	Control de T en función de los pulsos
Descripción	Secuencias de activación de la resistencia y detección de la temperatura
Objetivo	Controlar la temperatura de nuestra resistencia con Arduino

TAREA 7	
Nombre	Acondicionamiento físico LM35, NTC e Infrarrojo
Inicio	5
Fin	6
Entrada	Arduino
Salida	Parámetros físico necesarios
Descripción	Obtención de señal física por parte de los sensores
Objetivo	Obtener el parámetro eléctrico dado por los sensores indicados

TAREA 8	
Nombre	Acondicionamiento físico RTD y termopar
Inicio	6
Fin	7
Entrada	Arduino
Salida	Parámetros físico necesarios
Descripción	Obtención de señal física por parte de los sensores
Objetivo	Obtener el parámetro eléctrico dado por los sensores indicados

TAREA 9	
Nombre	Programación LM35, NTC e Infrarrojo
Inicio	7
Fin	8
Entrada	Arduino
Salida	Funciones acondicionamiento temperatura
Descripción	Calculo de T en función del parámetros recibido por los componentes
Objetivo	Obtener la temperatura en función del parámetro dado por los sensores indicados

TAREA 10	
Nombre	Programación RTD y termopar
Inicio	8
Fin	9
Entrada	Arduino
Salida	Funciones acondicionamiento temperatura
Descripción	Cálculo de T en función de los parámetros recibidos por los componentes
Objetivo	Obtener la temperatura en función del parámetro dado por los sensores indicados

TAREA 11	
Nombre	Programación escalones
Inicio	9
Fin	10
Entrada	Arduino, Osciloscopio
Salida	Diferentes escalones
Descripción	Programación de diferentes entradas escalón
Objetivo	Modificación de la entrada de calefacción para diferentes variaciones de temperatura

TAREA 12	
Nombre	Conexión Arduino-Raspberry
Inicio	10
Fin	10,5
Entrada	Arduino, Raspberry Pi
Salida	Intercambio de datos entre ambas tarjetas
Descripción	Programación de secuencias para la conexión entre ambas tarjetas
Objetivo	Intercambio de datos entre ambas tarjetas utilizando el puerto serie

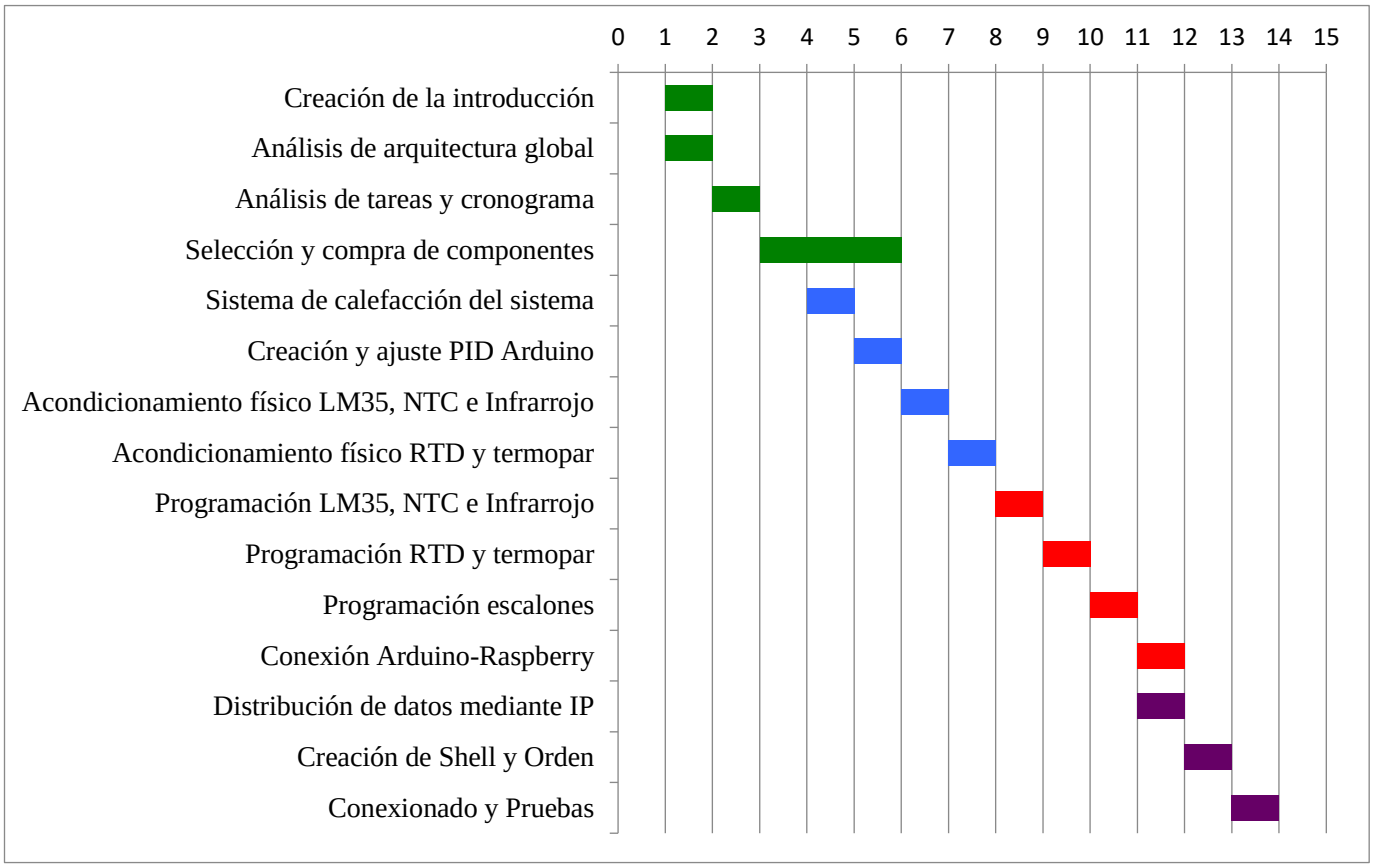
TAREA 13	
Nombre	Distribución de datos mediante IP
Inicio	10,5
Fin	11
Entrada	Arduino, Raspberry Pi
Salida	Conexión a Arduino por red local
Descripción	Configuración de tarjetas para su control o visualización de datos mediante conexión IP
Objetivo	Visualización de datos de arduino, accediendo a la Raspberry Pi mediante conexión IP

TAREA 14	
Nombre	Creación de Shell y Orden
Inicio	11
Fin	12
Entrada	Arduino, Raspberry Pi, componentes
Salida	Integración como un único sistema
Descripción	Creación de una maqueta en donde se haga la inclusión del trabajo realizado
Objetivo	Integración de los diferentes sistemas en uno

TAREA 15	
Nombre	Conexionado y Pruebas
Inicio	12
Fin	13
Entrada	Todas las tareas anteriores
Salida	Sistema final
Descripción	Conexionado y pruebas de funcionamiento
Objetivo	Comprobar que el sistema funciona conforme a lo establecido

5.1.CRONOGRAMA:

Se realiza un desglose de las tareas mencionadas anteriormente utilizando un diagrama de Gantt, utilizando un desglose semanal:



6. PRESUPUESTO:

Se realiza un desglose en forma de tabla del presupuesto del material empleado para la realización de dicho trabajo, incluyendo el precio por unidad y total, proveedor de dicho material y precio final. Los precios son orientativos, y dependen de la calidad y del lugar de compra.

Cantidad	Artículo	Descripción	Precio (€)	Precio total (€)	Proveedor
1	**Raspberry pi 2**	Placa SBC Raspberry Pi Model 2 B con procesador Broadcom BCM2836 ARM Cortex A7 1 GB de RAM	37,44 €	37,44 €	UPCT
1	**Arduino uno**	Arduino UNO microcontrolador ATMega 328 Entradas/Salidas digitales : 14 (6 PWM) Entradas Analógicas: 6	19,90 €	19,90 €	Aliexpress
1	**Acondicionamiento termopar tipo K analógico**	Compensación de unión fría analógico AD8495 para termopar tipo K Rango: -250°C to +750°C Sensibilidad: 5mV/°C	3,14 €	3,14 €	Aliexpress
1	**Acondicionamiento termopar tipo K digital**	Compensación de unión fría termopar tipo K MAX31855 Rango: -200C To +1350°C bus SPI Resolución: 0.25°C	5,15 €	5,15 €	Aliexpress
2	**Termopar tipo K**	Sensor de temperatura termopar tipo K con cable trenzado de acero inoxidable y rango -100 a 1250°C	3,38 €	6,76 €	Aliexpress
1	**Acondicionamiento pt100 analógico**	Acondicionamiento de PT100 para 2/3 hilos con salida 0-4V y FS de 80 °C	12,54 €	12,54 €	Placa diseñada y calibrada en IE
1	**Pt100 de tres hilos**	Sonda de temperatura PT100 de 3 hilos	8,17 €	8,17 €	Miki.Pro
1	**Sensor DS18B20 OneWire**	Sensor de temperatura digital integrado OneWire	0,84 €	0,84 €	Aliexpress
1	**Acondicionamiento analógico NTC**	Breakout board para termistor NTC mediante divisor de tensión para su compensación	2,27 €	2,27 €	DM DIY and More
1	**NTC**	Sensor de temperatura con termistor NTC 10K 1% resistente al agua Rango: -20 to 105 °C	0,54 €	0,54 €	DM DIY and More
1	**Sensor infrarrojo MLX90614**	Sensor de temperatura infrarrojo con comunicación por I2C Precisión: 0.5ºC Rango trabajo: -40 + 125 °C	3,94 €	3,94 €	Aliexpress
1	**Modem WiFi USB**	Mini Adaptador USB Inalámbrico N de 300Mbps	9,99 €	9,99 €	UPCT
4	**Resistencias calefactora 10W**	Resistencia bobinada de 2.2 ohmios y 10W o superior	0,93 €	3,72 €	UPCT y Rayte
1	**Circuito de calentamiento**	Transistor de efecto campo IRL530, disipador y adaptadores	2,70 €	2,70 €	Ebay
1	**Módulo I2C + LCD 16x2**	Módulo LCD 16x2 junto con adaptador a bus I2C para Arduino	3,95 €	3,95 €	Aliexpress
1	**Cableado**	Placas de montaje, protoboard y cableado vario	5 €	5,00 €	Aliexpress
Total				**126,05 €**	

Tabla 4: Tabla de presupuestos

7. CONCLUSIONES Y TRABAJOS FUTUROS:

El principal objetivo de este trabajo era la caracterización de diferentes sensores de temperatura que tengan un bajo precio de manera que esté al alcance de cualquier estudiante, y el posterior tratado de datos con una interfaz tan universal y cada vez más utilizada como es Arduino, además de permitir el control de dicha temperatura mediante la creación de un módulo de calentamiento y un sistema de control para poder manipular adecuadamente dicha temperatura. Para ello, se ha perseguido buscar diferentes módulos ya existentes para dicha plataforma con la intención de que sea fácilmente modificable y adaptable, además de poder invertir más tiempo en la magnitud física de la temperatura y no tanto en la compensación que tenemos que realizar para llegar a ella, y se ha creado una pequeña placa con la que se gestiona la temperatura aprovechando las salidas PWM de Arduino, obteniendo un código sencillo de tratamiento de datos fácilmente adaptable a más sensores e incluso a otras plataformas en el que se han aplicado técnicas de tratamiento de datos, manejo de tiempos, control PID y envío de datos por el puerto serie.

Otro principal objetivo de este trabajo era el tratamiento de estos datos en red o a distancia, para permitir la monitorización de datos en caso de no estar presente o en caso de realizar dicha práctica fuera de la situación actual del experimento que se ha abordado implementando en una Raspberry que trabaja bajo Raspbian, un *bridge* de datos entre el puerto serie y una base de datos que nos permitiera acceder a esos datos. Para evitar el desarrollo de una base de datos convencional, con el almacenamiento, puesta en marcha y tiempo de acceso que conlleva, se implementa en una base de datos en tiempo real como es *Firebase* con la que podemos conseguir tiempos de tan solo unos milisegundos. Para solventar la demora que existe debida al hardware que se dispone, que nos aporta unos tiempos en torno al segundo, se propone la utilización del protocolo UDP frente a TCP, ya que la perdida de datos no es realmente importante para hacer el envío de dicha información, con el que se realizará multicast de datos de temperatura a los clientes, y la transferencia de información de control mediante un socket en UDP en un puerto diferente al anterior, bajo código Python utilizando un hilo para el envío y otro para recibir datos.

Un buen sistema de control además incluye un entorno de control y de monitorización que se ha llevado a cabo para las dos plataformas de envío de aspecto muy similar pero un poco diferentes por dentro. Se ha conseguido monitorizar la temperatura de los 6 sensores en tiempo real, y es posible el control del sistema ante diferentes entradas programables desde dicho panel en LabView, en el que para poder enviar este control

deberé de conocer la contraseña, además se ha incluido la posibilidad de cambiar los parámetros de PID para cada uno de los sensores estudiados.

Para finalizar esta parte, cabe añadir que el principal objetivo ha sido la enseñanza con el lema *"Learn by doing"*, pudiendo haber aprendido control de tiempos y librerías de Arduino, lenguaje Python, en especial a trabajar con datos, con el puerto serie y con sockets de conexión con UDP/TCP, aprender a trabajar con una Raspberry, y las diferentes librerías dentro de LabView hasta conseguir la función requerida.

Este trabajo se queda abierto a muchas modificaciones, como movimiento de sensores para ver qué posición es la mejor, variación de fuentes, cambio de sensores, variación en las resistencias y por supuesto, la temperatura no es la única variable importante.

Tras finalizar dicho trabajo se quedan muchos horizontes abiertos siguiendo en el terreno del internet de las cosas. Para empezar, hemos mezclado dos lenguajes fáciles y muy arraigados como son Python y Arduino (proviene del lenguaje C) por lo que la variedad de modalidades será muy amplia.

Como ya se ha mencionado durante el trascurso del trabajo la temperatura no es la única variable importante, hay otras también muy significativas como lo son la presión el nivel y poder controlar todas ellas nos permite llegar a un dispositivo totalmente conectado.

En la industria no se utiliza Arduino como microcontrolador, si no que existen objetos con mayor funcionalidad como lo son los PLC (*Programming Logic Controller*), que nos permiten trabajar con datos de entrada salida a potencias mucho mayores. Se propone reemplazar dicha Arduino por el S7-1200 que disponemos en los laboratorios y utilizar sensores reales industriales e incorporar LabView para el control, a modo de centro de control como un DeltaV.

También considero notable mencionar la posibilidad de crear una aplicación ejecutable desde cualquier móvil Android que nos permita estar al tanto de las temperaturas y de notificarnos en caso de que exista algún error notorio utilizando para ello la utilidad que nos ofrece MIT App Inventor, con la que podremos crear una aplicación sin la necesidad de utilizar código directamente.

Cabe mencionar que debido al desconocimiento de su uso final, se planteó la creación de un PCB que se dispusiese encima del Arduino y que fuese el punto de conexión de todos los módulos de medida, y que se acoplase a modo de *protoshield* como la que vemos en la siguiente figura a la Arduino y que podemos ver su creación pero no ajustada de medidas ya que se desconoce si se incorporará algún componente más o si se reemplazará dicho Arduino por otro modelo.

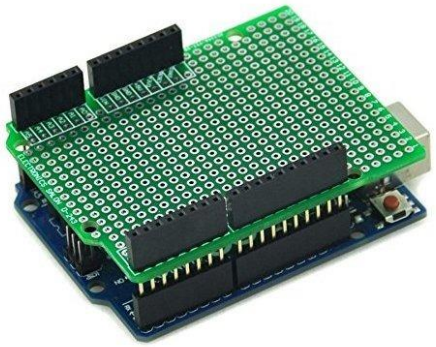


Figura 88: Protoshield Arduino UNO

Y, por último, aunque como en el anterior caso no se llevó a cabo la creación de una caja en donde introducir ambas placas con los respectivos módulos, de manera que se pueda ver y no manipular el cableado, la introducción del LCD, y un ambiente conocido similar al que tenemos en la maqueta con 6 o más agujeros en donde podamos introducir las vainas de los sensores y realizar los distintos experimentos. Podemos consultarlo en ANEXO 4: MONTAJE EN PCB: y ANEXO 5: DESARROLLO DE MAQUETA DE PROYECTO:.

8. ANEXOS:

8.1. ANEXO 1: LIBRERÍAS DE SENSORES DE ARDUINO

Además de las librerías ya incluidas en el IDE de Arduino que necesitaremos como <math.h> , <SPI.h> y <millis()>, con las que utilizaremos funciones matemáticas, protocolo de bus digital o control de tiempo sin parar la simulación, respectivamente, y de las que podemos encontrar información en la documentación de la página oficial de Arduino, utilizaremos otras librerías externas:

8.1.1. Librería OneWire:

Puesto que disponemos de un sensor con esta tecnología que simplifica el trabajo y cableado, utilizaremos dicha librería. Podemos encontrar más información de esta librería en la siguiente página:

https://www.pjrc.com/teensy/td_libs_OneWire.html

8.1.2. Librería DS18B20:

Esta librería nos permite trabajar con dicho componente, incluso poner varios en serie. Podemos encontrar más información aquí:

https://github.com/milesburton/Arduino-Temperature-Control-Library

8.1.3. Librería Wire:

Esta librería la utilizaremos para poder trabajar con el bus I2C, que nos vendrá bien para trabajar con el MLX90614. Podemos encontrar más información en la siguiente página:

https://www.pjrc.com/teensy/td_libs_Wire.html

8.1.4. Librería MAX31855:

Esta librería nos ofrece directamente el valor de la temperatura tanto en Celsius como en Fahrenheit tanto de la unión fría como de la unión caliente del termopar. Podemos ver información sobre ella en la siguiente página:

https://github.com/adafruit/Adafruit-MAX31855-library

8.1.5. Librería MLX90614:

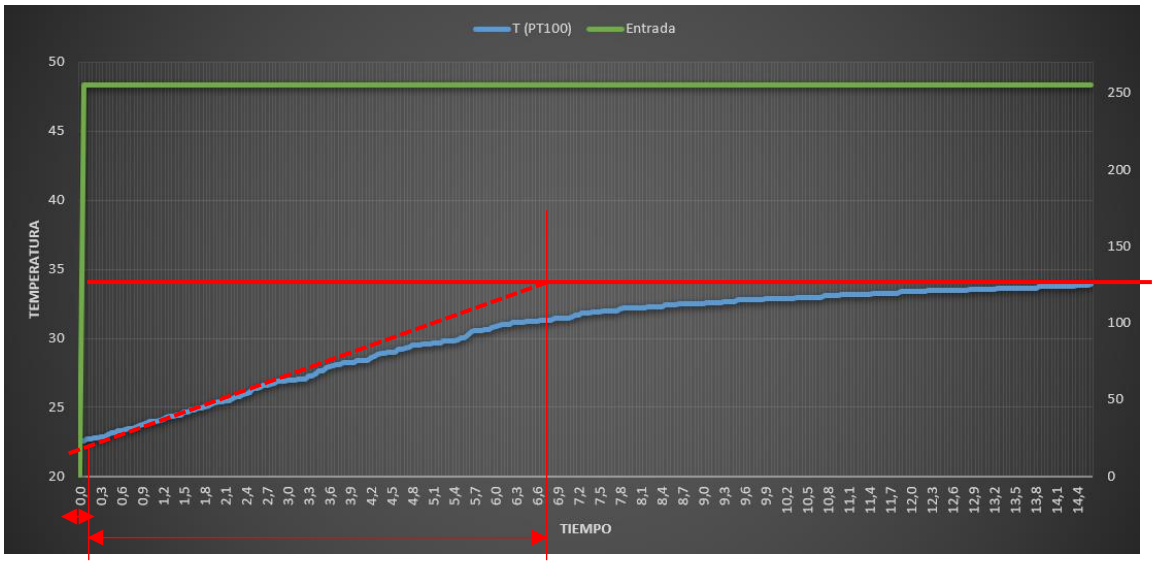
Esta librería nos permite obtener directamente la temperatura del sensor infrarrojo, de manera similar a la librería anterior, ambas desarrolladas por Sparkfun. Podemos obtener más información en la siguiente página.

https://github.com/adafruit/Adafruit-MLX90614-Library

8.2.ANEXO 2: CÁLCULO DE PARÁMETROS DE PID

Adjunto, tras la obtención de la prueba diferentes veces, el cálculo final de los parámetros de PID para los sensores utilizados.

SENSOR 1: **PT100**



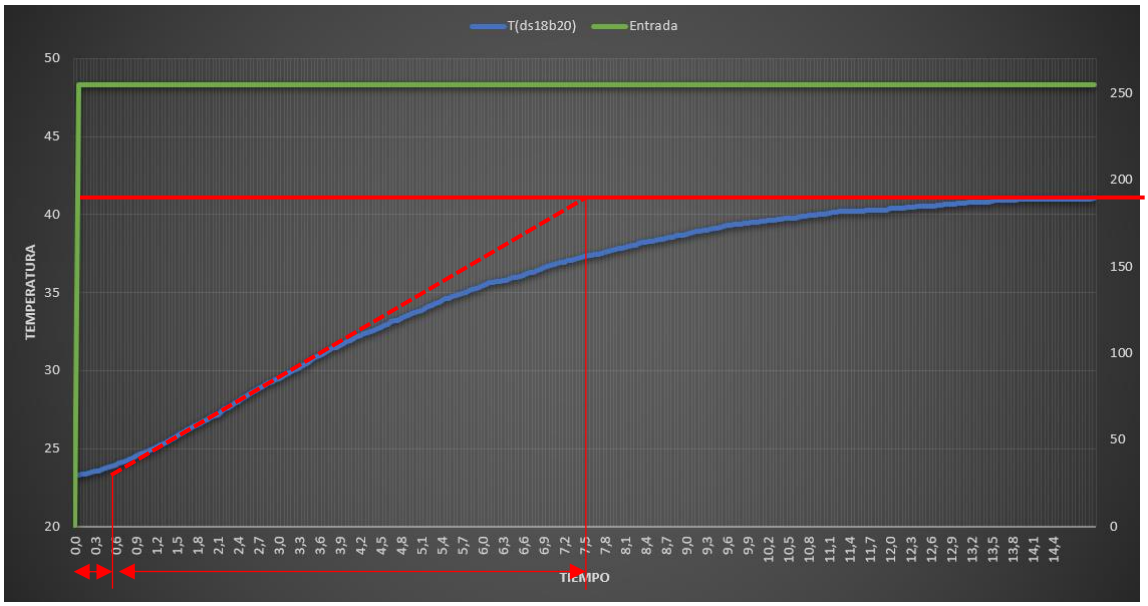
Parámetros de Ziegler-Nichols:

- L = 0.1
- T = 6.7

Calculamos ahora los parámetros en función de la tabla de Ziegler-Nichols:

K_p	**80.4**
T_I	**0.2**
T_D	**0.05**

SENSOR 2: **DS18B20**



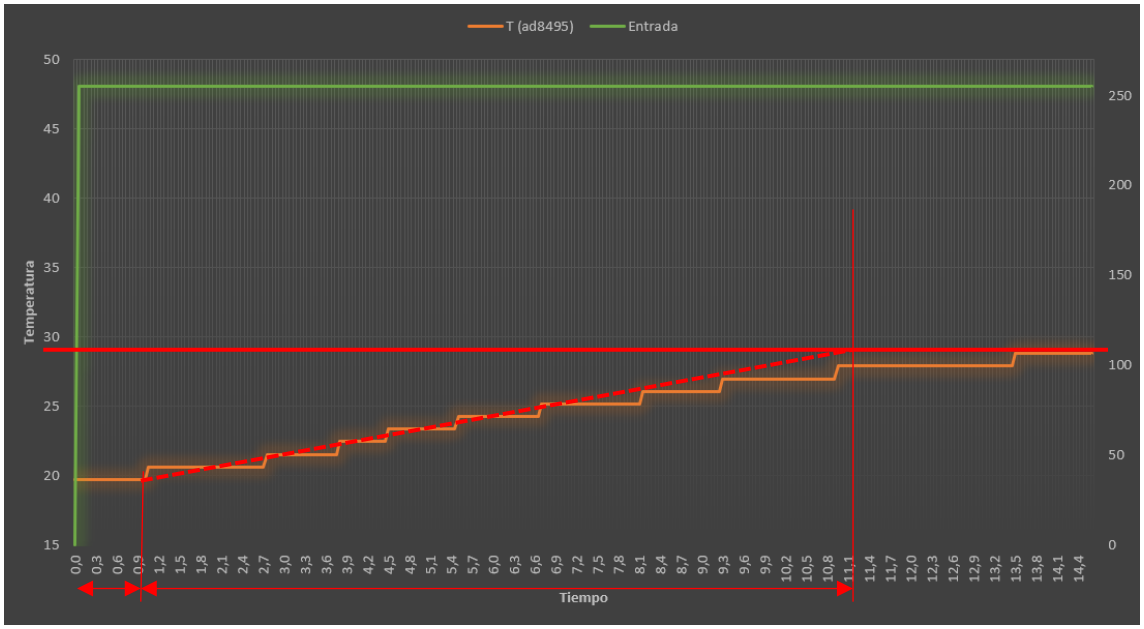
Parámetros de Ziegler-Nichols:

- L = 0.5
- T = 7

Calculamos ahora los parámetros en función de la tabla de Ziegler-Nichols:

K_p	**16.8**
T_I	**1**
T_D	**0.25**

SENSOR 3: **AD8495**



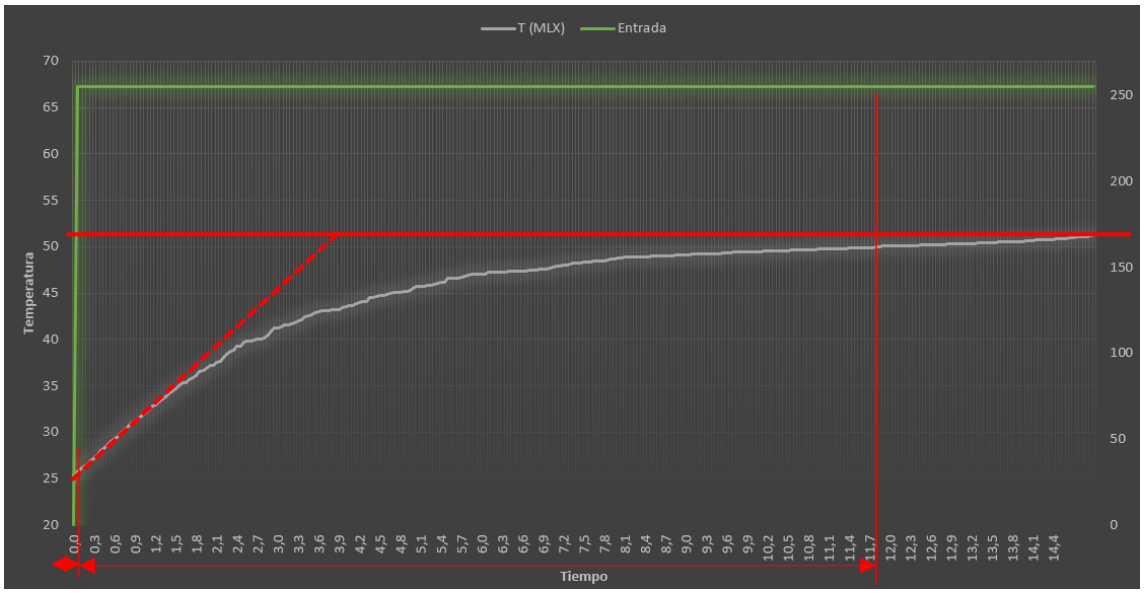
Parámetros de Ziegler-Nichols:

- L = 0.9
- T = 10.2

Calculamos ahora los parámetros en función de la tabla de Ziegler-Nichols:

K_p	**13.6**
T_I	**1.8**
T_D	**0.45**

SENSOR 4: **MLX90614**



Parámetros de Ziegler-Nichols:

- L = 0.05
- T = 11.75

Calculamos ahora los parámetros en función de la tabla de Ziegler-Nichols:

K_p	**282**
T_I	**0.1**
T_D	**0.025**

SENSOR 5: **NTC**

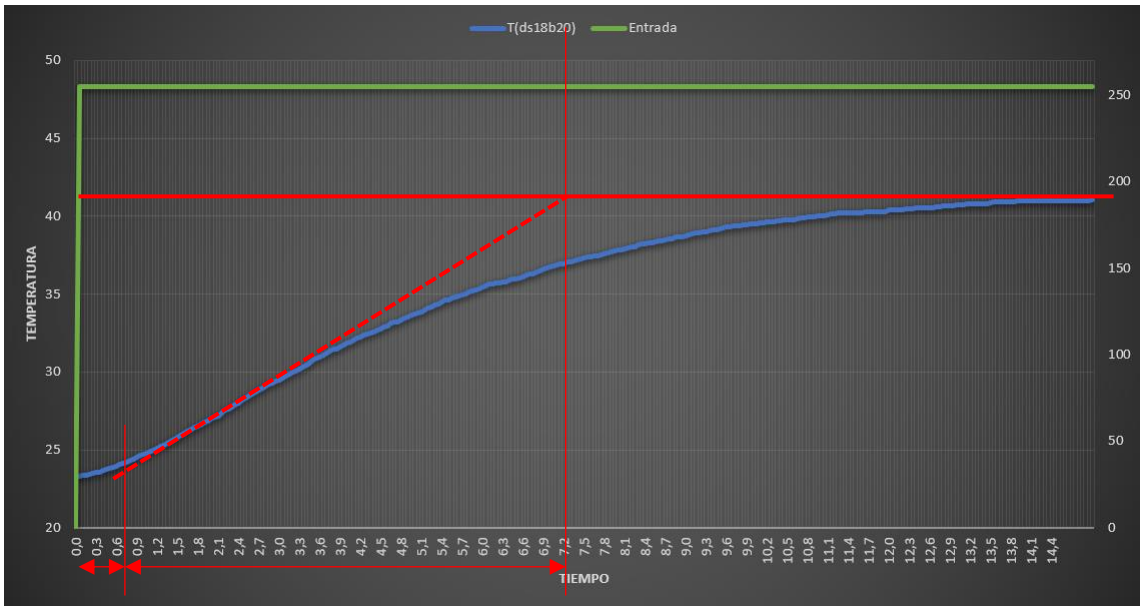
Parámetros de Ziegler-Nichols:

- L = 1.6
- T = 8

Calculamos ahora los parámetros en función de la tabla de Ziegler-Nichols:

K_p	**6**
T_I	**3.2**
T_D	**0.8**

SENSOR 6: **DS18B20**



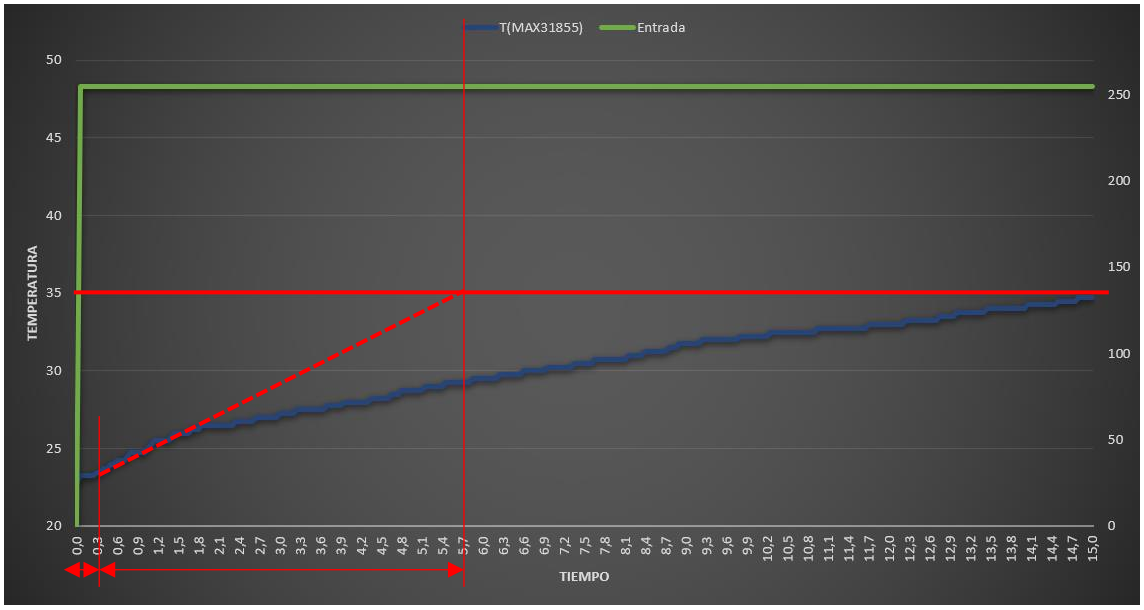
Parámetros de Ziegler-Nichols:

- L = 0.7
- T = 6.5

Calculamos ahora los parámetros en función de la tabla de Ziegler-Nichols:

K_p	**11.14**
T_I	**1.4**
T_D	**0.35**

SENSOR 7: **MAX31855**



Parámetros de Ziegler-Nichols:

- L = 0.3
- T = 5.4

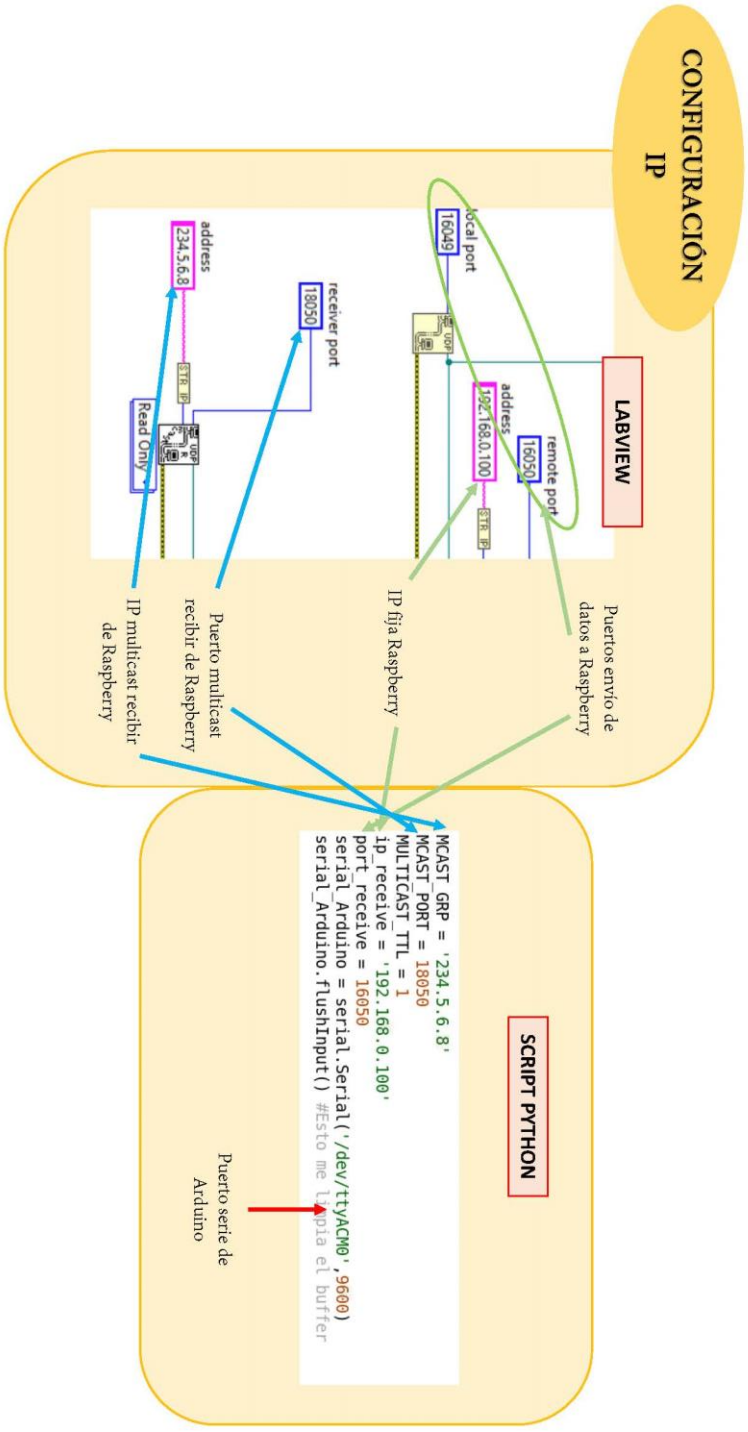
Calculamos ahora los parámetros en función de la tabla de Ziegler-Nichols:

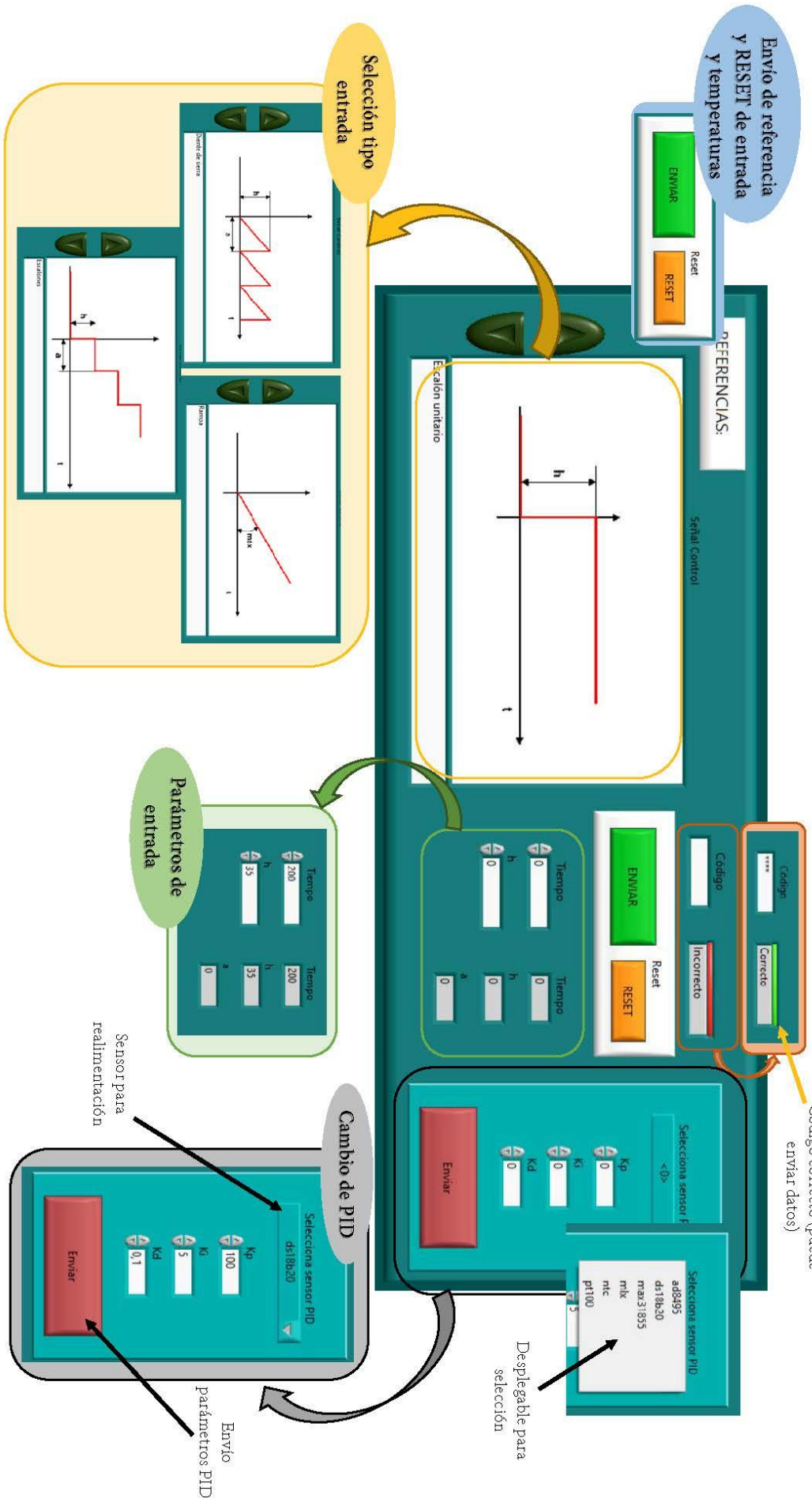
K_p	**21.6**
T_I	**0.6**
T_D	**0.15**

Como se puede apreciar en las gráficas cada uno de los sensores tiene una respuesta totalmente distinta y nos permite llegar a una temperatura diferente. Para dicho experimento se ha colocado la resistencia bobinada de USB proporcionada por el tutor, con una fuente de 5W trabajando al máximo y los sensores colocados en la misma situación. El tiempo se estipula en minutos.

8.3.ANEXO 3: MANUAL PANEL LABVIEW

Adjunto un manual gráfico de la utilización del panel de LabView, aunque es bastante sencillo:




Envío de referencia y RESET de entrada y temperaturas
ENVIAR
Reset
RESET
REFERENCIAS:
Selección tipo entrada
Señal Control
Escalón unitario
Código
****
Correcto
Incorrecto
Código correcto (puede enviar datos)
Parámetros de entrada
Tiempo
Cambio de PID
Selecciona sensor PID
ds18b20
Kp
Ki
Kd
Enviar
Sensor para realimentación
Selecciona sensor PID
ad8495
ds18b20
max31855
mlx
ntc
pt100
Desplegable para selección
Envío parámetros PID

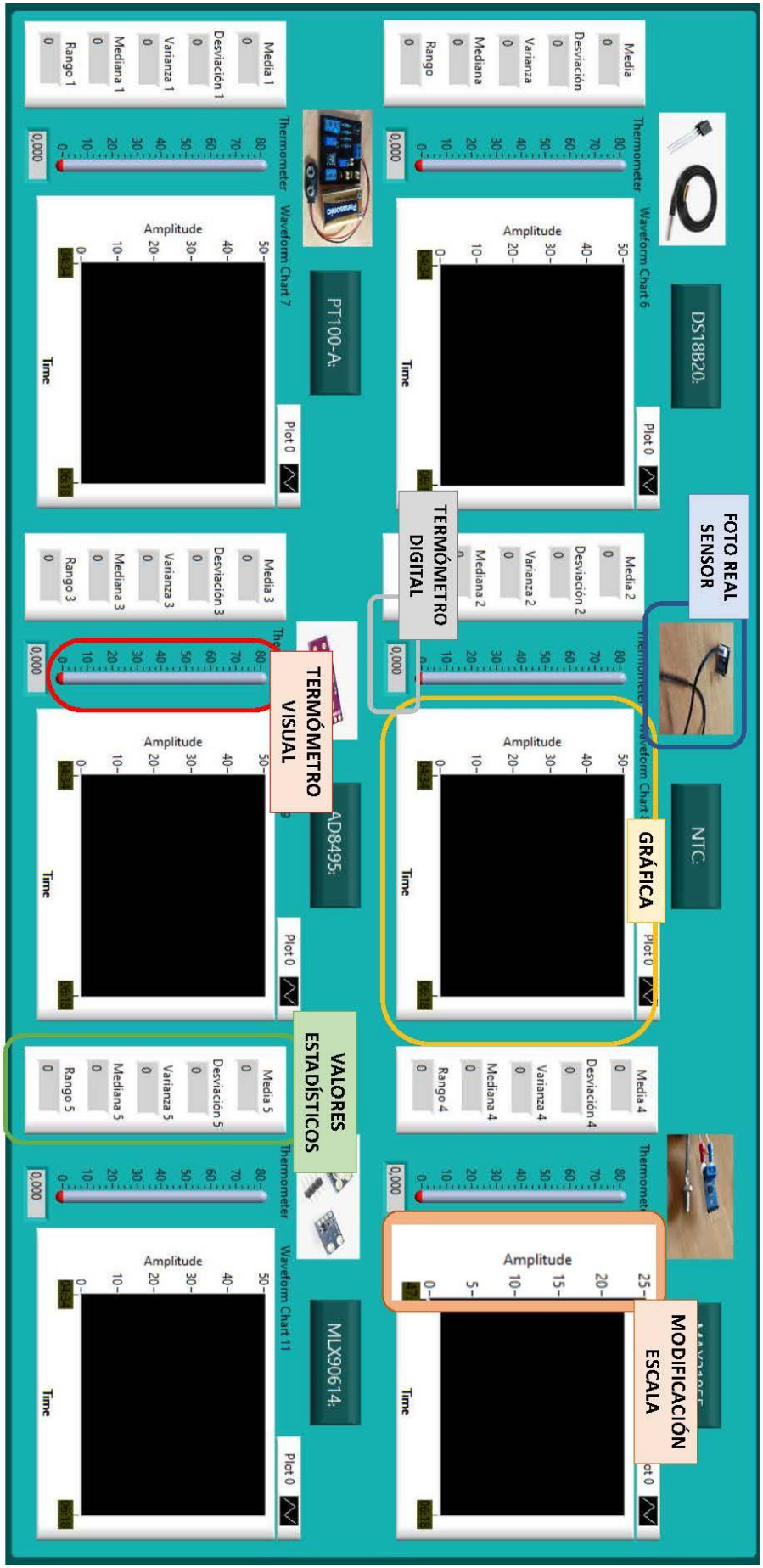
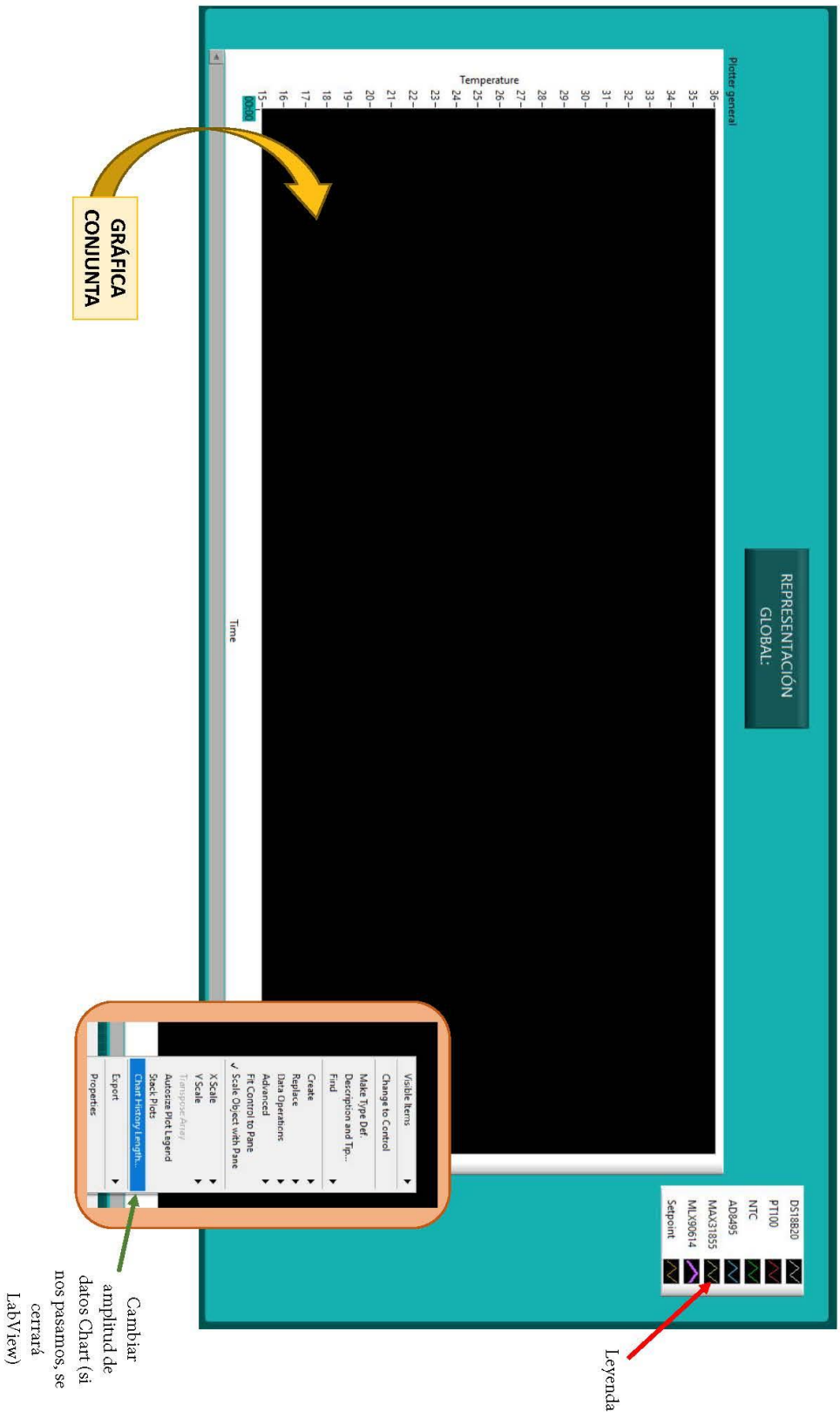
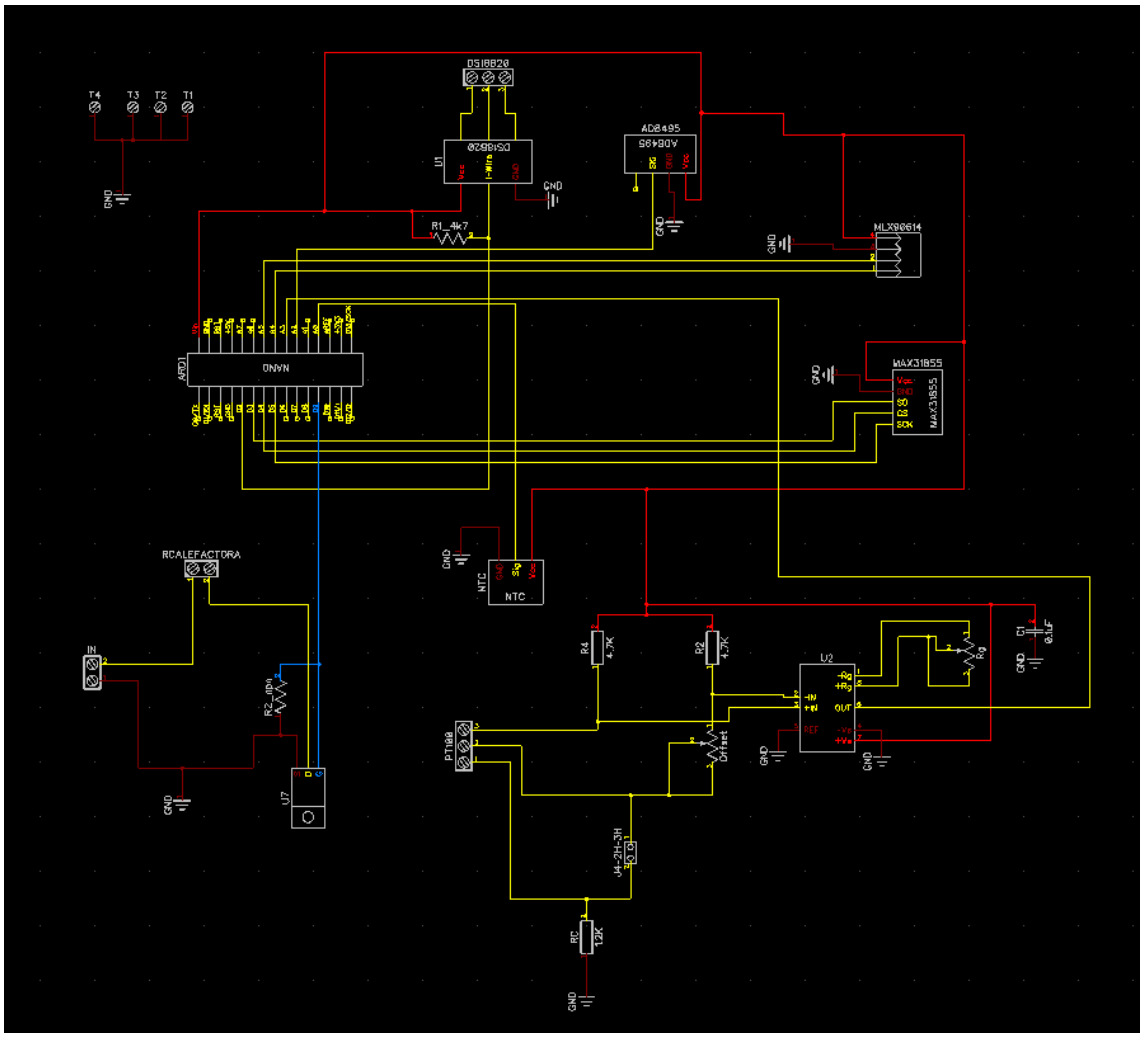


FOTO REAL SENSOR
TERMÓMETRO DIGITAL
GRÁFICA
VALORES ESTADÍSTICOS
TERMÓMETRO VISUAL
MODIFICACIÓN ESCALA
DS18B20:
PT100-A:
NTC:
AD8495:
MLX90614:
Media
Desviación
Varianza
Mediana
Rango
Thermometer
Waveform Chart
Amplitude
Time
Plot 0


REPRESENTACIÓN GLOBAL:
Plotter general
Temperature
Time
GRÁFICA CONJUNTA
DS18B20
PT100
NTC
AD8495
MAX31855
MLX90614
Setpoint
Leyenda
Visible Items
Change to Control
Make Type Def.
Description and Tip...
Find
Create
Replace
Data Operations
Advanced
Fit Control to Pane
Scale Object with Pane
X Scale
Y Scale
Transpose Array
Autosize Plot Legend
Stack Plots
Chart History Length...
Export
Properties
Cambiar amplitud de datos Chart (si nos pasamos, se cerrará LabView)

8.4. ANEXO 4: MONTAJE EN PCB:

Una vez desarrollado y comprobado que nuestro sistema funciona correctamente podríamos realizar un PCB para dicha utilidad que nos permita trabajar de una manera más profesional aprovechando los conocimientos recibidos sobre la ordenanza y montaje de dichas placas, aunque perderíamos en versatilidad. Para ello utilizaremos el programa *Diptrace*, que nos permite realizar un PCB con gran ayuda a partir del esquemático que nosotros mismos le subvencionamos. A continuación, se adjunta el posible esquemático:



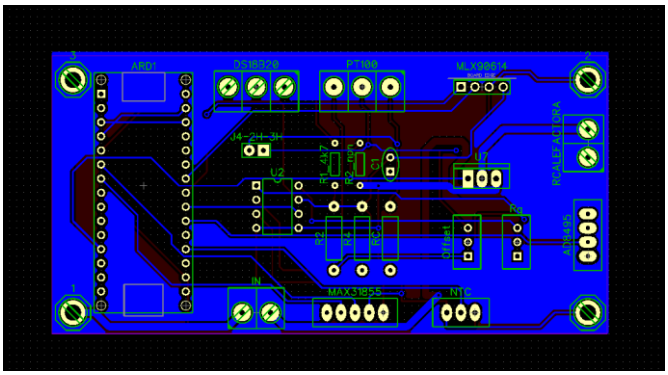
En dicho esquemático podemos ver en la zona inferior izquierda el módulo de calentamiento del que previamente hemos hablado en el apartado 3.7, además de cada uno de los sensores que irán pinchados en la placa, excepto la pt100 analógica que irá integrada en dicha placa.

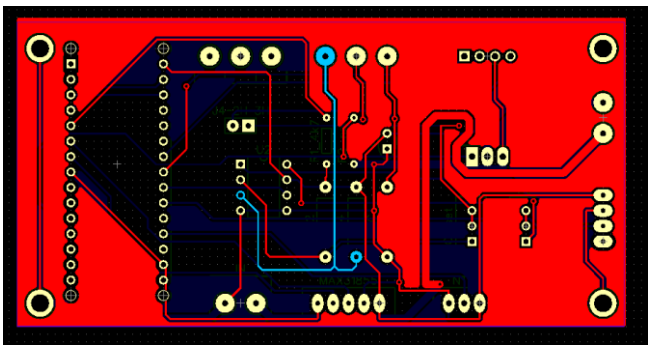
Tras realizar y comprobar el esquemático, pasamos a la parte de creación del PCB, en la que realizaremos una reestructuración y comprobaremos las pistas para evitar que

podamos tener problemas de electromagnetismo con los picos, que podamos pinchar los componentes sin que ninguno de ellos tenga problema de espacio, además de ampliar aquellas pistas que vayan a ser atravesadas por una mayor corriente siguiendo la siguiente estimación:

Ancho de pista en (Inches)-(mm)	Intensidad en A
0.010 inch - 0.254 mm	0.3A
0.015 inch - 0.381 mm	0.4A
0.020 inch - 0.508 mm	0.7A
0.025 inch - 0.635 mm	1A
0.050 inch - 1.27 mm	2A
0.100 inch - 2.54 mm	4A
0.150 inch - 3.8 mm	6A

El diseño de PCB ajustados los parámetros quedan de la siguiente manera:

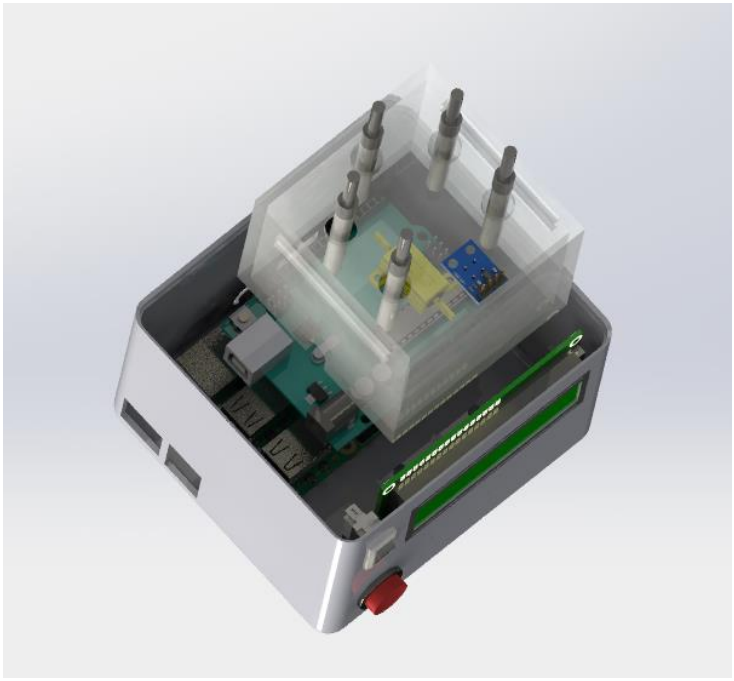




Una vez hecho todo, si quisiésemos que una empresa externa nos fabrique el PCB debemos de extraer el Gerber y encargárselo a una empresa introduciendo ciertos parámetros en su página web, así como grosor, color, etc. Si todo se introduce correctamente, no nos dará error y en un tiempo estimado de fabricación y envío tendremos la placa lista para introducir los componentes y soldarla.

8.5.ANEXO 5: DESARROLLO DE MAQUETA DE PROYECTO:

Desarrollé con la intención de llevarlo a cabo una maqueta para permitir el empaquetado de todo el sistema para su disposición para las prácticas futuras del alumnado, a falta de introducir la *protoshield* y sensores. Se adjuntan dos imágenes con y sin tapa:



9. REFERENCIAS:

[1] «El Internet de las cosas III: Monitor de Temperatura |», *Booleanbite*. [En línea]. Disponible en: https://booleanbite.com/web/el-internet-de-las-cosas-iii-monitor-de-temperatura/.

[2] «Revista ElectroIndustria - Sensores Inteligentes». [En línea]. Disponible en: http://www.emb.cl/electroindustria/articulo.mvc?xid=496&edi=10&xit=sensores-inteligentes.

[3] J. Sobota, R. PiŜl, P. Balda, y M. Schlegel, «Raspberry Pi and Arduino boards in control education», *IFAC Proc. Vol.*, vol. 46, n.º 17, pp. 7-12, ene. 2013, doi: 10.3182/20130828-3-UK-2039.00003.

[4] «Temperatura y humedad remota por IP. Envío de alarmas por email.» [En línea]. Disponible en: http://www.ditecom.com/monitorizacion_IP/control-temperatura-por-ip.shtml.

[5] «MQTT». [En línea]. Disponible en: http://mqtt.org/.

[6] L. López Brizuela, «Empleo de la plataforma Arduino para el desarrollo de actividades prácticas relacionadas con las mediciones», Thesis, Universidad Central «Marta Abreu» de Las Villas. Facultad de Ingeniería Eléctrica. Departamento de Automática y Sistemas Computacionales, 2017.

[7] B. Orozco y C. José, «Desarrollar e implementar una arquitectura de red basada en IoT, que permita la recopilación de información a través de sensores de temperatura.», nov. 2016.

[8] S. Ferdoush y X. Li, «Wireless Sensor Network System Design Using Raspberry Pi and Arduino for Environmental Monitoring Applications», *Procedia Comput. Sci.*, vol. 34, pp. 103-110, 2014, doi: 10.1016/j.procs.2014.07.059.

[9] M. S. Soliman, A. A. Alahmadi, A. A. Maash, y M. O. Elhabib, «Design and Implementation of a Real-Time Smart Home Automation System Based on Arduino Microcontroller Kit and LabVIEW Platform», vol. 12, n.º 18, p. 6, 2017.

[10] O. W. Chuan y S. H. Ruslan, «Medical warehouse monitoring and control system using LabVIEW», en *2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)*, 2016, pp. 2396-2401, doi: 10.1109/ICEEOT.2016.7755123.

[11] «Arduino - Introduction». [En línea]. Disponible en: https://www.arduino.cc/en/Guide/Introduction.

[12] «Arduino Uno», *Wikipedia, la enciclopedia libre*. 06-mar-2020.

[13] «Raspberry Pi», *Wikipedia, la enciclopedia libre*. 13-mar-2020.

[14] «Raspberry Pi», *Wikipedia, la enciclopedia libre*. 13-mar-2020.

[15] «Diferencias entre UDP y TCP», *PC Solución*, 04-abr-2018. [En línea]. Disponible en: https://pc-solucion.es/2018/04/04/diferencias-entre-udp-y-tcp/.

[16] «TcpCommunication - Python Wiki». [En línea]. Disponible en: https://wiki.python.org/moin/TcpCommunication.

[17] «UdpCommunication - Python Wiki». [En línea]. Disponible en: https://wiki.python.org/moin/UdpCommunication.

[18] «Firebase», *Firebase*. [En línea]. Disponible en: https://firebase.google.com/docs/libraries?hl=es-419.

[19] «¿Qué es Firebase Realtime Database?», *Ascenso*. [En línea]. Disponible en: https://ascenso.org/categoria/actualidad-digital/que-es-firebase-realtime-database/.

[20] «Firebase RealTime Database with Operations in Android with Examples», *GeeksforGeeks*, 27-jun-2019. [En línea]. Disponible en:

https://www.geeksforgeeks.org/firebase-realtime-database-with-operations-in-android-with-examples/.

[21] «firebase_admin.auth module». [En línea]. Disponible en: https://firebase.google.com/docs/reference/admin/python/firebase_admin.auth?hl=es-419.

[22] J. R. L. Vizcaíno y J. P. Sebastiá, *LabVIEW: Entorno gráfico de programación*. Marcombo, 2011.

[23] «10 clones de Arduino que no conocías | robologs». [En línea]. Disponible en: https://robologs.net/2013/12/03/10-clones-de-arduino-que-no-conocias/.

[24] I. PE, «▷ Comparativa de las placas Raspberry Pi y competencia», *ComoHacer.eu » ¿Inventamos juntos?*, 12-ago-2014. [En línea]. Disponible en: https://comohacer.eu/comparativa-y-analisis-raspberry-pi-vs-competencia/.

[25] designthemes, «Raspberry Pi 3 | Tienda y Tutoriales Arduino». [En línea]. Disponible en: https://www.prometec.net/indice-raspberry-pi/.

[26] «IOT: Comunicación Raspberry PI con Arduino», *Booleanbite*. [En línea]. Disponible en: https://booleanbite.com/web/internet-de-las-cosas-1-conexion-raspberry-arduino/.

[27] jecrespom, «Proyecto – Grabar datos de temperatura (Datalogger)», *Aprendiendo Arduino*, 08-jul-2016. [En línea]. Disponible en: https://aprendiendoarduino.wordpress.com/2016/07/08/proyecto-grabar-datos-de-temperatura-datalogger/.

[28] «Arduino y el termómetro infrarrojo a distancia MLX90614». [En línea]. Disponible en: https://www.luisllamas.es/arduino-y-el-termometro-infrarrojo-a-distancia-mlx90614/.

[29] «El bus I2C en Arduino». [En línea]. Disponible en: https://www.luisllamas.es/arduino-i2c/.

[30] «MAX31855 Thermocouple», *Adafruit Learning System*. [En línea]. Disponible en: https://learn.adafruit.com/thermocouple/arduino-code.

[31] «El bus SPI en Arduino». [En línea]. Disponible en: https://www.luisllamas.es/arduino-spi/.

[32] «MAX6675. Conversor AD SPI para sonda de termopar K.», *polaridad.es*, 27-jul-2016. [En línea]. Disponible en: https://polaridad.es/max6675-termopar-k-thermocouple-temperatura-compensacion-union-fria-spi-arduino/.

[33] Unknown, «Sensors Tutorials with Arduino: Arduino-Analog K-Type Thermocouple Amplifier-AD8495 Breakout», *Sensors Tutorials with Arduino*, 08-nov-2014. [En línea]. Disponible en: http://sensors-tutorials-arduino.blogspot.com/2014/11/arduino-analog-k-type-thermocouple.html.

[34] «AD8495 Analog Output K-Type Thermocouple Amplifier», *Adafruit Learning System*. [En línea]. Disponible en: https://learn.adafruit.com/ad8495-thermocouple-amplifier/arduino.

[35] «Adafruit MAX31865 RTD PT100 or PT1000 Amplifier», *Adafruit Learning System*. [En línea]. Disponible en: https://learn.adafruit.com/adafruit-max31865-rtd-pt100-amplifier/arduino-code.

[36] «Medir temperatura con Arduino y termistor (MF52)», *Luis Llamas*. [En línea]. Disponible en: https://www.luisllamas.es/medir-temperatura-con-arduino-y-termistor-mf52/.

[37] «Medir temperatura de líquidos y gases con Arduino y DS18B20», *Luis Llamas*. [En línea]. Disponible en: https://www.luisllamas.es/temperatura-liquidos-arduino-ds18b20/.

[38] «Guidelines for Reliable Long Line 1-Wire Networks». [En línea]. Disponible en: https://www.maximintegrated.com/en/design/technical-documents/tutorials/1/148.html.
[39] «Wiring 1-Wire Devices», *ENEN Loxone Smart Home*. [En línea]. Disponible en: https://www.loxone.com/enen/kb/wiring-1-wire-devices/.
[40] «Controlar grandes cargas con Arduino y transistor MOSFET». [En línea]. Disponible en: https://www.luisllamas.es/arduino-transistor-mosfet/.
[41] «Revista Ciencia, Ingeniería y Desarrollo Tec Lerdo», p. 6, 2018.
[42] «Método de Ziegler-Nichols - Control Automático - Picuino». [En línea]. Disponible en: https://www.picuino.com/es/arduprog/control-ziegler-nichols.html.
[43] Redacción, «Cómo hacer un PID con Arduino y cómo implementar este tipo de control», *Arduino, Genuino, Raspberry Pi. Noticias y proyectos.*, 10-may-2019. [En línea]. Disponible en: https://descubrearduino.com/como-hacer-un-pid-con-arduino/.
[44] «Etching Pathways: LabVIEW and Raspberry Pi TCP/IP Communications». [En línea]. Disponible en: http://etchingpathways.blogspot.com/2013/09/labview-and-raspberry-pi-tcpip.html.
[45] *firebase/firebase-admin-python*. Firebase, 2020.
[46] «Conceptos: Introducción a JSON | Brightcove Aprendizaje». [En línea]. Disponible en: https://support.brightcove.com/es/concepts-introducing-json.
[47] O. Campos, «Multiprocesamiento en Python: Threads a fondo, introducción», *Genbeta*, 22-sep-2011. [En línea]. Disponible en: https://www.genbeta.com/desarrollo/multiprocesamiento-en-python-threads-a-fondo-introduccion.
[48] «COMO hacer Multicast sobre TCP/IP: Explicación del Multicast.» [En línea]. Disponible en: http://web.dit.upm.es/~jmseyas/linux/mcast.como/Multicast-Como-2.html.
[49] «Tutorial LCD con I2C, controla un LCD con solo dos pines». [En línea]. Disponible en: https://www.naylampmechatronics.com/blog/35_Tutorial--LCD-con-I2C-controla-un-LCD-con-so.html.